Red Hat Build of OptaPlanner 8.38

使用红帽构建的 OptaPlanner 开发解决方案

Last Updated: 2023-07-25

Red Hat Build of OptaPlanner 8.38 使用红帽构建的 OptaPlanner 开发解决方案

法律通告

摘要

本文档论述了如何使用红帽构建的 OptaPlanner 开发解决方案，以查找规划问题的最佳解决方案。

目录

前言

您可以使用红帽构建的 OptaPlanner 开发可决定规划问题的最佳解决方案。OptaPlanner 是 Red Hat Build of OptaPlanner 的内置组件。您可以使用 solvers 作为 Red Hat Build of OptaPlanner 的服务的一部分，来根据特定限制优化有限的资源。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 *CTO Chris Wright 信息*。

部分 I. RED HAT BUILD OF OPTAPLANNER 8.38 发行注记

本发行注记列出了新功能，并为 Red Hat Build of OptaPlanner 8.38 提供升级说明。

第 1 章 从 OPTAPLANNER 8.13 升级到 RED HAT BUILD OF OPTAPLANNER 8.38

要从 OptaPlanner 8.13 升级到 Red Hat Build of OptaPlanner 8.38，请按照以下顺序合并 OptaPlanner 的早期版本：

- 从 8.13.0.Final 到 8.14.0.Final
- 从 8.19.0.Final 到 8.20.0.Final
- 从 8.22.0.Final 到 8.23.0.Final
- 从 8.27.0.Final 到 8.28.0.Final
- 从 8.28.0.Final 到 8.29.0.Final
- 从 8.31.0.Final 到 8.32.0.Final
- 从 8.33.0.Final 到 8.34.0。最后
- 从 8.36.0.Final 到 8.37.0.Final

流程

1. 在浏览器中打开 OptaPlanner Upgrade Recipe 8 页面。
2. 完成您要升级的第一个版本的说明，如 **From 8.13.0.Final 到 8.14.0.Final**。
3. 重复说明，直到您已升级到 8.37.0.Final。

第 2 章 红帽构建的 OPTAPLANNER 8.38 新功能

本节重点介绍红帽构建的 OptaPlanner 8.38 中的新功能。

注意

Bavet 是用于快速分数计算的功能。Bavet 目前仅在 OptaPlanner 的社区版本中可用。红帽构建的 OptaPlanner 8.38 不提供它。

2.1. PILLAR MOVE 和 NEARBY 选择的性能改进

现在，OptaPlanner 可以自动探测到多个 pillar 缓存的情况，多个 pillar 缓存可以共享预计算的 pillar 缓存，而不是为每个移动选择器重新计算 pillar 缓存。例如，如果您组合了 pillar move，例如 **PillarChangeMove** 和 **PillarSwapMove**，您应该会看到更高的性能。

如果您使用 nearby 选择，这也适用。现在，OptaPlanner 可以自动检测预计算的距离列表，可在多个移动选择器之间共享，从而节省内存和 CPU 处理时间。

因此，以下接口的实现应该是无状态的：

- **org.optaplanner.core.impl.heuristic.selector.common.nearby.NearbyDistanceMeter**
- **org.optaplanner.core.impl.heuristic.selector.common.decorator.SelectionFilter**
- **org.optaplanner.core.impl.heuristic.selector.common.decorator.SelectionProbabilityWeightFactory**
- **org.optaplanner.core.impl.heuristic.selector.common.decorator.SelectionSorter**
- **org.optaplanner.core.impl.heuristic.selector.common.decorator.SelectionSorterWeightFactory**

通常，如果 solver 配置要求用户实施接口，则预期的是实施是无状态的，也不会尝试包含外部状态。通过这些性能改进，无法遵循此要求将导致微小的错误和分数损坏，因为解决者现在会在看到适合的情况下重复使用这些实例。

2.2. OPTAPLANNER 配置改进

EntitySelectorConfig 和 **ValueSelectorConfig** 等各种配置类包含新的构建程序方法，以便更轻松地将基于 XML 的解析器配置替换为流畅的 Java 代码。

2.3. 对 K-OPT MOVES 的 PLANNINGLISTVARIABLE 支持

添加了列表变量 **KOptListMoveSelector** 的新移动选择器。**KOptListMoveSelector** 选择一个实体，从路由中删除 **k** edges，并从删除的边缘端点中添加 **k** new edges。**KOptListMoveSelector** 可以帮助 solver escape local optima in vehicle routing 问题。

2.4. SOLUTIONMANAGER 支持更新 SHADOW 变量

SolutionManager（以前为 **ScoreManager**）方法（如 **解释(solution) 和更新(solution)**）收到带有额外参数 solution **UpdatePolicy** 的新过载。这对从持久性存储（如关系数据库）加载其解决方案的用户很有用，其中这些解决方案不包括由影子变量或分数传输的信息。通过调用这些新的过载并选择正确的策略，OptaPlanner 会自动计算解决方案中的所有 shadow 变量的值，或者重新计算分数或两者。

同样，**ProblemChangeDirector** 收到一个名为 **updateShadowVariables （）** 的新方法，以便您可以在实时计划中按需更新影子变量。

2.5. 值范围自动检测

在大多数情况下，现在可以自动检测规划变量和值范围之间的链接。因此，**@ValueRangeProvider** 不再需要提供 ID 属性。同样，规划变量不再需要通过 **valueRangeProviderRefs** 属性引用值范围 provider。

不需要更改代码或配置更改。比 brevity 更清楚的用户可以继续显式引用值范围提供程序。

部分 II. 红帽构建的 OPTAPLANNER 入门

作为业务规则开发人员，您可以使用红帽构建的 OptaPlanner 来查找根据一组有限资源和特定限制规划问题的最佳解决方案。

使用本文档开始使用 OptaPlanner 开发解决方案。

第 3 章 红帽构建的 OPTAPLANNER 简介

OptaPlanner 是一个轻量级、可嵌入的规划引擎，可优化计划问题。它有助于更有效地解决规划问题，并将优化 heuristics 和 metaheuristics 与非常有效的分数计算相结合。

例如，OptaPlanner 帮助解决各种用例：

- *employee/Patient Rosters* ：这有助于为 nurses 创建时间表，并跟踪病人人的管理。
- *教育*时间表：它有助于安排课程、课程、考试和会议演示。
- *Shop Schedules* ：它跟踪 car assembly 行、机器队列规划和工作人员任务规划。
- *Cutting Stock* ：通过减少纸张和钢材等消耗来最小化浪费。

每个机构都面临规划问题；也就是说，它们为产品和服务提供有限的资源集合（员工、资产、时间和金）。

OptaPlanner 是 Apache 软件许可证 2.0 下的开源软件。它是 100% 纯 Java，可在大多数 Java 虚拟机(JVM)上运行。

3.1. 后向兼容性

OptaPlanner 分隔 API 和实现：

- **公共 API** ：软件包命名空间 **org.optaplanner.core.api**、**org.optaplanner.benchmark.api**、**org.optaplanner.test.api** 和 org.optaplanner.persistence.api 和 **org.optaplanner.persistence.api** 中的 100% 在将来的次版本和补丁版本中向后兼容。在个别情况下，如果主版本号有变化，一些特定的类可能会有一些向后兼容的更改，但这些更改将在升级方法中明确记录。https://www.optaplanner.org/download/upgradeRecipe/
- **XML 配置** ：XML 解析器配置对所有元素向后兼容，但需要使用非公共 API 类的元素除外。XML solver 配置由软件包命名空间中的 **org.optaplanner.core.config** 和 **org.optaplanner.benchmark.config** 中定义的类定义。
- **实施类** ：所有其他类 *不向后兼容*。它们将在以后的主发行版本或次版本中改变。升级方法描述了相关的更改，以及如何在升级到更新的版本时解决它们。

3.2. 规划问题

根据有限资源和特定限制，*规划问题* 具有最佳目标。最佳目标可以是任意数量，例如：

- 利润最大化 - 最佳目标导致可能最高的利润。
- 最小化资源占用量 - 最佳目标最小对环境的影响。
- 最大限度地提高员工或客户的满意度 - 最佳目标优先选择员工或客户的需求。

实现这些目标的能力取决于可用资源的数量。例如，以下资源可能会受限制：

- 人员数量
- 时间
- 预算

- 物理资产，如 machinery、vehicles、计算机、构建

您还必须考虑与这些资源相关的特定限制，如个人工作小时数、使用某些机器或设备间的兼容性。

红帽构建的 OptaPlanner 帮助 Java 编程人员有效地解决约束性问题。它将优化 heuristics 和 metaheuristics 与高效的分数计算相结合。

3.3. 规划问题中的 NP-COMPLETENESS

提供的用例 *可能是* NP-complete 或 NP-hard，这意味着适用以下语句：

- 可轻松验证特定解决方案以合理时间的问题。
- 在合理时间查找问题的最佳解决方案没有简单的方法。

含义是解决您的问题可能比您预计的难度更高，因为这两种常见技术并不易受：

- 静默的强制算法（即使是更高级的变体）用时过长。
- 例如，一个快速算法（例如在 bin packing 问题 *中），首先考虑最大的项目* 会返回最佳解决方案。

通过使用高级优化算法，OptaPlanner 在合理时间内找到适合此类规划问题的良好解决方案。

3.4. 规划问题的解决方案

计划问题有很多解决方案。

多种解决方案类别是：

可能的解决方案

可能的解决方案是任何可能的解决方案，无论是是否会破坏任何限制。规划问题通常有大量可能的解决方案。其中许多解决方案都不有用。

可行的解决方案

可行的解决方案是不会破坏任何（负）硬限制的解决方案。可行的解决方案数量相对于可能的解决方案的数量。有时，没有可行的解决方案。每个可行的解决方案都是可能的解决方案。

最佳解决方案

最佳解决方案是具有最高分数的解决方案。规划问题通常有几个最佳解决方案。它们至少有一个最佳解决方案，即使没有可行的解决方案，并且最佳解决方案不可行。

找到最佳解决方案

最佳解决方案是在指定时间内实施的最高分数最高的解决方案。发现的最佳解决方案可能是可行的，只要有足够的时间，它是最佳解决方案。

因此，可能的解决方案数量是巨大的（如果正确计算），即使是小数据集。

在 **optaplanner-examples/src** 分发文件夹中提供的示例中，大多数实例都有大量可能的解决方案。由于无法保证找到最佳解决方案，因此任何实施都必须至少评估那些可能的解决方案的子集。

OptaPlanner 支持多种优化算法，以通过大量可能的解决方案来有效地实现。

根据用例，一些优化算法的性能比其他算法更好，但无法提前知道。使用 OptaPlanner，您可以在 XML 或代码的几行中更改 solver 配置来切换优化算法。

3.5. 规划问题的约束

通常，一个计划问题最少有两个限制：

- 不能中断 *（负）硬约束*。
  例如，一个教员无法同时查询两个不同的课程。
- 如果可以避免 *（负）软限制*，则不应中断它。
  例如，Teacher A 并不喜欢在周五的下午时公布。

有些问题也具有正的限制：

- 如果可能，应该 *实现正的软约束（或奖励）*。
  例如，Teacher B likestoonday mornings。

有些基本问题只存在硬限制。有些问题有 3 个或更多限制，如 hard、medium 和 soft 约束。

这些限制定义了规划问题 *的分数计算*（也称为适合性 *功能*）。计划问题的每个解决方案都使用分数进行评分。使用 OptaPlanner 时，分数限制使用面向对象的语言（如 Java）或 Drools 规则编写。

这种类型的代码非常灵活且可扩展的。

3.6. RED HAT BUILD OF OPTAPLANNER 提供的示例

Red Hat Build of OptaPlanner 提供了几个 OptaPlanner 示例。您可以查看代码以了解示例，并根据需要进行修改以满足您的需要。

注意

红帽不提供对 Red Hat Build of OptaPlanner 发行版中包含的示例代码的支持。

一些 OptaPlanner 示例解决了在学士学会的问题。以下表中的 **Contest** 列列出了 contests。它也识别一个示例为 *realistic* 或 *unrealistic* 用于 contest 的目的。 *真实的 contest* 是符合以下标准的官方独立测试：

- 明确定义的真实用例
- 实际限制
- 多个实际数据集
- 在特定硬件的特定时间限制内重复生成的结果
- 富士通和/或企业运营研究社区的积极参与。

真实证明证明可以为 OptaPlanner 提供目标比较，以及具有竞争性的软件和学研究。

表 3.1. 示例概述

示例	Domain	大小	候选版本	目录名称
n queens	1 个实体类 (1 变量)	实体 criu **256** Value ⇐ **256** 搜索空间 criu **10^616**	无点（可扩展）	**nqueens**

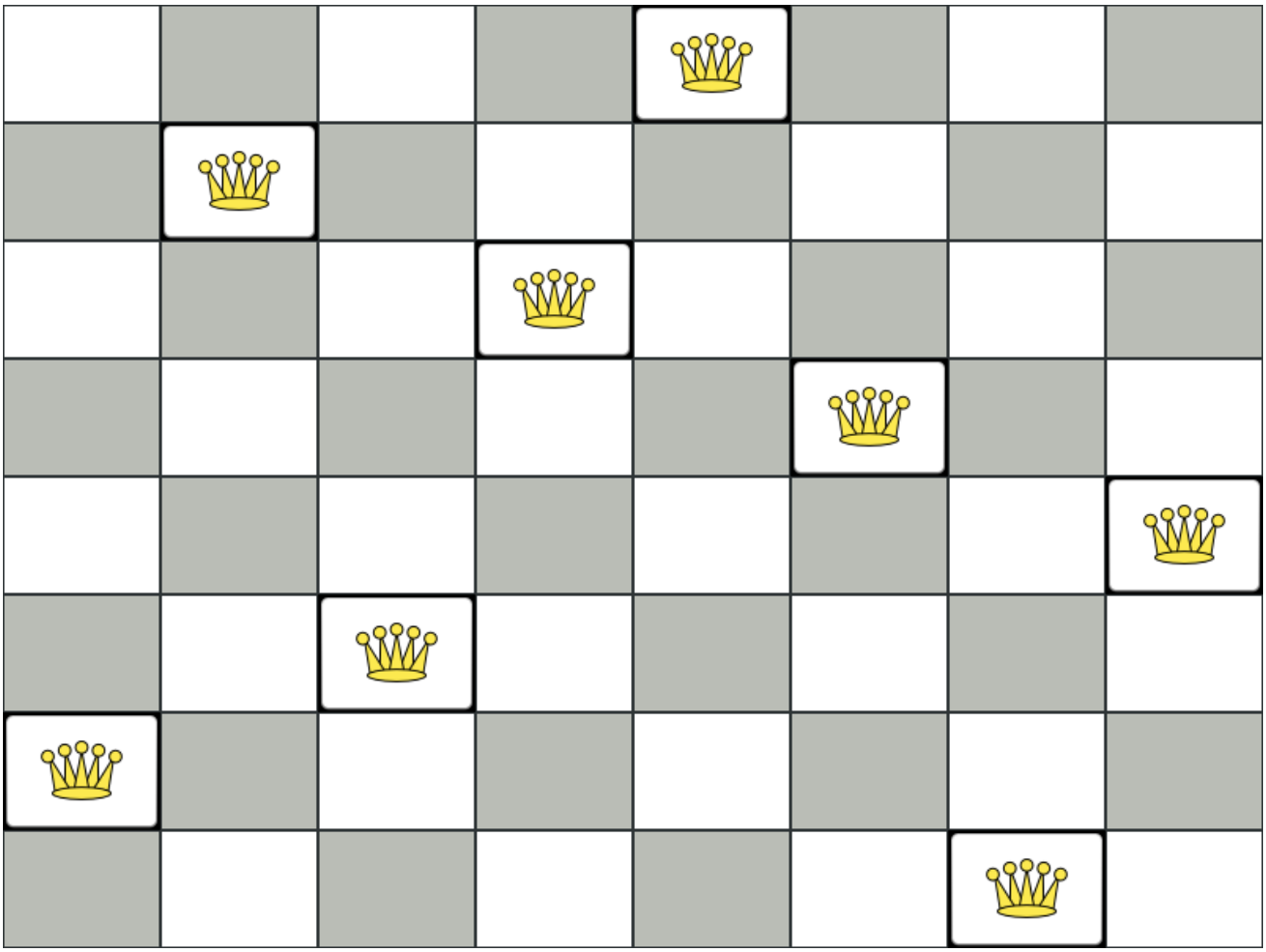
示例	Domain	大小	候选版本	目录名称
云平衡	1 个实体类 (1 变量)	实体 **2400** value criu **800** 搜索空间 wagon **10^6967**	否（由我们定义）	**cloudbalancing**
traveling salesman	1 个实体类 (1 个链变量)	实体 criu **980** value criu **980** 搜索空间 criu **10^2504**	不切实际的 TSP Web	**tsp**
Tennis club 调度	1 个实体类 (1 变量)	实体 **72** value criu **7** 搜索空间 wagon **10^60**	否（由我们定义）	**tennis**
满足调度	1 个实体类 (2 变量)	实体 criu **10** value criu **320** 和 criu **5** 搜索空间 criu **10^320**	否（由我们定义）	**满足调度**
course timetabling	1 个实体类 (2 变量)	实体 criu **434** value criu **25** 和 criu **20** Search space criu **10^1171**	真实的 ITC 2007 跟踪 3	**curriculumCourse**
机器重新分配	1 个实体类 (1 变量)	实体 criu **50000** value criu **5000** 搜索空间 wagon **10^184948**	2012 年真正现实 ROADEF	**machineReassignment**
vehicle 路由	1 个实体类 (1 个链变量) 1 个影子实体类 (1 个自动影子变量)	实体 **55** value criu **2750** Search space criu **10^8380**	不切实际的 VRP Web	**vehiclerouting**

示例	Domain	大小	候选版本	目录名称
带有时间窗的载体路由	所有 Vehicle 路由 (1 个影子变量)	实体 **55** value criu **2750** Search space criu **10^8380**	不切实际的 VRP Web	**vehiclerouting**
项目作业调度	1 个实体类 (2 变量) (1 个影子变量)	实体 **640** value **?** 和 iwl **?** 搜索空间 iwl **?**	2013 年现实 MISTA	**projectjobscheduling**
任务分配	1 个实体类 (1 列表变量) 1 个影子实体类 (1 个自动影子变量) (1 影子变量)	实体 criu **20** value criu **500** Search space categories **10^1168**	没有由我们定义	**taskassigning**
考试时间选项卡	2 个实体类（相同层次结构） (2 变量)	实体 criu **1096** value criu **80** 和 **49** Search space criu **10^3374**	真实的 ITC 2007 跟踪 1	**examination**
Nurse rostering	1 个实体类 (1 变量)	实体 **752** value criu **50** Search space criu **10^1277**	2010 年现实 INRC	**nurserostering**
traveling tournament	1 个实体类 (1 变量)	实体 **1560** value **78** 搜索空间 wagon **10^2301**	不切实际的 TTP	**travelingtournament**
会议调度	1 个实体类 (2 变量)	实体 **216** value criu **18** 和 iwl **20** 搜索空间 wagon **10^552**	没有由我们定义	**会议排期**

示例	Domain	大小	候选版本	目录名称
飞行人员的调度	1 个实体类 (1 变量) 1 个影子实体类 (1 个自动影子变量)	实体 **4375** value wagon **750** 搜索空间 criu **10^12578**	没有由我们定义	**flightcrewschedulin g**

3.7. N QUEENS

在一个 *n* 大小的象棋盘中放置 *n* 个皇后，没有两个皇后可以相互攻击。最常见的 *n* queens puzzle 是 8 个 queens puzzle，*n = 8* ：



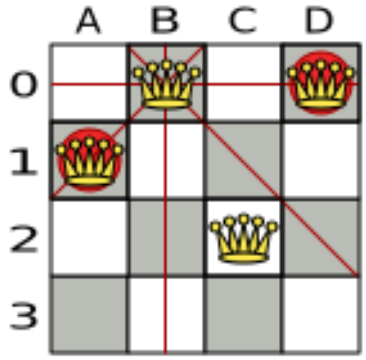
约束：

- 使用 *n* 列和 *n* 行的 chessboard。
- 将 *n* queens 放在板上。
- 没有两个排队可以相互攻击。queen 可以在同一横向、垂直或诊断行中攻击任何其他排队。

本文档主要使用 4 queens puzzle 作为主要示例。

建议的解决方案可以是：

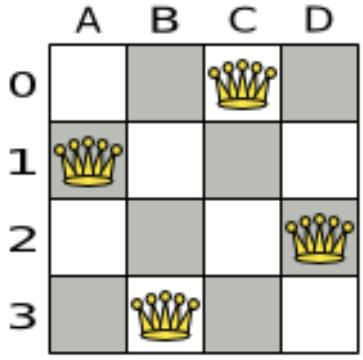
图 3.1. 错误解决方案，用于 4 queens puzzle



以上解决方案是错误的，因为 queens **A1** 和 **B0** 可以相互攻击（因此可以 queens **B0** 和 **D0**）。删除 queen **B0** 将遵循 "no two queens 可相互攻击"约束，但会破坏 "place *n* queens" 约束。

以下是正确的解决方案：

图 3.2. 为 Four queens puzzle 的正确解决方案



满足所有限制，因此解决方案正确。

请注意，大多数 *n* queens puzzles 有多个正确的解决方案。我们将专注于查找特定 *n* 的正确解决方案，而不是查找特定 *n* 的可能的正确解决方案数量。

问题大小

```
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.
```

n queens 示例的实现没有优化，因为它充当初级示例。然而，它可以轻松地处理 64 queens。随着一些变化，它已被显示，可轻松处理 5000 排队等。

3.7.1. N queens 的域模型

这个示例使用域模型来解决四个问题。

- **创建域模型**
  良好的域模型可以更轻松地理解和解决您的规划问题。

这是 *n* queens 示例的域模型：

```
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
}
```

- **计算搜索空间.**
  Queen 实例有一个 **Column** （例如： 0 是列 A, 1 is column B, ...） 和一个 **Row** （例如，0 为行 0，1 为行 1, ...）。

  可以根据列和行计算升序行和降序排列的 diagonal 行。

  列和行索引从 chessboard 的左上角开始。

```
public class NQueens {

    private int n;
    private List<Column> columnList;
    private List<Row> rowList;

    private List<Queen> queenList;

    private SimpleScore score;

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

- **查找解决方案**
  单个 **NQueens** 实例包含所有 **Queen** 实例的列表。它是通过 Solver 提供并检索到的 **解决方案** 实施。

请注意，在四个 queens 示例中，NQueens **getN （）** 方法始终返回四。

图 3.3. 适用于 Four Queens 的解决方案

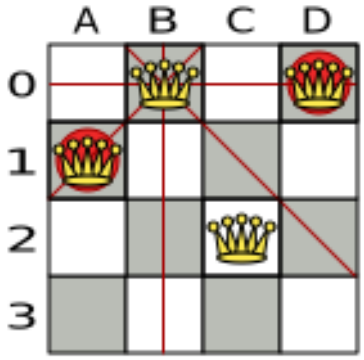


表 3.2. 域模型中解决方案的详情

	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

当两个 queens 共享同一列、row 或 diagonal 行，如 glock 和(**)时，它们可以相互攻击。

3.8. 云平衡

有关此示例的详情，请参考 *红帽构建的 OptaPlanner 快速启动指南*。

3.9. 旅行销售人(TSP - TRAVELING SALESMAN 问题)

给出一个城市列表，找到一个每周访问每个城市的销售人员的最短导览。

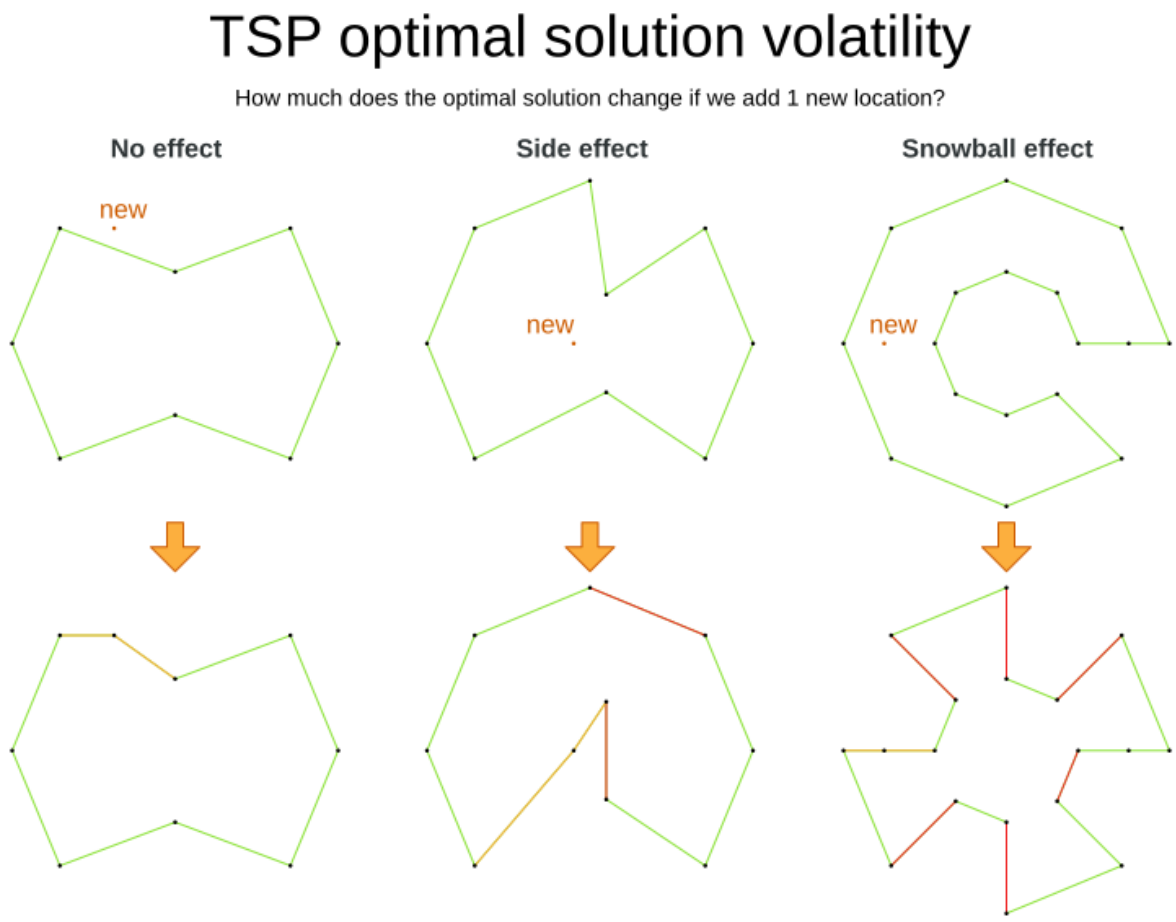
此问题由 Wikipedia 定义。它是计算数 中最密调查的问题之一。然而，在现实世界中，它通常只是规划问题的一部分，以及其他限制，如员工的转变限制。

问题大小

```
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.
```

问题困难

尽管 TSP 的简单定义，但问题似乎难以解决。由于这是一个 NP-hard 问题（如大多数规划问题），当数据问题稍微改变时，特定问题数据集的最佳解决方案可能会改变很多问题：



3.10. TENNIS CLUB 调度

每周，十nis 冲突有四个团队相互循环。公平为团队分配这四个位置。

硬限制：

- 冲突：团队每天只能扮演一次。
- 不可用：有些团队在某些日期上不可用。

中等限制：

- 公平分配：所有团队都应扮演相同次数的（几乎）数。

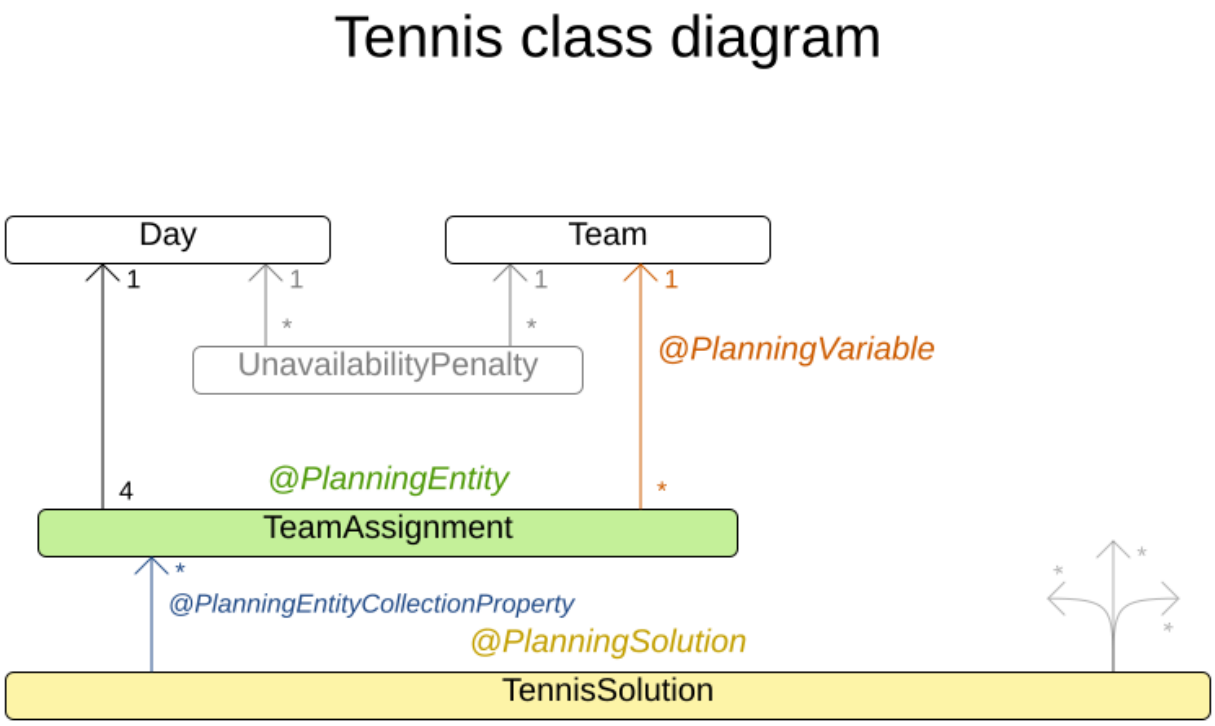
软限制：

- 平均预测：每个团队应针对其他团队相等的次数。

问题大小

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

图 3.4. 域模型



3.11. 满足调度

将每个会议分配到开始时间和房间。会议具有不同的持续时间。

硬限制：

- 房间冲突：两个会议不得同时使用相同的房间。
- 必需的参与：该人员不能同时拥有两个需要满足的会议。
- 需要的房间容量：会议不能处于不适合所有会议的与会者的房间。
- 在同一天开始和结束：会议不应在多天内调度。

中等限制：

- 偏好参加：该人员不能同时拥有两个偏好的会议，也不能同时具有首选和必需的会议。

软限制：

- 更早而不是更新：尽快计划所有会议。
- 会议之间的休息：两者间任何两个会议均应至少有一个时间中断。
- 重叠会议：为了尽量减少并行会议数量，用户不必再一次选择一个会议。

- 首先分配较大的房间：如果应该为该房间分配更大的空间，以便尽可能多地满足相关人员，即使他们尚未注册到该会议。
- 房间稳定性：如果个人连续两个会议，它们之间有两或更少的时间差差，那么它们最好是相同的房间。

问题大小

> 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.

3.12. 课程时间选项卡(ITC 2007 跟踪 3 - CURRICULUM COURSE 计划)

将每个讲座安排到一个 timeslot 和一个房间。

硬限制：

- 教师冲突：教员不能在同一时间段内有两个指导。
- 课程冲突：课程课程不能在同一时间段内有两个指导。
- 空间占用：两个讲座不能在同一时间段内位于同一房间。
- 不可用期限（指定每个数据集）：特定讲义不得分配给特定期间。

软限制：

- 房间容量：房间的容量不应小于其讲座的学位。
- 最小工作日：同一课程的演讲应分为最少的天数。
- 课程紧凑：属于相同课程的演讲应相互相邻（在连续的期间内）。
- 房间稳定性：应将同一课程的介绍分配到相同的房间。

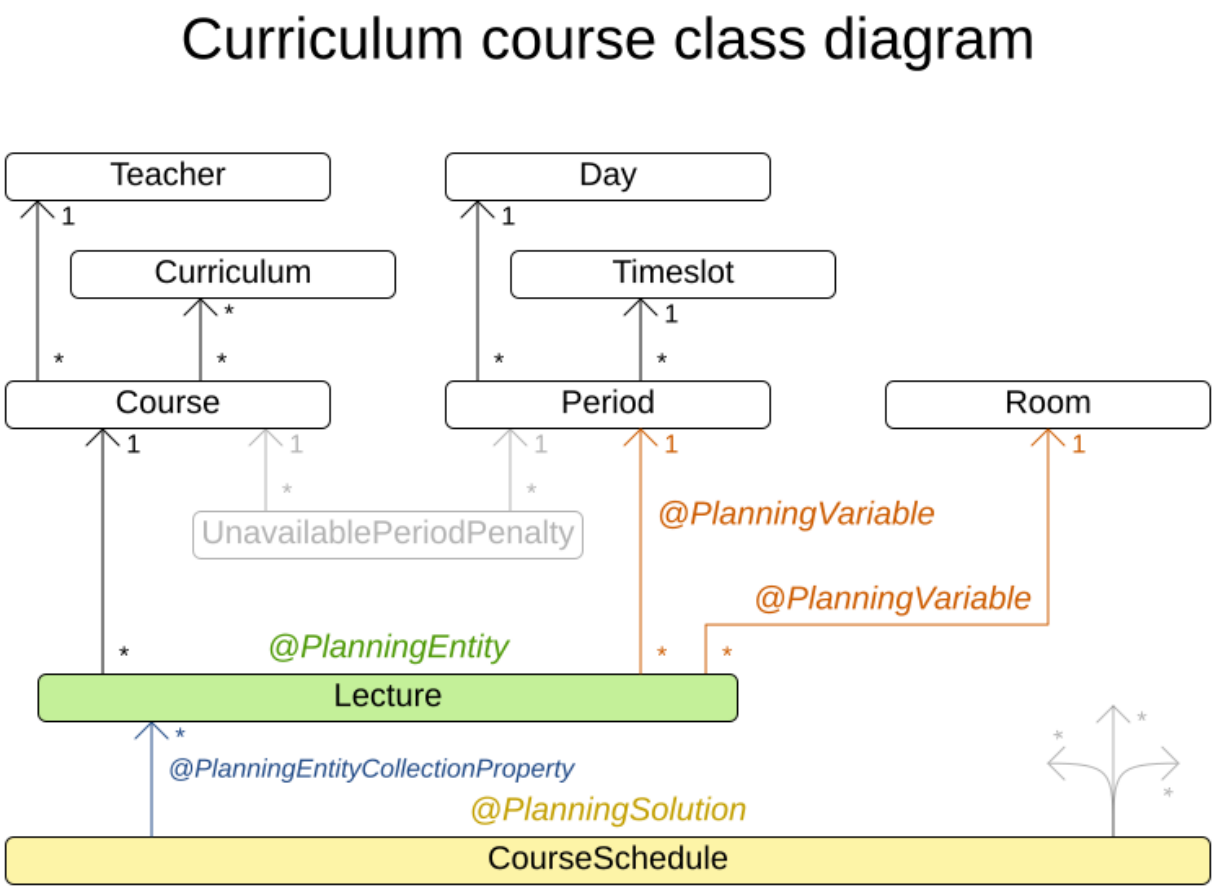
该问题由 International Timetabling Competition 2007 track 3 定义了。

问题大小

> 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.
```

图 3.5. 域模型



3.13. 机器重新分配(GOOGLE ROADEF 2012)

将每个进程分配给机器。所有进程都已有原始（未优化）分配。每个进程都需要一个资源（如 CPU 或 RAM）。这是云负载平衡示例更为复杂的版本。

硬限制：

- 最大容量：不得超过每台机器的每个资源的最大容量。
- 冲突：同一服务的进程必须在不同的机器上运行。
- 分散：同一服务的进程必须在位置上分布。
- 依赖项：服务的进程取决于其他服务的进程必须在其他服务的进程的邻居中运行。
- 临时使用：有些资源是临时的，计算原始机器作为新分配的机器的最大容量。

软限制：

- load ：不应超过每台机器的每个资源的安全容量。
- 平衡：通过平衡每台计算机上的可用资源，为未来分配留出空间。
- 流程迁移成本：进程具有迁移成本。
- 服务迁移成本：服务具有迁移成本。
- 机器移动成本：将流程从机器 A 移动到机器 B 具有另一个特定于 A-B 的移动成本。

这个问题由 Google ROADEF/EURO Challenge 2012 定义。

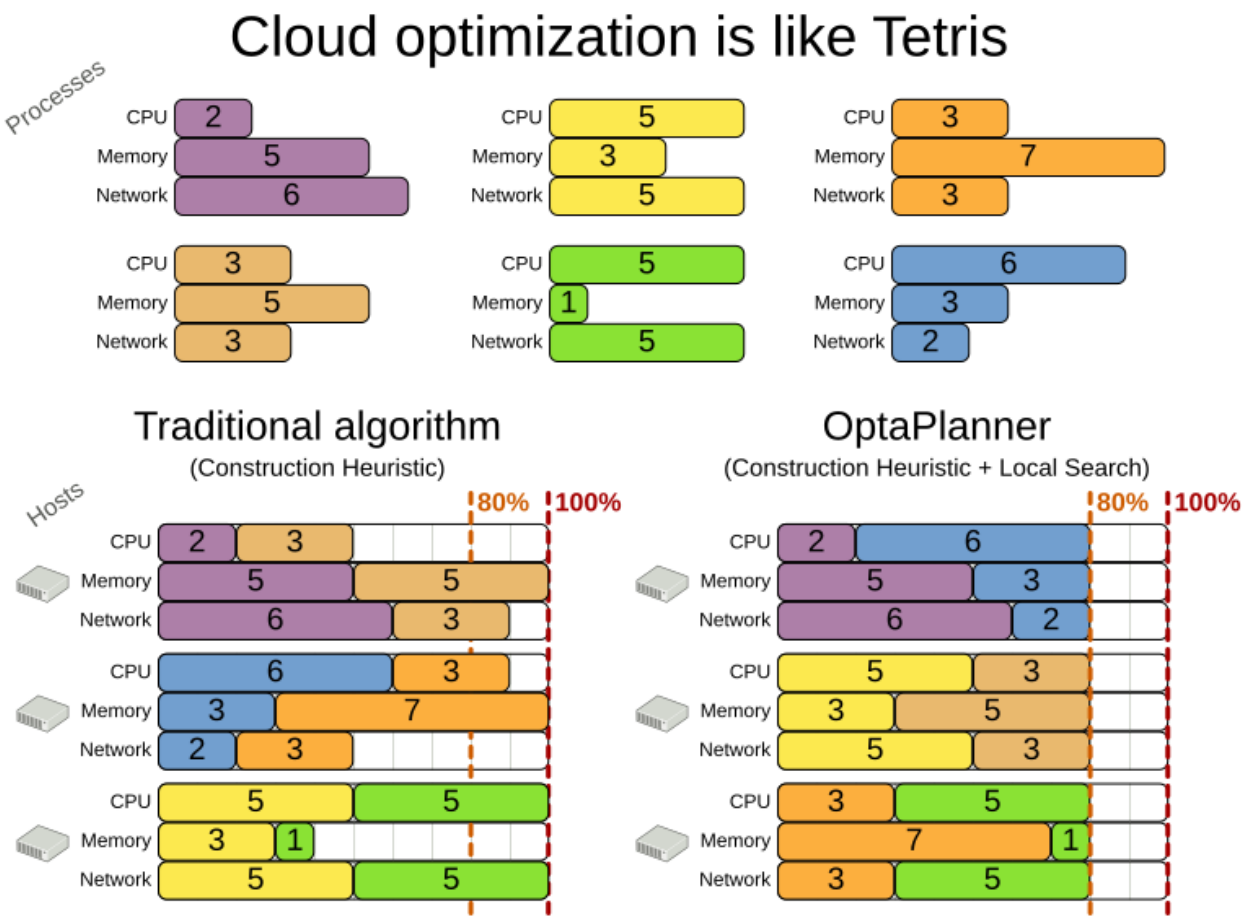
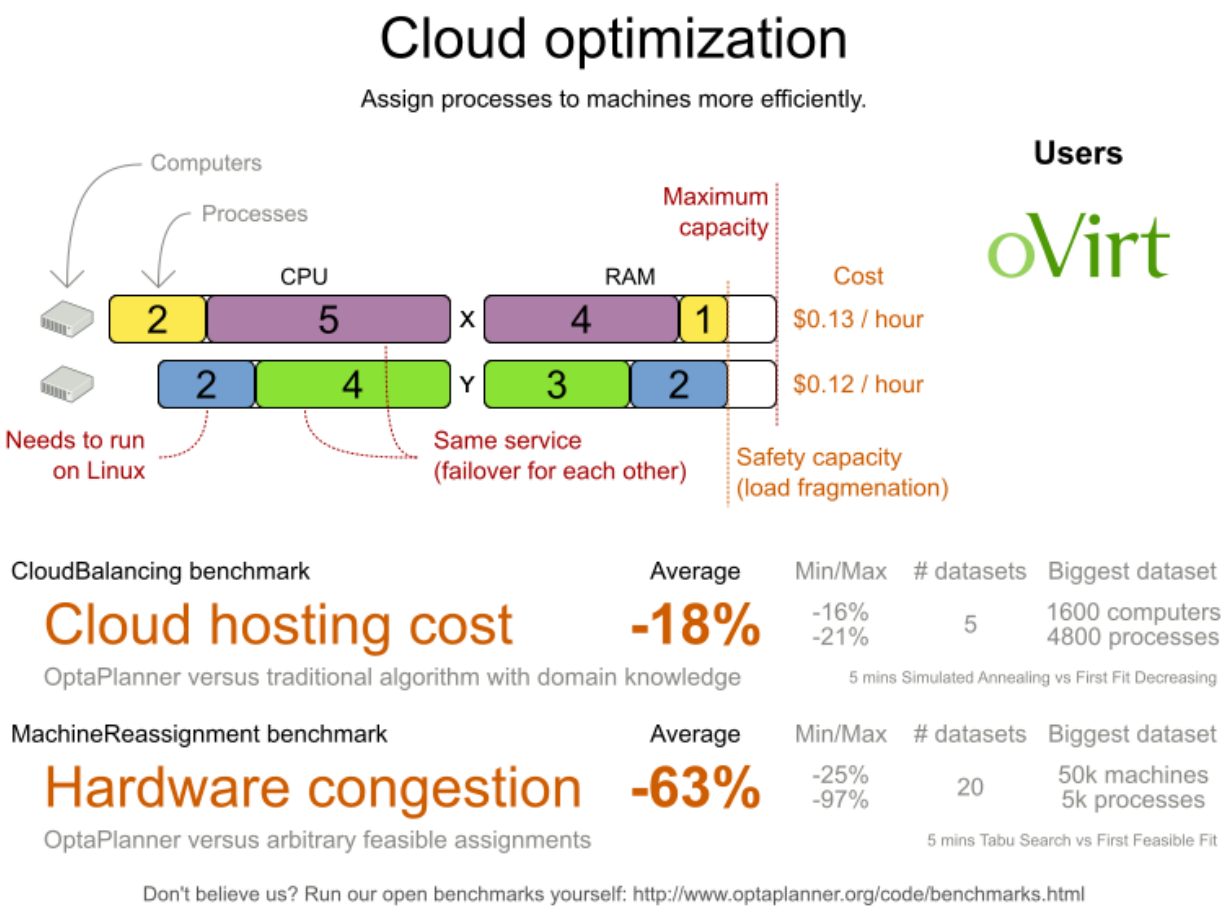


图 3.6. 价值定位

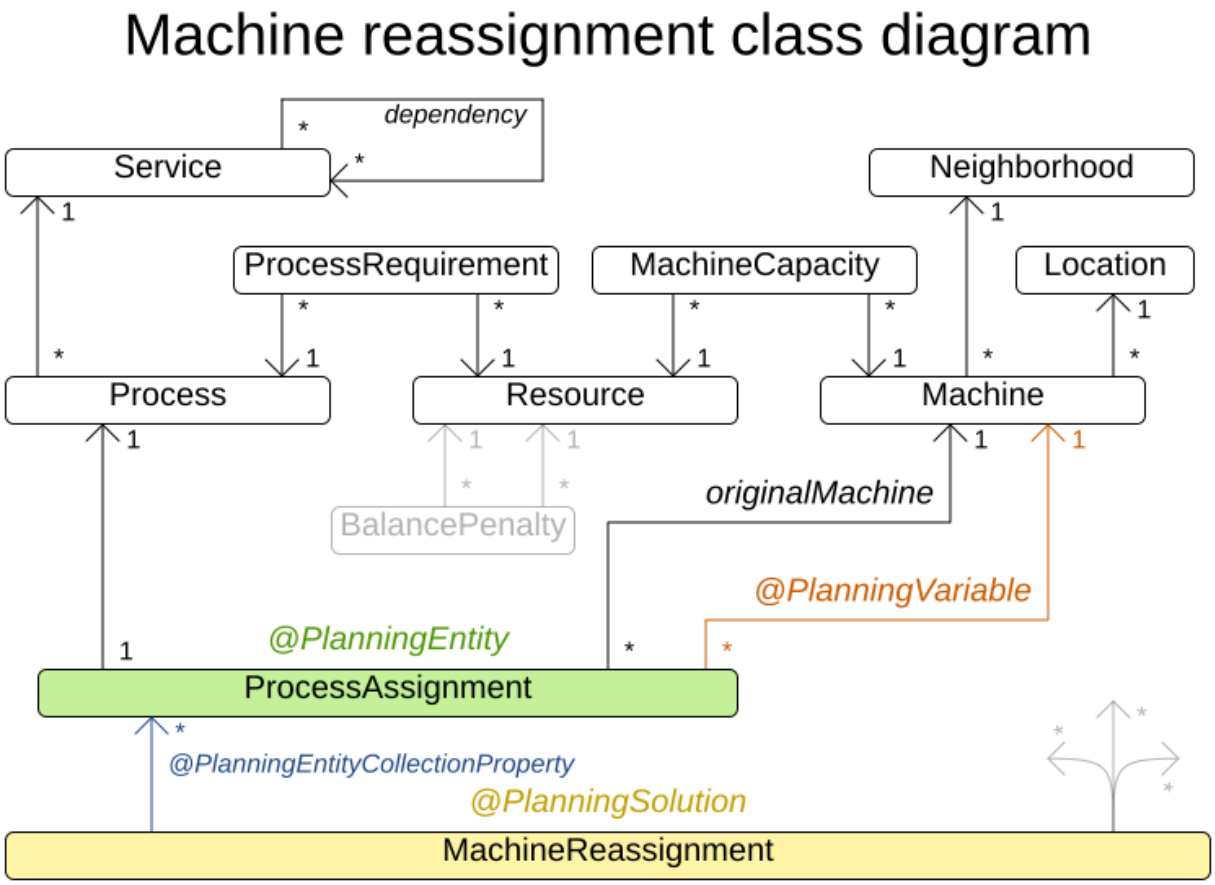


问题大小

```
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.
```

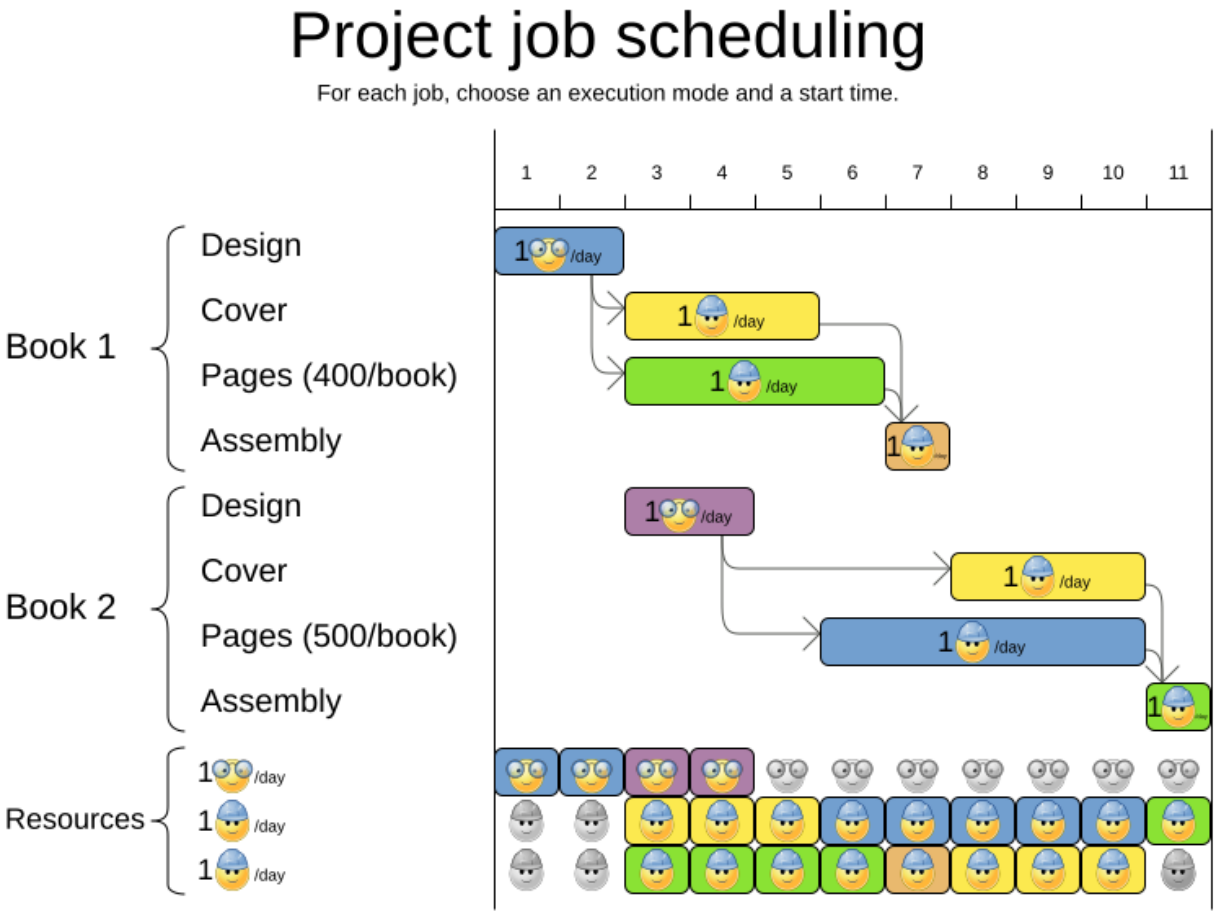
model_b_3 has 6 resources, 5 neighborhoods, 10 locations, 100 machines, 15025 services, 20000 processes and 0 balancePenalties with a search space of 10^40000.
model_b_4 has 6 resources, 5 neighborhoods, 50 locations, 500 machines, 1732 services, 20000 processes and 1 balancePenalties with a search space 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.

图 3.7. 域模型



3.14. 项目作业调度

以时间和执行模式调度所有作业，以最小化项目延迟。每个作业都是项目的一部分。作业可以以不同的方式执行：每种方法都是一个不同的执行模式，它代表了不同的持续时间，但也有不同的资源使用量。这是灵活的 *工作跃点调度形式*。



硬限制：

- 作业优先级：作业只能在所有前一个作业都完成时才启动。
- 资源容量： 除了可用的资源外，请勿使用更多资源。
  - 资源是本地的（在同一项目的作业之间共享）或全局（在所有作业之间共享）
  - 资源可以续订（每天可用）或不可续订（所有天可用）

中等限制：

- 项目总延迟：最小化每个项目的持续时间(makespan)。

软限制：

- 总 makespan：最小化整个多项目调度的持续时间。

问题由 MISTA 2013 年挑战 定义。

问题大小

> 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.

3.15. 任务分配

将每个任务分配到员工队列中的 spot。每个任务都有一个持续时间，受员工的关联性级别和任务的客户影响。

硬限制：

- 技能：每个任务都需要一个或多个技能。员工必须具有所有这些技能。

软级别 0 限制：

- 关键任务：首先完成关键任务，比主要和次任务更早。

软级别 1 限制：

- 最小化 makespan：减少完成所有任务的时间。
  - 首先，从最长的工作员工开始，然后是第二个最长的工作员工等，以创建公平和负载平衡。

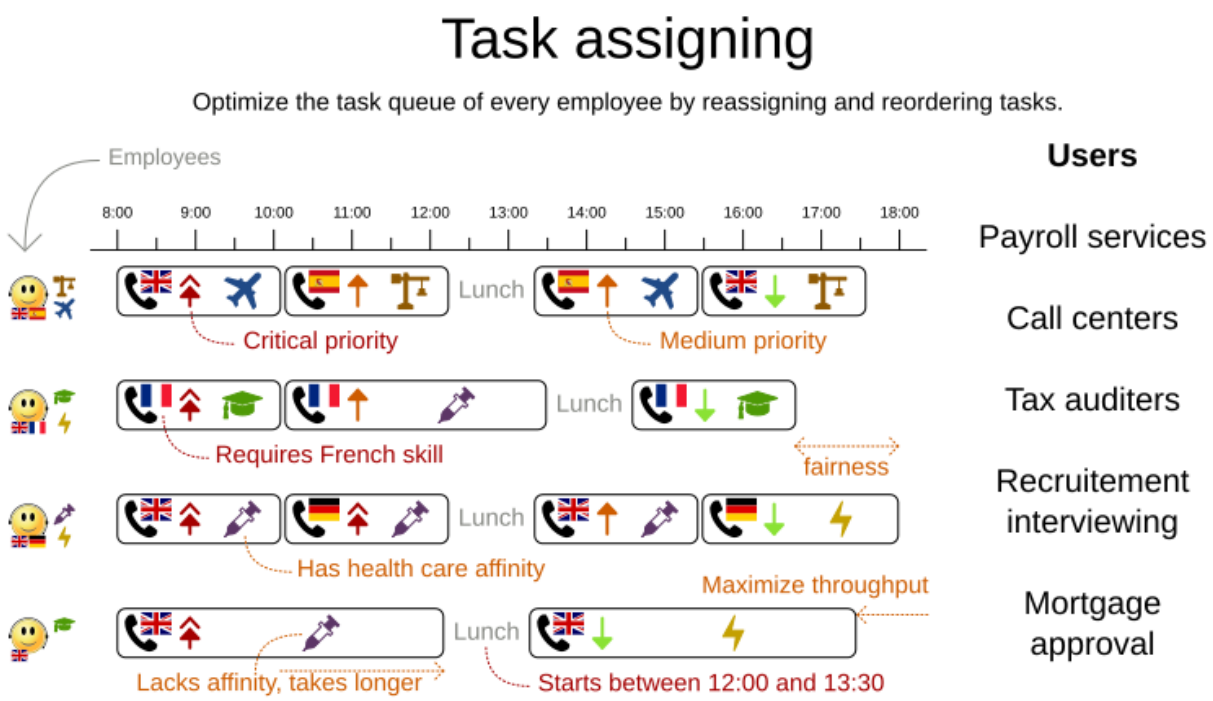
软级别 2 限制：

- 主要的任务：尽快完成主要任务，比次任务更早完成。

软级别 3 限制：

- 次要任务：尽快完成次要任务。

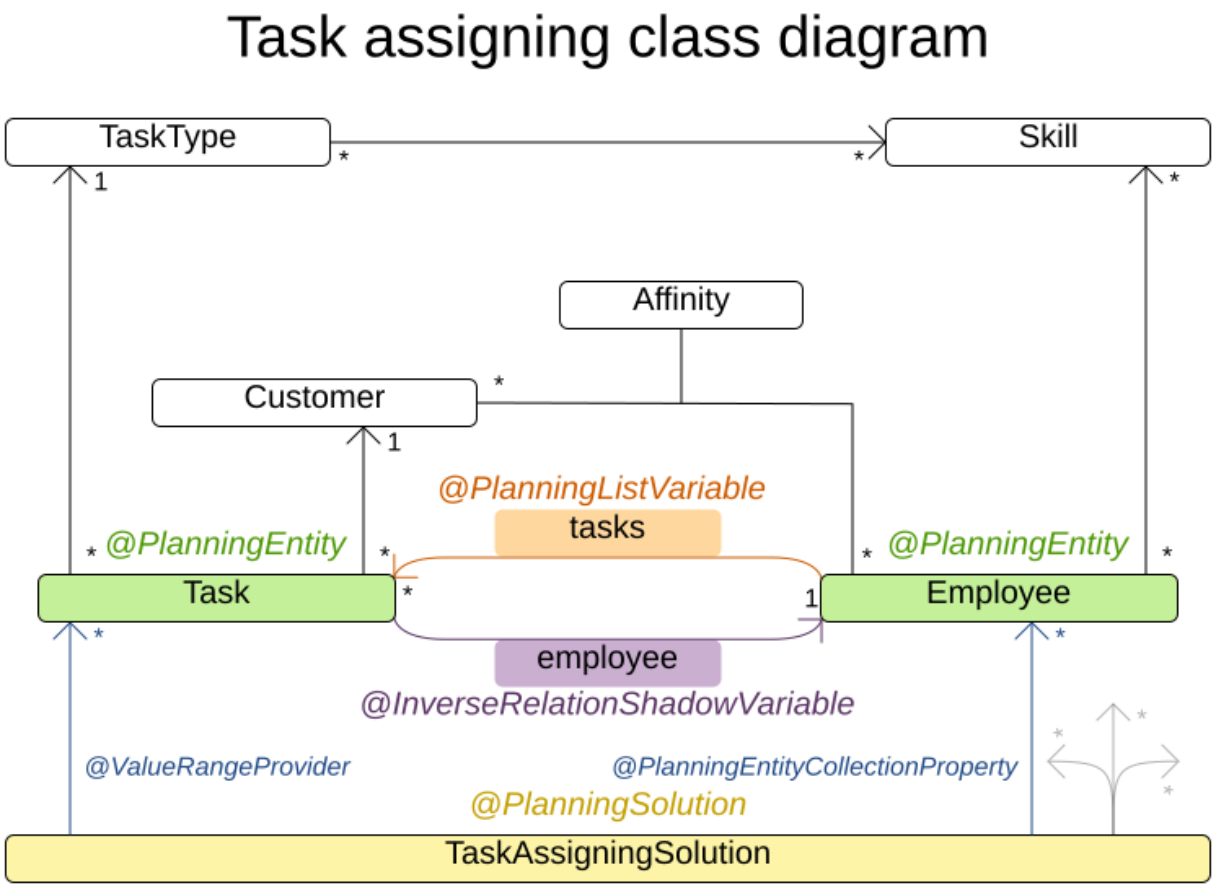
图 3.8. 价值定位



问题大小

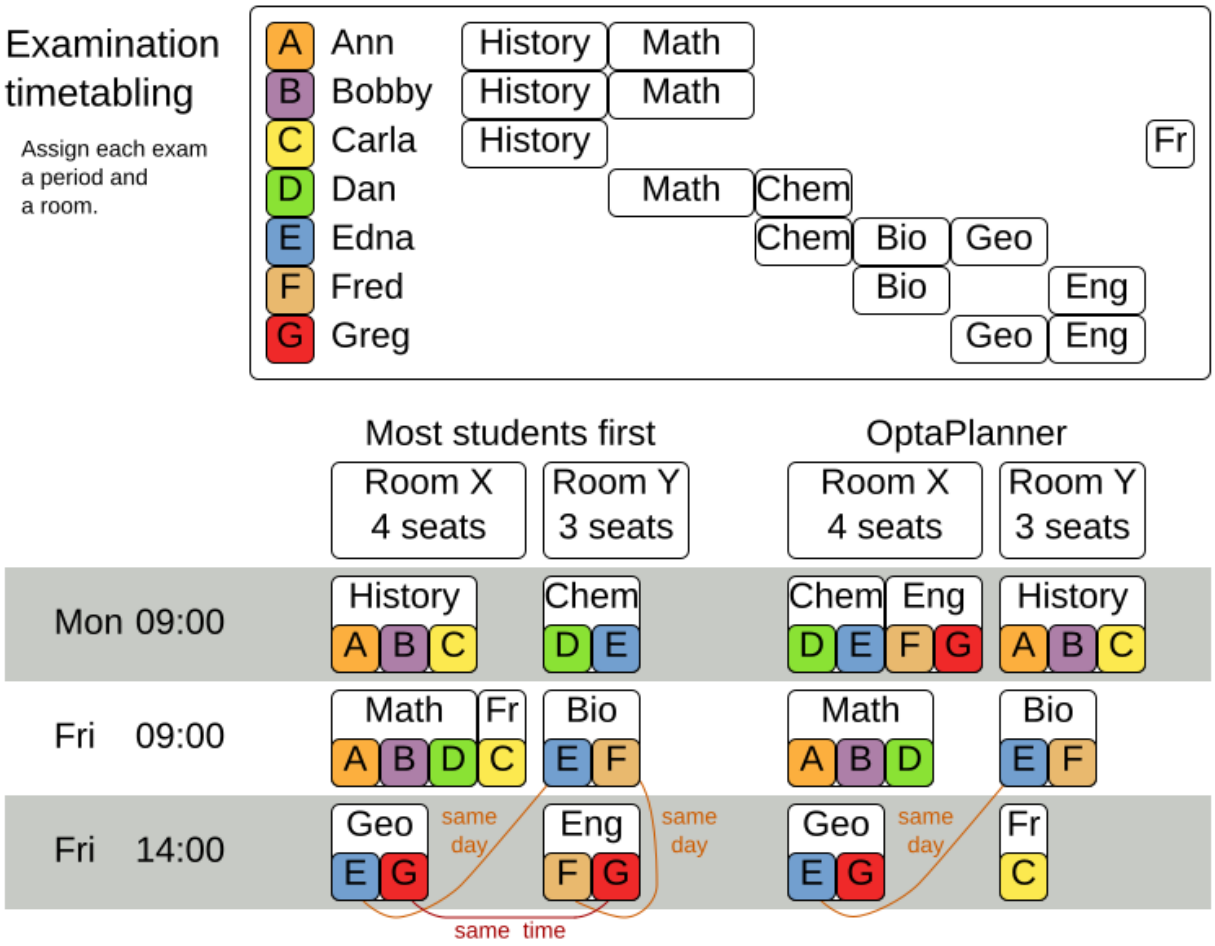
> 24tasks-8employees has 24 tasks, 6 skills, 8 employees, 4 task types and 4 customers with a search space of 10^30.
> 50tasks-5employees has 50 tasks, 5 skills, 5 employees, 10 task types and 10 customers with a search space of 10^69.
> 100tasks-5employees has 100 tasks, 5 skills, 5 employees, 20 task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

图 3.9. 域模型



3.16. 考试时间选项卡(ITC 2007 年跟踪 1 - 考核)

将每个考试安排在一个期间内并进入一个房间。多个考试可以在相同期间内共享相同的房间。



硬限制：

- 考试冲突：两个共享学员的考试项目不得同时发生。
- 房间容量：空间的隔离容量必须始终保持影响。
- 时间段：其所有考试的持续时间必须保持有效。
- 周期相关的硬限制（每个数据集指定）：
  - 协作：两个指定的考试必须使用相同的时间段（但可能还会有其他房间）。
  - 排除：两个指定的考试不得使用相同的周期。
  - 之后：在另一指定考试的期间内必须进行指定的考试。
- 空间相关的硬限制（每个数据集指定）：
  - 独占：一个指定的考试不必与任何其他考试共享空间。

软限制（每个有至关重要的损失）：

- 同一人不应连续两个考试。
- 同一天内不应有两个考试。
- 会议会：两门共享生的考试应成为多个阶段。
- 混合持续时间：两个共享房间考试的考试不应有不同的持续时间。

- 前端负载：在时间表中应提前安排大型考试。
- 周期损失（每个数据集指定）：有些时段在使用时有损失。
- 房间损失（每数据集指定）：一些房间使用时有损失。

它使用大量实际大学的测试数据集。

该问题由 International Timetabling Competition 2007 track 1 定义了。Geoffrey De Smet 在这个竞争中完成了 4 点，并带有非常早版本的 OptaPlanner。从那时起就进行了很多改进。

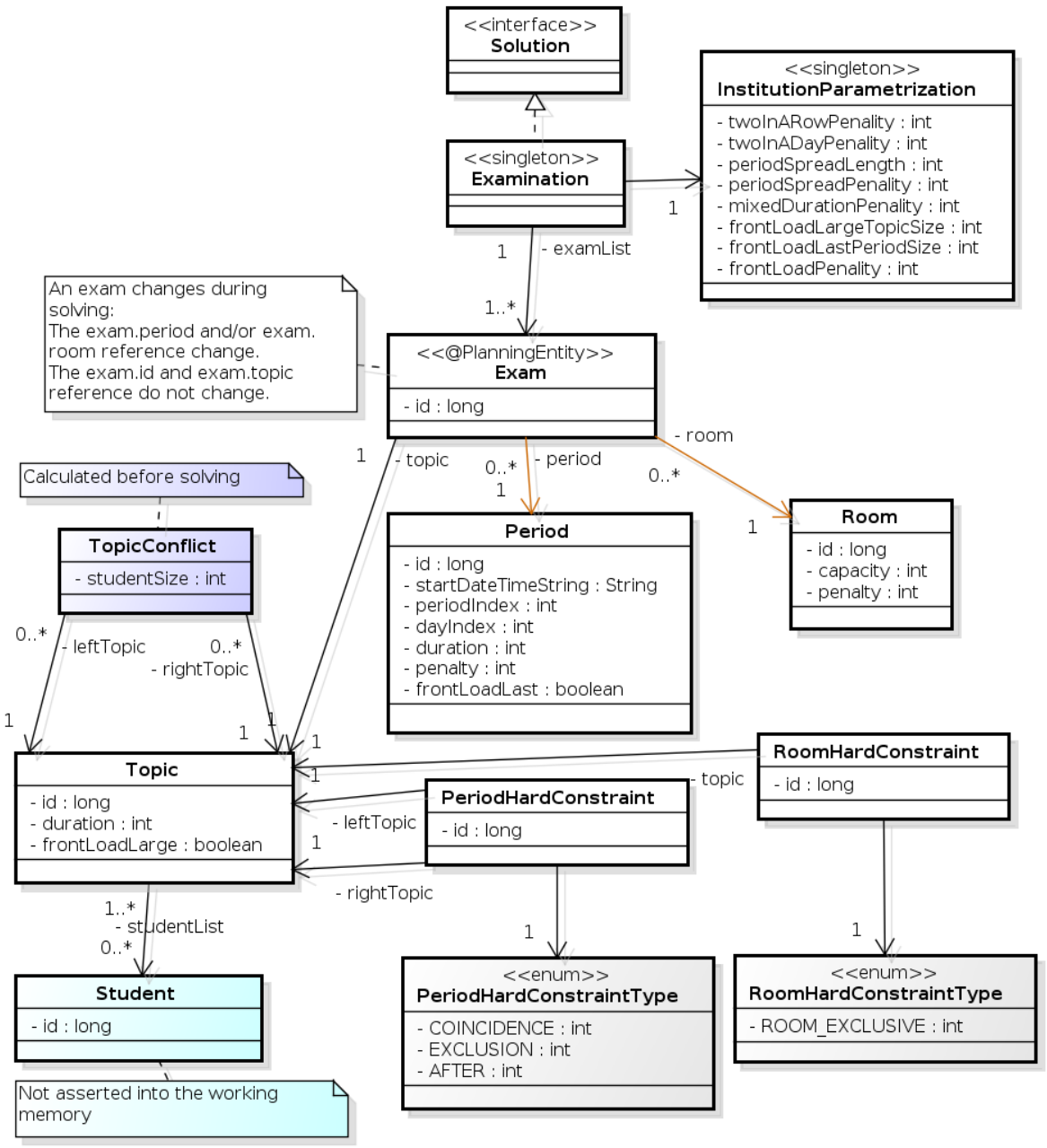
问题大小

```
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.
```

3.16.1. 用于测试时间建立的域模型

下图显示了主要考试域类：

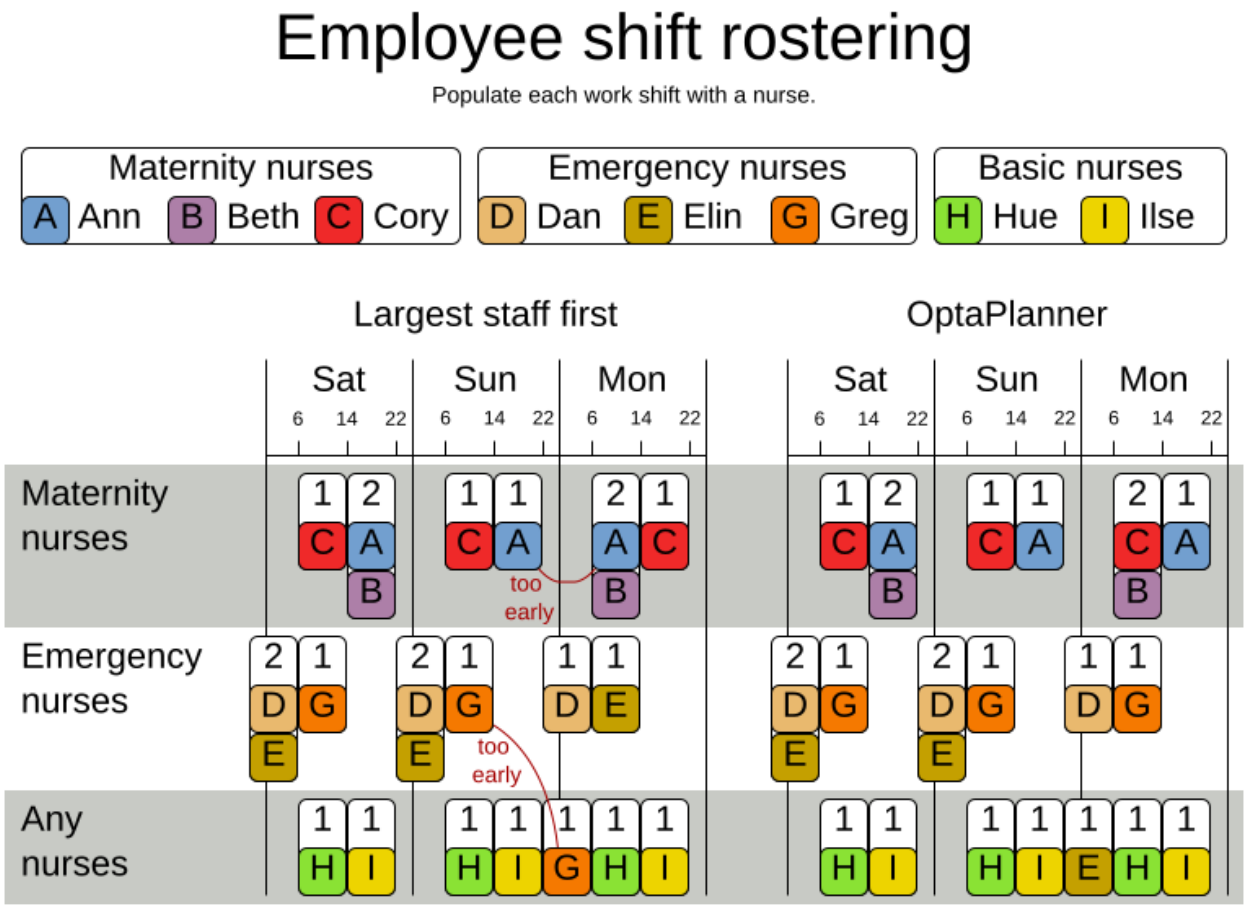
图 3.10. 考核域课程图



请注意，我们已将考试概念分成 **考试** 课程和**主题**课程。在解决时（这是计划实体类）的**考试**实例在它们的期间或房间属性发生变化时发生了变化。**主题**、**Period** 和 **Room** 实例在解决过程中永远不会变化（它们是问题的事实，就像某些其他类一样）。

3.17. NURSE ROSTERING (INRC 2010)

对于每个变化，请分配一个 nurse 来实现这一转变。



硬限制：

- **无未分配的** 转换（内置）：每个转变都需要分配给员工。
- **改变冲突** ：员工每天只能有一个变化。

软限制：

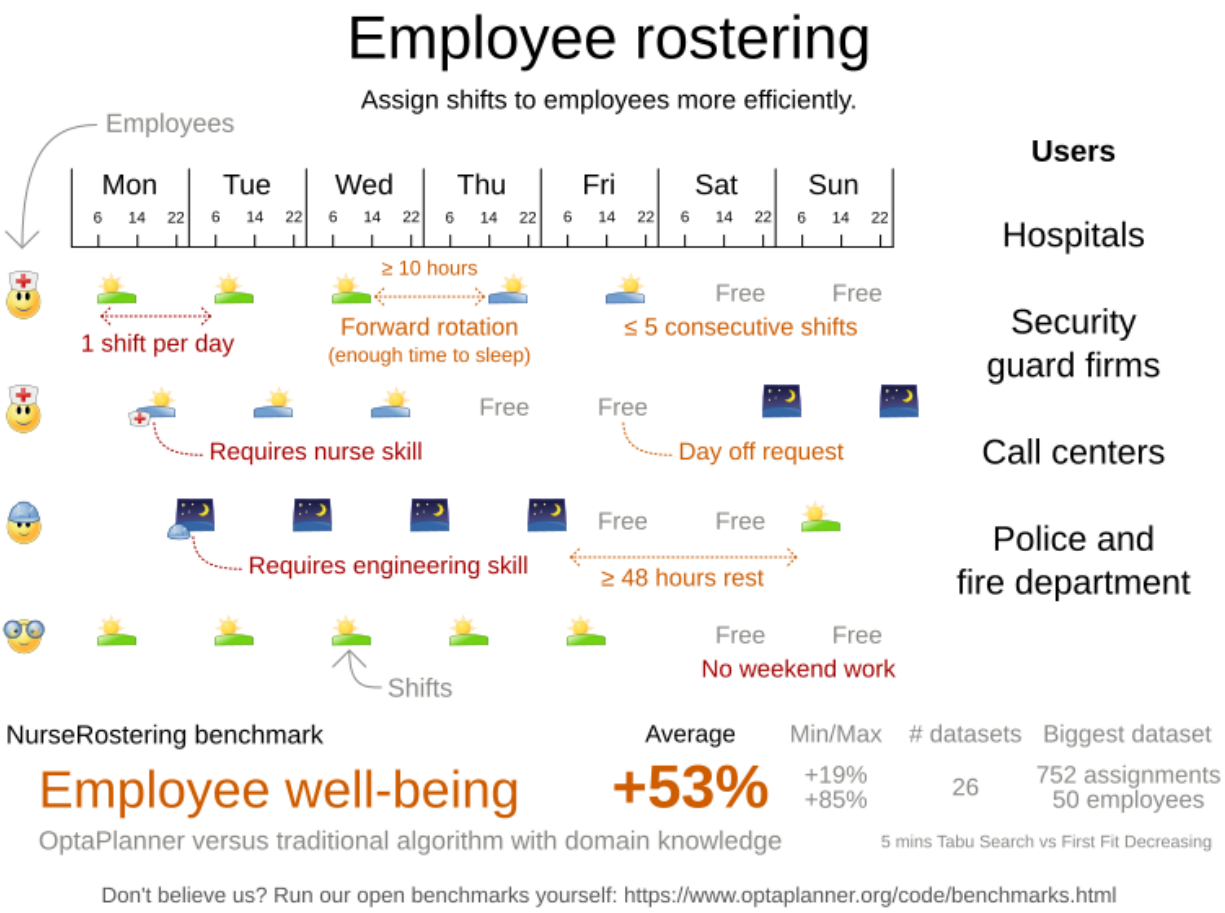
- 合同义务.业务经常违反了这些，因此他们决定将其定义为软限制，而不是硬约束。
  - **最小和最大分配** ：每个员工需要工作超过 x 的转换，少于 y 个转换（取决于其合同）。
  - **最小和最大连续工作天数** ：每个员工需要在一行内 x 到 y 天之间工作（取决于合同）。
  - **最少和最多连续的免费天数** ：每个员工都需要在行中的 x 到 y 天之间自由（取决于合同）。
  - **最少和最多的连续工作周末** ：每个员工需要在一行内 x 和 y 周末（取决于其合同）之间工作。
  - **完整周末** ：每个员工需要每天在周末或根本不工作。
  - **每周端的相同变化类型** ：同一员工的每周末的每周切换都必须相同转换类型。
  - 不需要 **的模式** ：一行中不需要的转换类型的组合，例如，上次转换后接初的转换后再进行移动。
- 员工希望：
  - **申请日** ：员工希望每天工作。

- **第一天请求** ：员工不希望在特定日期工作。
- **改变请求** ：员工希望被分配到特定的转变。
- **转移请求** ：员工不希望分配到特定转换。

- **备选技能** ：分配给技能的员工应具备该转变所需的每个技能的精通。

这个问题由 国际 Nurse Rostering Competition 2010 定义了。

图 3.11. 价值定位



问题大小

有三个数据集类型：

- print：必须以秒为单位解决。
- Medium：必须在几分钟内解决。
- 长：必须以小时为单位解决。

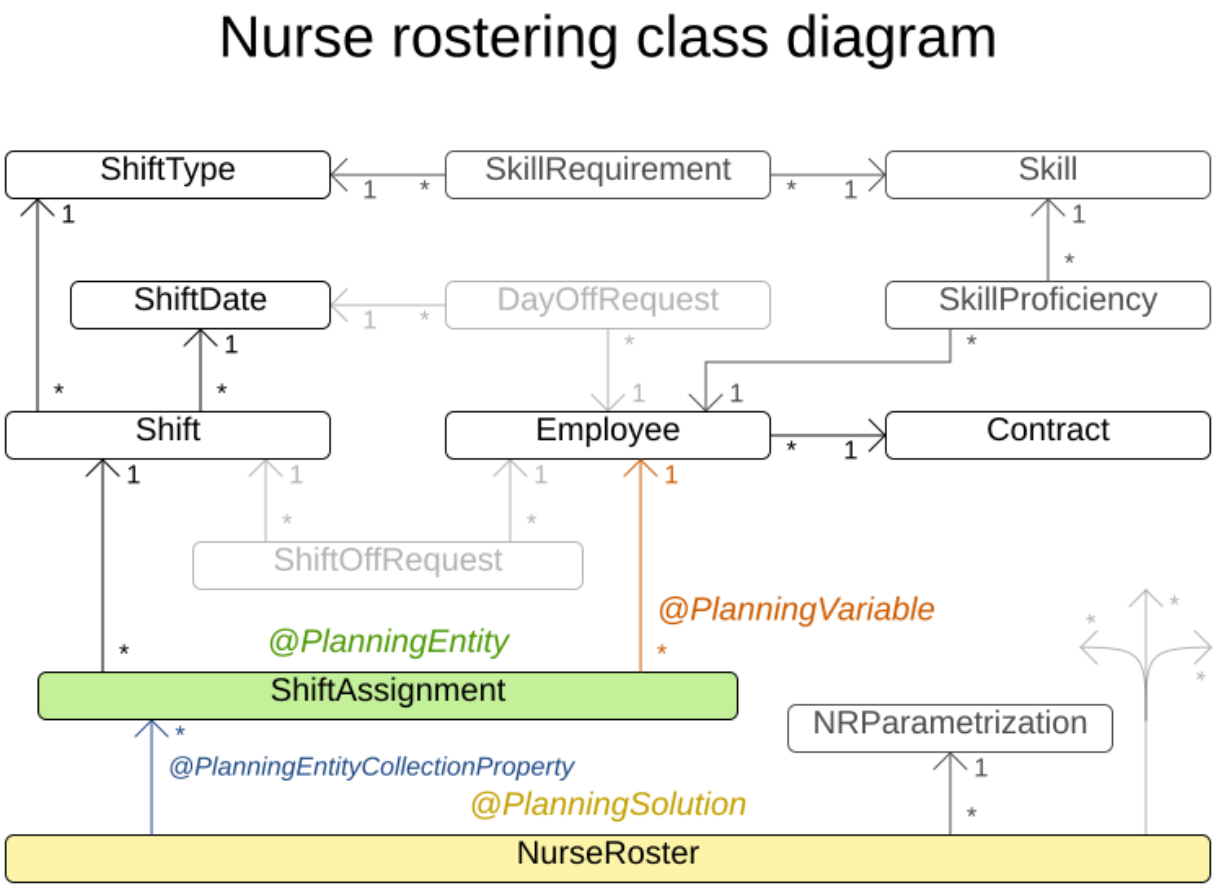
```
toy1         has 1 skills, 3 shiftTypes, 2 patterns, 1 contracts,  6 employees,  7 shiftDates,  35 shiftAssignments and   0 requests with a search space of   10^27.
toy2         has 1 skills, 3 shiftTypes, 3 patterns, 2 contracts, 20 employees, 28 shiftDates, 180 shiftAssignments and 140 requests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

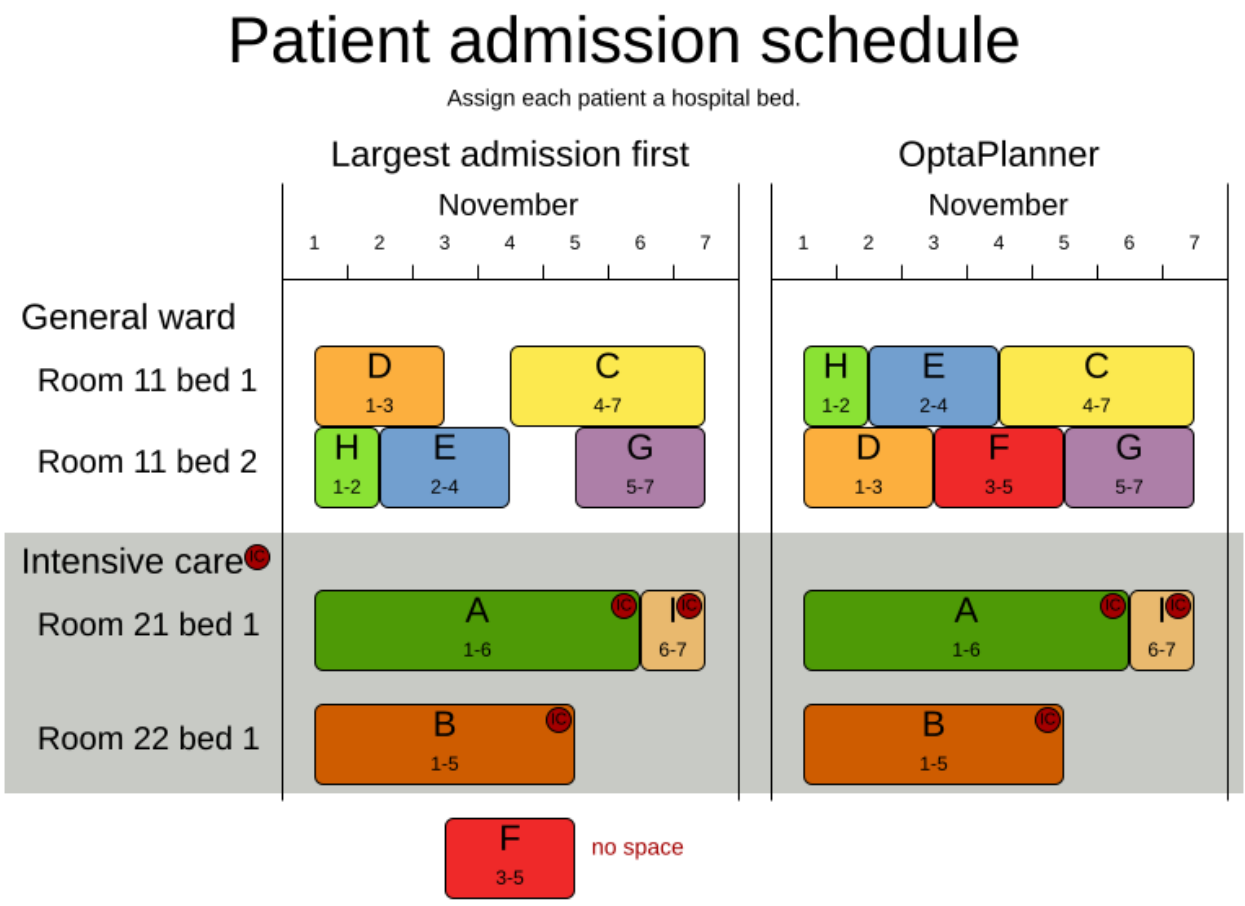
图 3.12. 域模型



3.18. 追踪准入调度

病人护理准入调度(PAS)（也称为 letleting）计划，为每个受监管的病人分配获奖。在病人安排的期间，会分配给病人人为病人的病人。每个机构都属于一个房间，每个房间均属于一个部门。病人人为主办方的辅助日期得到修复。您只需要分配一个 bed。

这个问题比受限的数据集提供。当不需要分配所有计划实体时，最好分配所需数量的实体，而无需破坏硬限制。这称为受限规划。



硬限制：

- 不得将两个病人分配到同一晚上。weight: **-1000hard * conflictNightCount**.
- 房间可能会存在一些限制：只有人意，只是在同一晚上相同的 gender 或 nogender 限制。weight: **-50hard * nightCount**.
- 部门可以具有最短或最长期限。weight: **-100hard * nightCount**.
- 病人可能需要带有特定设备的空间。weight: **-50hard * nightCount**.

中等限制：

- 除非 dataset 受限，否则给每个病人分配有一定的病人。weight: **-1medium * nightCount**.

软限制：

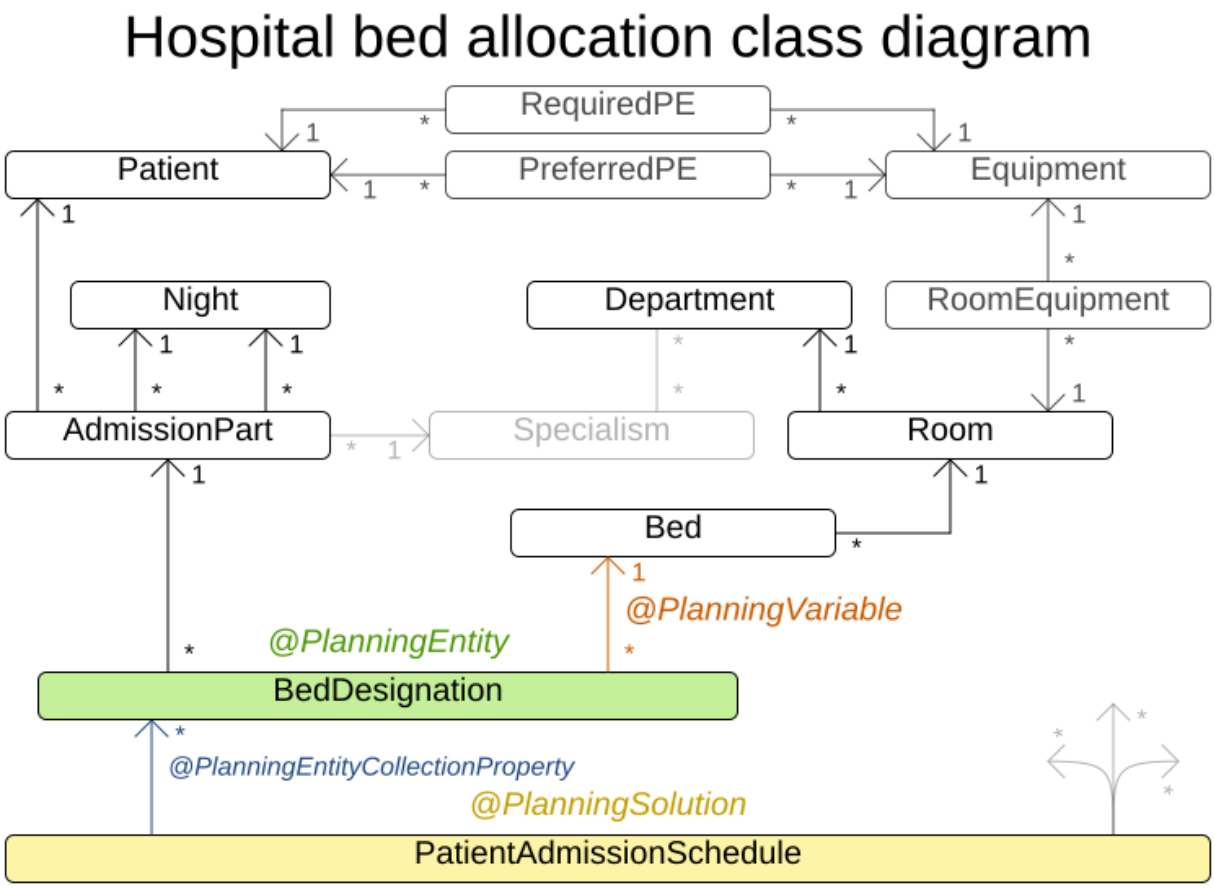
- 病人可以为最大房间大小指定优先权，例如，如果病人想要单个房间。weight: **-8soft * nightCount**.
- 病人最好地分配给一个专门在病人医疗问题中的部门。weight: **-10soft * nightCount**.
- 病人最好地分配给一个在病人类问题中特别存在的房间。weight: **-20soft * nightCount**.
  - 房间特殊性应优先级 1。weight: **-10soft *(priority - 1)* nightCount**.
- 病人可以为具有特定设备的空间指定优先权。weight: **-20soft * nightCount**.

问题是 Kaho 的 Patient Scheduling 中的一个变体，数据集来自真实世界的 Heury。

问题大小

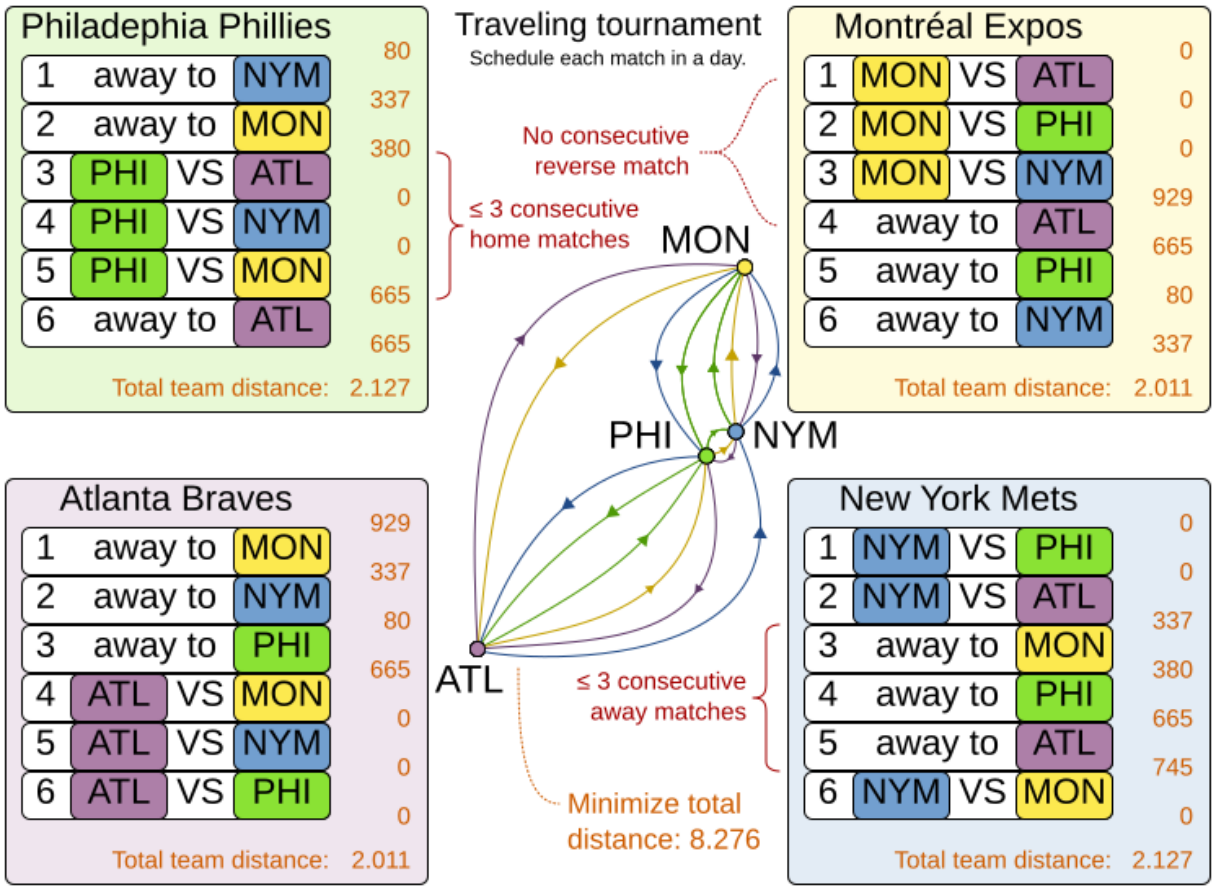
> overconstrained01 has 6 specialisms, 4 equipments, 1 departments, 25 rooms, 69 beds, 14 nights, 519 patients and 519 admissions with a search space of 10^958.
> testdata01 has 4 specialisms, 2 equipments, 4 departments, 98 rooms, 286 beds, 14 nights, 652 patients and 652 admissions with a search space of 10^1603.
> testdata02 has 6 specialisms, 2 equipments, 6 departments, 151 rooms, 465 beds, 14 nights, 755 patients and 755 admissions with a search space of 10^2015.
> testdata03 has 5 specialisms, 2 equipments, 5 departments, 131 rooms, 395 beds, 14 nights, 708 patients and 708 admissions with a search space of 10^1840.
> testdata04 has 6 specialisms, 2 equipments, 6 departments, 155 rooms, 471 beds, 14 nights, 746 patients and 746 admissions with a search space of 10^1995.
> testdata05 has 4 specialisms, 2 equipments, 4 departments, 102 rooms, 325 beds, 14 nights, 587 patients and 587 admissions with a search space of 10^1476.
> testdata06 has 4 specialisms, 2 equipments, 4 departments, 104 rooms, 313 beds, 14 nights, 685 patients and 685 admissions with a search space of 10^1711.
> testdata07 has 6 specialisms, 4 equipments, 6 departments, 162 rooms, 472 beds, 14 nights, 519 patients and 519 admissions with a search space of 10^1389.
> testdata08 has 6 specialisms, 4 equipments, 6 departments, 148 rooms, 441 beds, 21 nights, 895 patients and 895 admissions with a search space of 10^2368.
> testdata09 has 4 specialisms, 4 equipments, 4 departments, 105 rooms, 310 beds, 28 nights, 1400 patients and 1400 admissions with a search space of 10^3490.
> testdata10 has 4 specialisms, 4 equipments, 4 departments, 104 rooms, 308 beds, 56 nights, 1575 patients and 1575 admissions with a search space of 10^3922.
> testdata11 has 4 specialisms, 4 equipments, 4 departments, 107 rooms, 318 beds, 91 nights, 2514 patients and 2514 admissions with a search space of 10^6295.
> testdata12 has 4 specialisms, 4 equipments, 4 departments, 105 rooms, 310 beds, 84 nights, 2750 patients and 2750 admissions with a search space of 10^6856.
> testdata13 has 5 specialisms, 4 equipments, 5 departments, 125 rooms, 368 beds, 28 nights, 907 patients and 1109 admissions with a search space of 10^2847.

图 3.13. 域模型



3.19. TRAVELING TOURNAMENT 问题(TTP)

调度在 *n* 个团队数量之间匹配。



硬限制：

- 每个团队都针对其他团队进行两次：家后，一次。
- 每个团队对每个时间都只有一个匹配项。
- 没有团队必须有超过三个以上连续的家，或连续三个匹配。
- 无重复者：连续两个与团队相同的匹配。

软限制：

- 尽可能减少所有团队传输的总距离。

此问题在 Michael Trick 的网站（还包含全局记录） 上定义。

问题大小

```
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.
```

3.20. 更便宜的时间调度

在时间和机器上调度所有任务以最大程度降低电源成本。电源价格随时间不同。这是 *作业* 跃点调度的一种形式。

硬限制：

- 开始时间限制：每个任务都必须在其最早的开始和最新的开始限制之间启动。
- 最大容量：不得超过每台机器的每个资源的最大容量。
- 启动和关闭：每台机器必须在已分配任务的期间内处于活动状态。在任务间，允许闲置它以避免启动和关闭成本。

中等限制：

- 电源成本：减少整个计划的总电源成本。
  - 机器电源成本：每个活跃或空闲的机器都会消耗电源，这会降低电源成本（取决于该时的电源价格）。
  - 任务电源成本：每个任务都会消耗电源，这会降低电源成本（取决于电源价格）。

- 机器启动和关闭成本：每次机器启动或关闭时，都会产生额外的成本。

软限制（在原始问题定义中添加）：

- 早期启动：首选更早启动任务，而不是稍后开始。

问题由 ICON 挑战 定义。

问题大小

> 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.

3.21. 投资资产类分配(PORTFOLIO OPTIMIZATION)

决定对每个资产类投资的相对数量。

硬限制：

- 风险最大值：标准开发总数不能超过标准 deviation 的最大值。
  - 标准开发计算总数通过应用 Markowitz 组合 Theory 来考虑资产类相关性。
- 地区最大值：每个区域都有最大数量。
- 扇区最大值 ：每个扇区的最大数量。

软限制：

- 最大化预期收益。

问题大小

> 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.

大型数据集尚未创建或测试，但不应造成问题。良好的数据源是 此资产协调网站。

3.22. 会议调度

将每个会议讨论分配到一个 timeslot 和一个房间。Timeslots 可能会重叠。从可以使用 LibreOffice 或 Excel 编辑的文件，读取和写入文件。

硬限制：

- **talk type of timeslot:** 对话的类型必须与 **timeslot** 的 **talk** 类型匹配。

- 空间不可用时间：在通信时，必须有机房间的房间。
- 房间冲突：两个讨论在重叠时slot 无法使用相同的房间。
- speaker 不可用时间：在对话的 timeslot 期间，每个对话的发言人都必须提供。
- speaker 冲突：两个讨论无法在重叠的slot 期间共享发言人。
- 通用目的时间slot 和 room 标签：
  - speaker 需要 timeslot 标签：如果 speaker 有一个所需的 timeslot 标签，那么所有的讨论都必须分配给具有该标签的时间slot。
  - speaker prohibited timeslot 标签 ：如果 speaker 有禁止的 timeslot 标签，则所有他或她的对话都不能分配给具有该标签的 timeslot。
  - talk required timeslot 标签：如果对话有一个所需的 timeslot 标签，则必须将其分配给带有该标签的时间slot。
  - talk prohibited timeslot 标签：如果对话有禁止的 timeslot 标签，则无法将其分配给带有该标签的时间slot。
  - speaker 需要的房间标签：如果发言人具有所需的房间标签，那么所有的对话都必须分配给具有该标签的房间。
  - speaker prohibited room tag：如果发言人有一个禁止的房间标签，则所有的讨论都不能分配给具有该标签的房间。
  - 通信所需的房间标签：如果通信有一个所需的房间标签，则必须将其分配给具有该标签的房间。
  - talk prohibited room tag：如果进行通信有禁止的房间标签，则无法将其分配给具有

该标签的房间。

- talk inter-exclusive-talks tag: Talks that share such a tag, not scheduled in overlapping timeslots.

- 讨论前提条件：必须在所有前提条件对话后安排讨论。

软限制：

- 主题跟踪冲突：最小化在重叠时间slots 期间共享主题标签的对话数量。

- 扇区冲突：最小化在重叠时间slot 期间共享同一扇区标签的对话数量。

- 内容受众级的冲突：对于每个内容标签，请在高级对话前安排简介讨论。

- 受众级多样性：每次slot 时，将与不同受众级别的通信数量最大化。

- 语言多样性：每次slot 时，将与不同语言的沟通数量最大化。

- 通用目的时间slot 和 room 标签：

  - speaker preferred timeslot 标签：如果 speaker 有一个首选 timeslot 标签，则所有的与会者的沟通都应分配给具有该标签的时间slot。

  - speaker undesired timeslot 标签：如果 speaker 有一个不必要的 timeslot 标签，则任何他或她的对话都应分配给具有该标签的时间slot。

  - talk preferred timeslot 标签：如果对话有一个首选 timeslot 标签，则应该将其分配给带有该标签的时间slot。

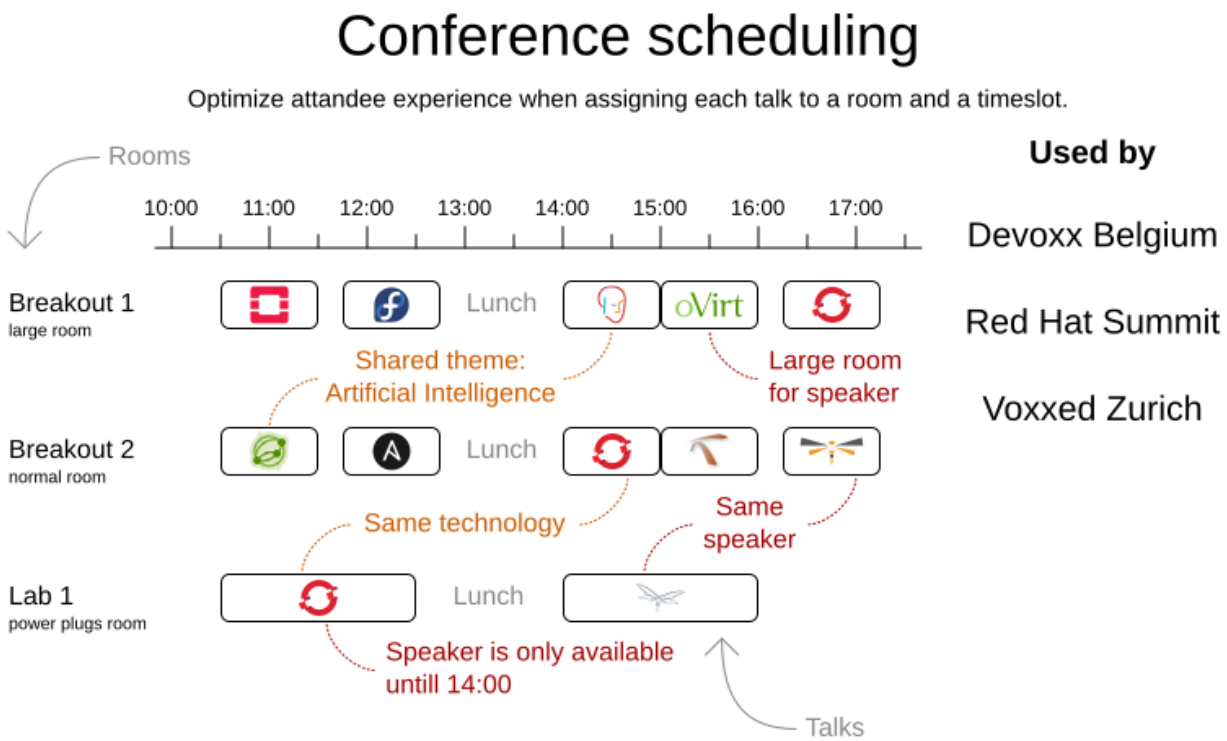
  - talk Undesired timeslot 标签：如果对话有不必要的 timeslot 标签，则不应将其分配

给带有该标签的时间slot。

- speaker preferred room tag：如果 speaker 有一个首选房间标签，则所有的讨论都应分配给具有该标签的房间。
- speaker undesired room 标签 ：如果发言人有一个不需要的房间标签，则任何他或她的讨论都应分配给具有该标签的房间。
- 讨论首选房间标签：如果通信有一个首选房间标签，则应将其分配给具有该标签的空间。
- 通信不必要的房间标签：如果通信有一个不必要的房间标签，则不应将其分配给具有该标签的空间。

- 相同的当天对话：所有共享主题标签或内容标签的对话都应以最少的天数（最好在同一天为单位）进行调度。

图 3.14. 价值定位



问题大小

> 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.

3.23. ROCKUR

从 show to showing rock bank bus from show, but schedule 仅在可用天显示。

硬限制：

- 计**划每个必需显示。**
- 计**划尽可能显示。**

中等限制：

- **最大化收益机会。**
- **最小化最小化时间。**
- **比以后更早地访问。**

软限制：

- 避免长时间驱动时间。

问题大小

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

3.24. 飞行人员的调度

为试用和机票分配机票。

硬限制：

- 所需知识：每个动态分配都有所需的知识。例如，flight AB0001 需要 2 个试验和 3 个动态参与者。
- flight 冲突：每个员工只能同时参加一个 flight
- 在两个 flights 之间传输：在两个机间传输，员工必须能够从 arrival airport 转移到国外机。例如，Amsterdam 到达 Brussels 达到 10:00。
- 员工不可用：员工必须在机班的某一天内可用。例如，An 位于 1-Feb 上的 PTO。

软限制：

- 首次从家分配
- 最后分配在家处的分配

- **每个员工总数的负载均衡 flight 持续时间**

问题大小

> 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.

第 4 章 下载并构建红帽构建的 OPTAPLANNER 示例

您可以下载 Red Hat Build of OptaPlanner 示例，作为红帽客户门户网站上提供的 Red Hat Build of OptaPlanner 源软件包的一部分。

注意

红帽构建的 OptaPlanner 没有 GUI 依赖项。它在服务器或移动 JVM 上同样在桌面上运行。

流程

1. 进入红帽客户门户网站中的 Software Downloads 页面（需要登录），然后从下拉菜单中选择产品和版本：
   - 产品： 红帽构建的 OptaPlanner
   - Version: 8.38
2. 下载 红帽构建的 OptaPlanner 8.38 源分发。
3. 提取 rhbop-8.38.0-optaplanner-sources.zip 文件。

   提取的 org.optaplanner.optaplanner-8.38.0.Final-redhat-00004/optaplanner-examples/src/main/java/org/optaplanner/examples 目录包含示例源代码。
4. 要构建示例，在 org.optaplanner.optaplanner-8.38.0.Final-redhat-00004 目录中，输入以下命令：

   ```
   mvn clean install -Dquickly
   ```
5. 进入 examples 目录：

   ```
   optaplanner-examples
   ```

6. **要运行示例，请输入以下命令：**

```
mvn exec java
```

第 5 章 在 RED HAT BUILD OF QUARKUS 平台上开始使用 RED HAT BUILD OF OPTAPLANNER

红帽构建的 OptaPlanner 与红帽构建的 Quarkus 平台集成。平台工件依赖项的版本（包括 OptaPlanner 依赖项）在 materials (BOM)文件的 Quarkus bill 中维护，该文件 com.redhat.quarkus.platform:quarkus-bom。您不需要指定哪些依赖项版本协同工作。相反，您可以将 Quarkus BOM 文件导入到 pom.xml 配置文件，其中依赖项版本包含在 <dependencyManagement> 部分中。因此，您不需要列出由 pom.xml 文件中指定 BOM 管理的单个 Quarkus 依赖项版本。

其他资源

- 有关使用 Maven 插件在 Quarkus 平台上创建 OptaPlanner 项目的说明，请参阅 第 5.2 节 “使用 Maven 插件在 Quarkus 平台上创建红帽构建的 OptaPlanner 项目”。
- 有关使用 code.quarkus.redhat.com 网站在 Quarkus 平台上生成 OptaPlanner 项目的说明，请参阅 第 5.3 节 “使用 code.quarkus.redhat.com 在 Quarkus 平台上创建红帽构建的 OptaPlanner 项目”。
- 有关使用 CLI 在 Quarkus 平台上生成 OptaPlanner 项目的说明，请参阅 第 5.4 节 “使用 Quarkus CLI 在 Quarkus 平台上创建红帽构建的 OptaPlanner 项目”。

5.1. APACHE MAVEN 和红帽构建的 QUARKUS

Apache Maven 是 Java 应用程序开发中使用的分布式构建自动化工具，用于创建、管理和构建软件项目。Maven 使用名为 Project Object Model(POM)文件的标准配置文件来定义项目并管理构建流程。POM 文件描述了模块和组件依赖项，使用 XML 文件描述生成的项目打包和输出的构建顺序和目标。这可确保以正确、一致的方式构建项目。

Maven 存储库

Maven 存储库存储 Java 库、插件和其他构建构件。默认公共存储库是 Maven 2 Central Repository，但存储库可以是私有和内部的，以在开发团队之间共享通用工件。也可从第三方提供存储库。

您可以将在线 Maven 存储库与 Quarkus 项目一起使用，也可以下载红帽构建的 Quarkus Maven 存储库。

Maven 插件

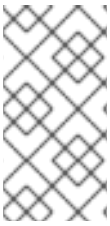
Maven 插件是 POM 文件的定义部分，实现一个或多个目标。Quarkus 应用程序使用以下 Maven 插件：

- Quarkus Maven 插件(quarkus-maven-plugin): 启用 Maven 来创建 Quarkus 项目，支持生成 uber-JAR 文件，并提供开发模式。

- Maven Surefire 插件(maven-surefire-plugin)：在构建生命周期的测试阶段使用，以便在应用程序上执行单元测试。插件生成包含测试报告的文本和 XML 文件。

5.1.1. 为在线存储库配置 Maven settings.xml 文件

您可以通过配置用户 settings.xml 文件，将在线 Maven 存储库与 Maven 项目搭配使用。这是推荐的方法。与共享服务器上的存储库管理器或存储库一起使用的 Maven 设置提供更好的项目控制和易管理性。



注意

当您修改 Maven settings.xml 文件来配置存储库时，这些更改将应用到所有 Maven 项目。

流程

1. 在文本编辑器中打开 Maven ~/.m2/settings.xml 文件或集成开发环境(IDE)。

注意

如果 ~/.m2/ 目录中没有 settings.xml 文件，请将 settings.xml 文件从 $MAVEN_HOME/.m2/conf/ 目录中复制到 ~/.m2/ 目录中。

2. 在 settings.xml 文件的 <profiles > 元素中添加以下行：

```
<!-- Configure the Maven repository -->
<profile>
 <id>red-hat-enterprise-maven-repository</id>
 <repositories>
  <repository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>https://maven.repository.redhat.com/ga/</url>
   <releases>
    <enabled>true</enabled>
   </releases>
```

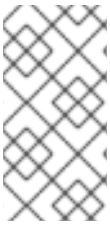
```
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

3. 在 settings.xml 文件的 < activeProfiles > 元素中添加以下行并保存文件。

```
<activeProfile>red-hat-enterprise-maven-repository</activeProfile>
```

5.1.2. 下载并配置 Quarkus Maven 存储库

如果您不想使用在线 Maven 存储库，您可以下载并配置 Quarkus Maven 存储库，以使用 Maven 创建 Quarkus 应用程序。Quarkus Maven 存储库包含 Java 开发人员通常用于构建其应用程序的许多要求。此流程描述了如何编辑 settings.xml 文件来配置 Quarkus Maven 存储库。



注意

当您修改 Maven settings.xml 文件来配置存储库时，这些更改将应用到所有 Maven 项目。

流程

1. 从红帽客户门户的软件下载页面（需要登录） 下载 红帽构建的 Quarkus Maven 存储库 ZIP 文件。

2. 展开下载的存档。

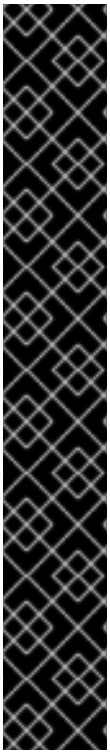
3. 将目录更改为 ~/.m2/ 目录，并在文本编辑器中打开 Maven settings.xml 文件或集成开发环境(IDE)。

4. **将以下行添加到 settings.xml 文件的 & lt;profiles > 元素中，其中 *QUARKUS_MAVEN_REPOSITORY* 是您下载的 Quarkus Maven 存储库的路径。*QUARKUS_MAVEN_REPOSITORY* 的格式必须是 file://$PATH，例如 file:///home/userX/rh-quarkus-2.13.8.GA-maven-repository/maven-repository。**

```
<!-- Configure the Quarkus Maven repository -->
<profile>
  <id>red-hat-quarkus-maven-repository</id>
  <repositories>
    <repository>
      <id>red-hat-quarkus-maven-repository</id>
      <url>QUARKUS_MAVEN_REPOSITORY</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>red-hat-quarkus-maven-repository</id>
      <url>QUARKUS_MAVEN_REPOSITORY</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

5. **在 settings.xml 文件的 < activeProfiles > 元素中添加以下行并保存文件。**

```
<activeProfile>red-hat-quarkus-maven-repository</activeProfile>
```



重要

如果您的 Maven 存储库包含过时的工件，您可能会在构建或部署项目时遇到以下 Maven 错误消息之一，其中 *ARTIFACT_NAME* 是缺少工件，*PROJECT_NAME* 是您要构建的项目的名称：

- 缺少工件 *PROJECT_NAME*
- [ERROR] Failed to execute goal on project *ARTIFACT_NAME*; Could not resolve dependencies for *PROJECT_NAME*

要解决这个问题，请删除 ~/.m2/repository 目录中的本地存储库的缓存版本，以强制下载最新的 Maven 工件。

5.2. 使用 MAVEN 插件在 QUARKUS 平台上创建红帽构建的 OPTAPLANNER 项目

您可以使用 Apache Maven 和 Quarkus Maven 插件获取并运行红帽构建 OptaPlanner 和 Quarkus 应用程序。

先决条件

- 已安装了 OpenJDK 11 或更高版本。红帽构建的 Open JDK 可通过红帽客户门户网站中的 Software Downloads 页面（需要登录）。
- Apache Maven 3.8 或更高版本已安装。Maven 位于 Apache Maven Project 网站。

流程

1. 在命令终端中，输入以下命令验证 Maven 是否使用 JDK 11，并且 Maven 版本是否为 3.8 或更高版本：

```
mvn --version
```

2. 如果前面的命令没有返回 JDK 11，请将 JDK 11 的路径添加到 PATH 环境变量中，然后再次输入前面的命令。

3.

要生成 Quarkus OptaPlanner quickstart 项目，请输入以下命令，其中 redhat-0000x 是 Quarkus BOM 文件的当前版本：

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:2.13.8.SP1-redhat-0000x:create \
    -DprojectGroupId=com.example \
    -DprojectArtifactId=optaplanner-quickstart  \
    -DplatformGroupId=com.redhat.quarkus.platform
    -DplatformArtifactId=quarkus-bom
    -DplatformVersion=2.13.8.SP1-redhat-0000x \
    -DnoExamples
    -Dextensions="resteasy,resteasy-jackson,optaplanner-quarkus,optaplanner-quarkus-jackson" \
```

此命令在 ./optaplanner-quickstart 目录中创建以下元素：

- Maven 结构
- src/main/docker中的 Dockerfile 文件示例
- 应用程序配置文件

表 5.1. mvn io.quarkus:quarkus-maven-plugin:2.13.8.SP1-redhat-0000x:create 命令中使用的属性

属性	描述
projectGroupId	项目的组 ID。
projectArtifactId	项目的工件 ID。
extensions	用于此项目的 Quarkus 扩展的逗号分隔列表。如需 Quarkus 扩展的完整列表，请在命令行中输入 **mvn quarkus:list-extensions**。
noExamples	创建一个项目结构，但没有测试或类。

projectGroupID 和 projectArtifactID 属性的值用于生成项目版本。默认项目版本为 1.0.0-SNAPSHOT。

4. 要查看您的 OptaPlanner 项目，请将目录改为 OptaPlanner Quickstarts 目录：

```
cd optaplanner-quickstart
```

5. 查看 **pom.xml** 文件。内容应类似以下示例：

```
<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>optaplanner-quickstart</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <properties>
    <compiler-plugin.version>3.8.1</compiler-plugin.version>
    <maven.compiler.release>11</maven.compiler.release>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
    <quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
    <quarkus.platform.version>2.13.8.SP1-redhat-0000x</quarkus.platform.version>
    <skipITs>true</skipITs>
    <surefire-plugin.version>3.0.0-M7</surefire-plugin.version>
  </properties>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>${quarkus.platform.group-id}</groupId>
        <artifactId>${quarkus.platform.artifact-id}</artifactId>
        <version>${quarkus.platform.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
      <dependency>
        <groupId>${quarkus.platform.group-id}</groupId>
        <artifactId>quarkus-optaplanner-bom</artifactId>
        <version>${quarkus.platform.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.optaplanner</groupId>
      <artifactId>optaplanner-quarkus</artifactId>
    </dependency>
    <dependency>
      <groupId>org.optaplanner</groupId>
      <artifactId>optaplanner-quarkus-jackson</artifactId>
    </dependency>
    <dependency>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-resteasy-jackson</artifactId>
```

```
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-arc</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-junit5</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
<repositories>
  <repository>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <id>redhat</id>
    <url>https://maven.repository.redhat.com/ga</url>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <id>redhat</id>
    <url>https://maven.repository.redhat.com/ga</url>
  </pluginRepository>
</pluginRepositories>
<build>
  <plugins>
    <plugin>
      <groupId>${quarkus.platform.group-id}</groupId>
      <artifactId>quarkus-maven-plugin</artifactId>
      <version>${quarkus.platform.version}</version>
      <extensions>true</extensions>
      <executions>
        <execution>
          <goals>
            <goal>build</goal>
            <goal>generate-code</goal>
```

```
            <goal>generate-code-tests</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${compiler-plugin.version}</version>
      <configuration>
        <compilerArgs>
          <arg>-parameters</arg>
        </compilerArgs>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <configuration>
        <systemPropertyVariables>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
          <maven.home>${maven.home}</maven.home>
        </systemPropertyVariables>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-failsafe-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>integration-test</goal>
            <goal>verify</goal>
          </goals>
          <configuration>
            <systemPropertyVariables>
              <native.image.path>${project.build.directory}/${project.build.finalName}-
runner</native.image.path>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
              <maven.home>${maven.home}</maven.home>
            </systemPropertyVariables>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
 </build>
 <profiles>
  <profile>
    <id>native</id>
    <activation>
      <property>
        <name>native</name>
      </property>
    </activation>
```

```
    <properties>
      <skipITs>false</skipITs>
      <quarkus.package.type>native</quarkus.package.type>
    </properties>
  </profile>
 </profiles>
</project>
```

5.3. 使用 CODE.QUARKUS.REDHAT.COM 在 QUARKUS 平台上创建红帽构建的 OPTAPLANNER 项目

您可以使用 code.quarkus.redhat.com 网站生成红帽构建的 OptaPlanner Quarkus Maven 项目，并自动添加并配置要在应用程序中使用的扩展。

本节介绍了生成 OptaPlanner Maven 项目并包括以下主题：

- 指定应用程序的基本详情。
- 选择您要包含在项目中的扩展。
- 使用您的项目文件生成可下载存档。
- 使用自定义命令编译和启动应用程序。

先决条件

- 您有一个 Web 浏览器。

流程

1. 在您的浏览器中打开 https://code.quarkus.redhat.com：
2. 指定项目详情：
3. 输入项目的组名称。名称的格式遵循 Java 软件包命名约定，如 com.example。

4. 输入您要用于项目中生成的 Maven 工件的名称，如 code-with-quarkus。

5. 选择 Build Tool > Maven 来指定您要创建一个 Maven 项目。您选择的构建工具决定了项目：

   - 生成的项目的目录结构
   - 您生成的项目中使用的配置文件格式
   - 在生成项目后，会显示用于编译和启动 code.quarkus.redhat.com 的自定义构建脚本和命令。

   注意

   红帽支持使用 code.quarkus.redhat.com 创建 OptaPlanner Maven 项目。红帽不支持生成 Gradle 项目。

6. 输入要在项目生成的工件中使用的版本。此字段的默认值为 1.0.0-SNAPSHOT。建议使用 语义版本，但如果您愿意，您可以使用不同类型的版本。

7. 输入构建工具在打包项目时生成的工件的软件包名称。

   根据 Java 软件包命名约定，软件包名称应与用于项目的组名称匹配，但您可以指定不同的名称。

8. 选择以下扩展作为依赖项包括：

   - RESTEasy JAX-RS (quarkus-resteasy)
   - RESTEasy Jackson (quarkus-resteasy-jackson)

- OptaPlanner AI 约束 solver (optaplanner-quarkus)
- OptaPlanner Jackson (optaplanner-quarkus-jackson)

红帽为列表中的独立扩展提供了不同的支持级别，这些扩展由每个扩展名称旁的标签表示：

  - 红帽完全支持 *SUPPORTED* 扩展，可用于生产环境中的企业应用程序。
  - 在 技术预览功能支持范围 下，红帽可能会受限地支持红帽产品。
  - *红帽不支持在生产环境中使用 DEV-SUPPORT 扩展，但 Red Hat 开发人员在开发新应用程序时支持它们提供的核心功能。*
  - *DEPRECATED 扩展计划被替换为提供相同功能的较新的技术或实施。*

    红帽不支持在生产环境中使用未标记扩展。

9. *选择 Generate your application 来确认您的选择并显示包含您生成的项目的存档的下载链接。覆盖屏幕还显示可用于编译和启动应用程序的自定义命令。*
10. *选择 Download the ZIP 将带有生成的项目文件的存档保存到您的系统中。*
11. *提取存档的内容。*
12. *进入包含您提取的项目文件的目录：*

```
cd <directory_name>
```

13. *以开发模式编译并启动应用程序：*

```
./mvnw compile quarkus:dev
```

5.4. 使用 QUARKUS CLI 在 QUARKUS 平台上创建红帽构建的 OPTAPLANNER 项目

您可以使用 Quarkus 命令行界面(CLI)创建 Quarkus OptaPlanner 项目。

先决条件

- 已安装 Quarkus CLI。如需更多信息，请参阅使用 Quarkus 命令行界面构建 Quarkus 应用程序。

流程

1. 创建一个 Quarkus 应用程序：

```
quarkus create app -P io.quarkus:quarkus-bom:2.13.8.SP1-redhat-0000x
```

2. 要查看可用的扩展，请输入以下命令：

```
quarkus ext -i
```

这个命令返回以下扩展：

```
optaplanner-quarkus
optaplanner-quarkus-benchmark
optaplanner-quarkus-jackson
optaplanner-quarkus-jsonb
```

3. 输入以下命令在项目的 pom.xml 文件中添加扩展：

```
quarkus ext add resteasy-jackson
quarkus ext add optaplanner-quarkus
quarkus ext add optaplanner-quarkus-jackson
```

4. 在文本编辑器中打开 pom.xml 文件。文件的内容应类似以下示例：

```
<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.acme</groupId>
```

```
 <artifactId>code-with-quarkus-optaplanner</artifactId>
 <version>1.0.0-SNAPSHOT</version>
 <properties>
<compiler-plugin.version>3.8.1</compiler-plugin.version>
<maven.compiler.parameters>true</maven.compiler.parameters>
<maven.compiler.source>11</maven.compiler.source>
<maven.compiler.target>11</maven.compiler.target>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
<quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
<quarkus.platform.group-id>io.quarkus</quarkus.platform.group-id>
<quarkus.platform.version>2.13.8.SP1-redhat-0000x</quarkus.platform.version>
<surefire-plugin.version>3.0.0-M5</surefire-plugin.version>
 </properties>
 <dependencyManagement>
<dependencies>
  <dependency>
    <groupId>${quarkus.platform.group-id}</groupId>
    <artifactId>${quarkus.platform.artifact-id}</artifactId>
    <version>${quarkus.platform.version}</version>
    <type>pom</type>
    <scope>import</scope>
  </dependency>
  <dependency>
    <groupId>io.quarkus.platform</groupId>
    <artifactId>optaplanner-quarkus</artifactId>
    <version>2.2.2.Final</version>
    <type>pom</type>
    <scope>import</scope>
  </dependency>
</dependencies>
 </dependencyManagement>
 <dependencies>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-arc</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-resteasy</artifactId>
</dependency>
<dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-quarkus</artifactId>
</dependency>
<dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-quarkus-jackson</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-resteasy-jackson</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-junit5</artifactId>
```

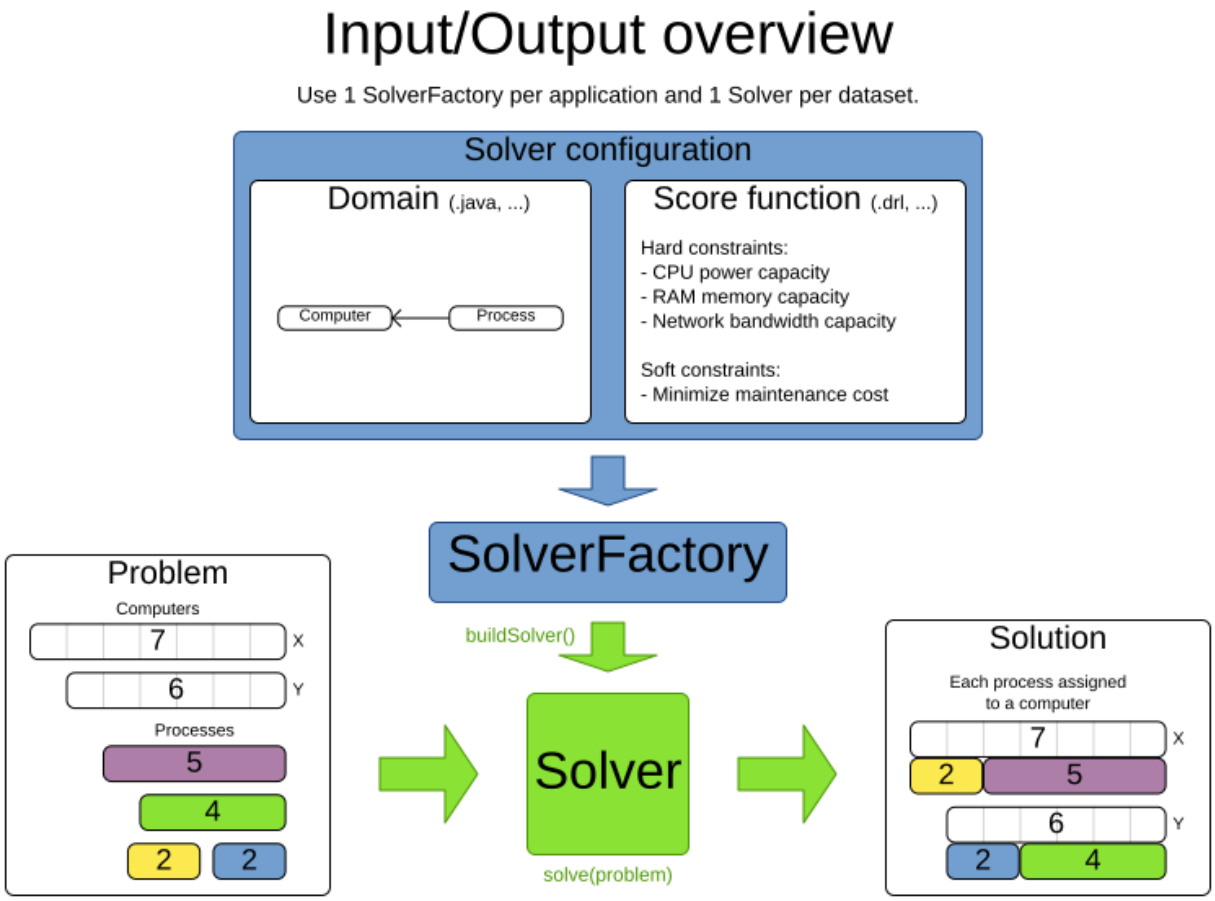
```
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
  <scope>test</scope>
</dependency>
 </dependencies>
 <build>
<plugins>
  <plugin>
    <groupId>${quarkus.platform.group-id}</groupId>
    <artifactId>quarkus-maven-plugin</artifactId>
    <version>${quarkus.platform.version}</version>
    <extensions>true</extensions>
    <executions>
      <execution>
        <goals>
          <goal>build</goal>
          <goal>generate-code</goal>
          <goal>generate-code-tests</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>${compiler-plugin.version}</version>
    <configuration>
      <parameters>${maven.compiler.parameters}</parameters>
    </configuration>
  </plugin>
  <plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>${surefire-plugin.version}</version>
    <configuration>
      <systemPropertyVariables>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
        <maven.home>${maven.home}</maven.home>
      </systemPropertyVariables>
    </configuration>
  </plugin>
</plugins>
 </build>
 <profiles>
<profile>
  <id>native</id>
  <activation>
    <property>
      <name>native</name>
    </property>
  </activation>
  <build>
    <plugins>
      <plugin>
```

```
        <artifactId>maven-failsafe-plugin</artifactId>
        <version>${surefire-plugin.version}</version>
        <executions>
          <execution>
            <goals>
              <goal>integration-test</goal>
              <goal>verify</goal>
            </goals>
            <configuration>
              <systemPropertyVariables>
                <native.image.path>${project.build.directory}/${project.build.finalName}-
run</native.image.path>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
                <maven.home>${maven.home}</maven.home>
              </systemPropertyVariables>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <properties>
    <quarkus.package.type>native</quarkus.package.type>
  </properties>
 </profile>
  </profiles>
</project>
```

部分 III. 红帽构建的 OPTAPLANNER SOLVER

使用 OptaPlanner 解决计划问题包括以下步骤：

1. 将您的计划问题建模 为带有 @PlanningSolution 注释的类（例如，NQueens 类）。
2. 配置 Solver （例如，任何 NQueens 实例的第一个 Fit 和 Tabu Search solver）。
3. 从 数据层加载问题 数据集（如 Four Queens 实例）。这是规划问题。
4. 通过 Solver .solve (problem)解决问题，这将返回最佳解决方案。



第 6 章 配置红帽构建的 OPTAPLANNER SOLVER

您可以使用以下方法配置 OptaPlanner solver：

- 使用 XML 文件。
- 使用 SolverConfig API。
- 在域模型中添加类注解和 JavaBean 属性注解。
- 控制 OptaPlanner 用于访问您的域的方法。
- 定义自定义属性。

6.1. 使用 XML 文件配置 OPTAPLANNER SOLVER

每个示例项目都有一个可编辑的解析器配置文件。& lt;EXAMPLE>SolverConfig.xml 文件位于 org.optaplanner.optaplanner-8.38.0.Final-redhat-00004/optaplanner-examples/src/main/resources/org/optaplanner/examples/<EXAMPLE> 目录中，其中 <EXAMPLE > 是 OptaPlanner 示例项目的名称。或者，您可以使用 SolverFactory .createFromXmlFile （）从文件创建 SolverFactory。但是，出于可移植性的原因，建议使用 classpath 资源。

Solver 和 SolverFactory 均有一个名为 Solution_ 的通用类型，这是代表计划问题和解决方案的类。

通过更改配置，OptaPlanner 使切换优化算法相对容易。

流程

1. 使用 Solver Factory 构建 Solver 实例。
2. 配置 solver 配置 XML 文件：
    a. 定义模型。

b. *定义 score 功能。*

c. *可选：配置优化算法。*

以下示例是 NQueens 问题的 solver XML 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <!-- Define the model -->

<solutionClass>org.optaplanner.examples.nqueens.domain.NQueens</solutionClass
>
  <entityClass>org.optaplanner.examples.nqueens.domain.Queen</entityClass>

  <!-- Define the score function -->
  <scoreDirectorFactory>

<scoreDrl>org/optaplanner/examples/nqueens/optional/nQueensConstraints.drl</sc
oreDrl>
  </scoreDirectorFactory>

  <!-- Configure the optimization algorithms (optional) -->
  <termination>
    ...
  </termination>
  <constructionHeuristic>
    ...
  </constructionHeuristic>
  <localSearch>
    ...
  </localSearch>
</solver>
```

注意

在某些环境中，如 OSGi 和 JBoss 模块，您的 JAR 文件中的 solver 配置、分数 DRLs 和域类等 classpath 资源可能无法提供给 optaplanner-core JAR 文件的默认 ClassLoader。在这些情况下，将您的类的 ClassLoader 作为参数提供：

```
SolverFactory<NQueens> solverFactory =
SolverFactory.createFromXmlResource(
        ".../nqueensSolverConfig.xml",
getClass().getClassLoader());
```

3. 使用 solver 配置 XML 文件来配置 SolverFactory，该文件作为 class Loader.getResource () 定义的类路径资源提供：

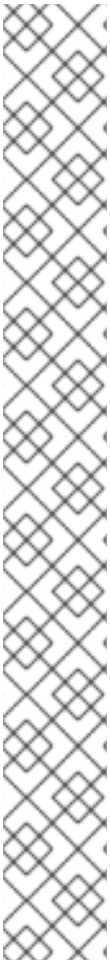
```
SolverFasctory<NQueens> solverFactory =
SolverFactory.createFromXmlResource(
        "org/optaplanner/examples/nqueens/optional/nqueensSolverConfig.xml");
Solver<NQueens> solver = solverFactory.buildSolver();
```

6.2. 使用 JAVA API 配置 OPTAPLANNER SOLVER

您可以使用 SolverConfig API 配置 solver。这在运行时动态更改值特别有用。以下示例在 NQueens 项目中构建 Solver 之前更改基于系统属性的运行时间：

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource(
        "org/optaplanner/examples/nqueens/optional/nqueensSolverConfig.xml");
solverConfig.withTerminationConfig(new TerminationConfig()
        .withMinutesSpentLimit(userInput));

SolverFactory<NQueens> solverFactory = SolverFactory.create(solverConfig);
Solver<NQueens> solver = solverFactory.buildSolver();
```

solver 配置 XML 文件中的每个元素都作为 Config 类或软件包命名空间 org.optaplanner.core.config 中的 Config 类的属性提供。这些配置 类是 XML 格式的 Java 表示。它们构建软件包命名空间的 org.optaplanner.core.impl 并将其编译成一个高效的 Solver。



注意

要为每个用户请求动态配置 SolverFactory，请在初始化过程中构建模板 SolverConfig，并使用每个用户请求的复制构造器复制它。以下示例演示了如何进行此操作，并带有 NQueens 问题：

```
private SolverConfig template;

public void init() {
    template = SolverConfig.createFromXmlResource(
"org/optaplanner/examples/nqueens/optional/nqueensSolverConfig.xml");
    template.setTerminationConfig(new TerminationConfig());
}

// Called concurrently from different threads
public void userRequest(..., long userInput) {
    SolverConfig solverConfig = new SolverConfig(template); // Copy it
    solverConfig.getTerminationConfig().setMinutesSpentLimit(userInput);
    SolverFactory<NQueens> solverFactory =
SolverFactory.create(solverConfig);
    Solver<NQueens> solver = solverFactory.buildSolver();
    ...
}
```

6.3. OPTAPLANNER 注解

您必须指定域模型中的哪些类是计划实体，哪些属性是规划变量等。使用以下方法之一在 OptaPlanner 项目中添加注解：

- 在域模型中添加类注解和 JavaBean 属性注解。属性注解必须在 getter 方法上，而不是在 setter 方法上。注解的 getter 方法不需要是公共的。这是推荐的方法。
- 在域模型中添加类注解和字段注解。注解的字段不需要是公共的。

6.4. 指定 OPTAPLANNER 域访问

默认情况下，OptaPlanner 使用反映访问您的域。与直接访问相比，反映可靠，但速度较慢。或者，您可以将 OptaPlanner 配置为使用 Gizmo 访问您的域，这将生成字节码，直接访问域的字段和方法，而无需反映。但是，此方法有以下限制：

- planning 注解只能位于公共字段和公共 getters 上。

- *io.quarkus.gizmo:gizmo 必须位于 classpath 上。*

注意

当您将 OptaPlanner 与 Quarkus 搭配使用时，这些限制不适用，因为 Gizmo 是默认的域访问类型。

流程

要使用 Quarkus 之外的 Gizmo，请在 solver 配置中设置 domainAccessType ：

```
<solver>
  <domainAccessType>GIZMO</domainAccessType>
</solver>
```

6.5. 配置自定义属性

在 OptaPlanner 项目中，您可以添加自定义属性以解决实例化类并明确提及自定义属性的文档。

先决条件

- *您有一个解决者。*

流程

1. *添加自定义属性。*

   例如，如果您的 EasyScoreCalculator 有大量缓存的计算，并且您希望在一个基准中增加缓存大小，请添加 myCacheSize 属性：

   ```
   <scoreDirectorFactory>
     <easyScoreCalculatorClass>...MyEasyScoreCalculator</easyScoreCalculatorClass>
     <easyScoreCalculatorCustomProperties>
       <property name="myCacheSize" value="1000"/><!-- Override value -->
     </easyScoreCalculatorCustomProperties>
   </scoreDirectorFactory>
   ```

2. *为每个自定义属性添加公共 setter，该属性在构建 Solver 时调用。*

```
public class MyEasyScoreCalculator extends EasyScoreCalculator<MySolution,
SimpleScore> {

    private int myCacheSize = 500; // Default value

    @SuppressWarnings("unused")
    public void setMyCacheSize(int myCacheSize) {
        this.myCacheSize = myCacheSize;
    }

  ...
}
```

大多数值类型都支持，包括 布尔值,int,double,BigDecimal,String 和 enums。

第 7 章 使用 OPTAPLANNER SOLVER

解决者为您规划问题找到最佳和最佳解决方案。一个解决者一次只能解决一个规划问题实例。solvers 使用 Solver Factory 方法 构建：

```
public interface Solver<Solution_> {

    Solution_ solve(Solution_ problem);

    ...
}
```

一个 solver 应该只从单个线程访问，除了 javadoc 中特别记录为 thread-safe 的方法。solve （） 方法调整当前线程。拖放线程可能会导致 REST 服务的 HTTP 超时，它需要额外的代码来并行解决多个数据集。要避免这些问题，请使用 SolverManager。

7.1. 解决问题

使用解决方案解决计划问题。

先决条件

- 从 solver 配置构建的 Solver
- 代表规划问题实例的 @PlanningSolution 注释

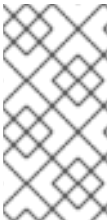
流程

提供计划问题作为 solve （） 方法的参数。解决者将返回找到最佳解决方案。

以下示例解决了 NQueens 问题：

```
NQueens problem = ...;
NQueens bestSolution = solver.solve(problem);
```

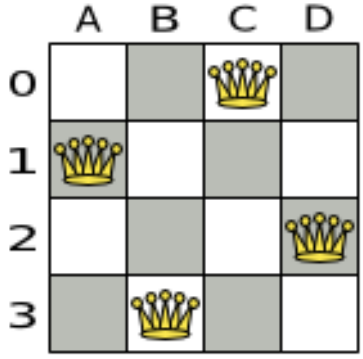
在本例中，solve （） 方法将返回 NQueens 实例，每个 Queen 分配到一个 Row。



注意

提供给 solve (Solution) 方法的解决方案实例可以部分或完全初始化，这通常是重复规划的情况。

图 7.1. 用于 8ms 中 Four Queens Puzzle 的最佳解决方案(Also a Optimal Solution)



solve (Solution) 方法可能需要很长时间，具体取决于问题大小和 solver 配置。Solver 智能地通过可能的解决方案搜索空间，并记住在解决过程中遇到的最佳解决方案。根据很多因素，包括问题大小、Solver 配置等因素、解决者配置等，最佳解决方案 可能是或 可能不是最佳 解决方案。

注意

提供给方法 解决(Solution) 的解决方案实例由 Solver 更改，但不要将其错误地用于最佳解决方案。

方法 解决(Solution) 或 getBestSolution （） 返回的解决方案实例最有可能是提供给方法 解决(Solution) 的规划克隆，这意味着它是一个不同的实例。

7.2. SOLVER 环境模式

solver 环境模式允许您检测实施中的常见错误。它不会影响日志级别。

一个 solver 有一个随机实例。有些 solver 配置使用随机实例，它比其他任何一个实例更多。例如，Simulated Annealing 算法高度依赖于随机数字，而 Tabu Search 仅依赖于它来解决分数绑定。环境模式会影响该随机实例的 seed。

您可以在 solver 配置 XML 文件中设置环境模式。以下示例设定 FAST_ASSERT 模式：

```
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <environmentMode>FAST_ASSERT</environmentMode>
  ...
</solver>
```

以下列表描述了您可以在 solver 配置文件中使用的环境模式：

- FULL_ASSERT 模式会打开所有断言，例如，每个移动分数计算未修正的断言，以及移动实施、约束、引擎本身等错误时失败。这个模式可以重现。它也是入侵的，因为它调用方法的 calculateScore （） 频率高于非assert 模式。FULL_ASSERT 模式非常慢，因为它不依赖于增量分数计算。

- NON_INTRUSIVE_FULL_ASSERT 模式将开启几个断言，在移动实施中的错误、约束、引擎本身等错误上快速失败。这个模式可以重现。它不是非入侵模式，因为它不会更频繁地调用方法 calculateScore （）。NON_INTRUSIVE_FULL_ASSERT 模式非常慢，因为它不依赖于增量分数计算。

- FAST_ASSERT 模式开启大多数断言，如 undoMove 的分数与在 Move 之前的断言相同，在移动实施、约束、引擎本身等错误时失败。这个模式可以重现。它也是入侵的，因为它调用方法的 calculateScore （） 频率高于非assert 模式。FAST_ASSERT 模式较慢。编写一个测试案例，它会在 FAST_ASSERT 模式下短暂运行您的计划问题。

- REPRODUCIBLE 模式是默认模式，因为它在开发过程中推荐使用。在这个模式中，两个以同一 OptaPlanner 版本运行，按照相同的顺序执行相同的代码。这两个运行在每个步骤中都有相同的结果，除非有以下备注。这可让您一致地重现错误。它还允许您对某些重构进行基准测试，如分数约束性能优化。

注意

虽然使用 REPRODCIBLE 模式，但出于以下原因，您的应用程序可能仍无法完全可重复生成：

- 使用 HashSet 或 其他集合 （在 JVM 运行之间具有不一致的顺序）用于规划实体或计划值的集合，特别是在解决方案实现中。使用 LinkedHashSet 替换它。
- 合并了时间的依赖算法，特别是 Simated Annealing 算法以及时间终止。分配的 CPU 时间有足够大的区别会影响时间逐步值。使用 Late Acceptance 算法替换 Simulated Annealing 算法，或者将终止时间替换为 step count termination。

- REPRODUCIBLE 模式可能比 NON_REPRODUCIBLE 模式稍慢。如果您的生产环境可从可重复性中受益，请在生产中使用此模式。在实践中，如果未指定 seed，则 REPRODUCIBLE 模式使用默认的固定随机 seed，同时禁用某些并发优化，如 work stealing。

- NON_REPRODUCIBLE 模式比 REPRODUCIBLE 模式稍快。避免在开发过程中使用它，因为它使调试和程序错误修复非常困难。如果您的生产环境中的可重复性不重要，请在生产中使用 NON_REPRODUCIBLE 模式。实际上，如果没有指定 seed，则此模式不使用固定随机的 seed。

7.3. 更改 OPTAPLANNER SOLVER 日志记录级别

您可以更改 OptaPlanner solver 中的日志级别，以检查解决者活动。以下列表描述了不同的日志记录级别：

- 错误 ：日志错误，除了作为 RuntimeException 丢弃给调用代码的除外。

  如果发生错误，OptaPlanner 通常会失败。它抛出子类 RuntimeException，其中包含调用代码的详细消息。为避免重复日志消息，它不会将其记录为错误。除非调用代码明确捕获并消除了 RuntimeException，否则 Thread's 默认"ExceptionHandler" 会将它记录为任何错误。同时，代码会造成进一步危害或模糊处理错误。

- 警告 ：日志可疑情况

- **info**：记录每个阶段和 solver 本身
- **debug**：记录每个阶段的每个步骤
- **trace**：记录每个阶段的每个步骤的每个移动

注意

指定 trace 日志记录会显著降低性能。但是，跟踪日志记录在开发过程中不可评估，以发现瓶颈。

即使调试日志记录在快速步骤算法（如 Late Acceptance 和 Simated Annealing）上可能会降低性能，但不适用于 Tabu Search 等较慢的步骤算法。

trace 的和 debug 日志都会导致在多线程中通过大多数附加器进行阻塞。

在 Eclipse 中，对控制台的调试日志往往会导致导致分数计算速度超过 10000 每秒的拥塞。IntelliJ 或 Maven 命令行都不受此问题的影响。

流程

将日志记录级别设置为 debug 日志记录，以查看阶段何时结束以及执行快速的步骤。

以下示例显示了 debug 日志的输出：

```
INFO  Solving started: time spent (3), best score (-4init/0), random (JDK with seed 0).
DEBUG     CH step (0), time spent (5), score (-3init/0), selected move count (1), picked move
(Queen-2 {null -> Row-0}).
DEBUG     CH step (1), time spent (7), score (-2init/0), selected move count (3), picked move
(Queen-1 {null -> Row-2}).
DEBUG     CH step (2), time spent (10), score (-1init/0), selected move count (4), picked move
(Queen-3 {null -> Row-3}).
DEBUG     CH step (3), time spent (12), score (-1), selected move count (4), picked move (Queen-0
{null -> Row-1}).
INFO  Construction Heuristic phase (0) ended: time spent (12), best score (-1), score calculation
speed (9000/sec), step total (4).
DEBUG     LS step (0), time spent (19), score (-1),     best score (-1), accepted/selected move count
(12/12), picked move (Queen-1 {Row-2 -> Row-3}).
```

```
DEBUG    LS step (1), time spent (24), score (0), new best score (0), accepted/selected move count (9/12), picked move (Queen-3 {Row-3 -> Row-2}).
INFO  Local Search phase (1) ended: time spent (24), best score (0), score calculation speed (4000/sec), step total (2).
INFO  Solving ended: time spent (24), best score (0), score calculation speed (7000/sec), phase total (2), environment mode (REPRODUCIBLE).
```

所有花费的时间都以毫秒为单位。

所有信息都记录到 SLF4J，它是一个简单的日志记录传真，它将每个日志消息委派给 Logback、Apache Commons Logging、Log4j 或 java.util.logging。向您选择的日志记录框架添加依赖项。

7.4. 使用 LOGBACK 记录 OPTAPLANNER SOLVER 活动

Logback 是推荐与 OptaPlanner 搭配使用的日志记录框架。使用 Logback 记录 OptaPlanner solver 活动。

先决条件

- 您有一个 OptaPlanner 项目。

流程

1. 将以下 Maven 依赖项添加到 OptaPlanner 项目的 pom.xml 文件中：

注意

您不需要添加额外的网桥依赖项。

```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.x</version>
</dependency>
```

2. 在 logback.xml 文件中的 org.optaplanner 软件包上配置日志级别，如下例所示，其中 < LEVEL> 是 第 7.4 节 “使用 Logback 记录 OptaPlanner solver 活动” 中列出的日志级别。

```
<configuration>
```

```
  <logger name="org.optaplanner" level="<LEVEL>"/>

  ...

</configuration>
```

3. 可选：如果您有一个多租户应用程序，其中多个 Solver 实例可能会同时运行，请将每个实例的日志记录分开到单独的文件中：

   a. 与 映射的诊断上下文 (MDC)周围的 solve （） 调用：

```
MDC.put("tenant.name",tenantName);
MySolution bestSolution = solver.solve(problem);
MDC.remove("tenant.name");
```

   b. 将您的日志记录器配置为为每个 ${tenant.name} 使用不同的文件。例如，在 logback.xml 文件中使用 SiftingAppender ：

```
<appender name="fileAppender" class="ch.qos.logback.classic.sift.SiftingAppender">
  <discriminator>
    <key>tenant.name</key>
    <defaultValue>unknown</defaultValue>
  </discriminator>
  <sift>
    <appender name="fileAppender.${tenant.name}" class="...FileAppender">
      <file>local/log/optaplanner-${tenant.name}.log</file>
      ...
    </appender>
  </sift>
</appender>
```

注意

当运行多个 solvers 或一个多线程解决时，大多数附加程序（包括控制台）会导致 debug 和 trace 日志记录。切换到 async 附加器以避免出现这个问题或关闭 调试日志记录。

4. 如果 OptaPlanner 无法识别新级别，请临时添加系统属性 -Dlogback.LEVEL=true 进行故障排除。

7.5. 使用 LOG4J 记录 OPTAPLANNER SOLVER 活动

如果您已使用 Log4J，且您不想切换到更快速的成功者，则您可以为 Log4J 配置 OptaPlanner 项目。

先决条件

- 您有一个 OptaPlanner 项目
- 您可以使用 Log4J 日志记录框架

流程

1. 将网桥依赖项添加到项目 pom.xml 文件中：

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.x</version>
</dependency>
```

2. 在 log4j.xml 文件中的软件包 org.optaplanner 上配置日志级别，如下例所示，其中 < LEVEL> 是 第 7.4 节 “使用 Logback 记录 OptaPlanner solver 活动” 中列出的日志级别。

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

 <category name="org.optaplanner">
   <priority value="<LEVEL>" />
 </category>

 ...

</log4j:configuration>
```

3. 可选：如果您有一个多租户应用程序，其中多个 Solver 实例可能会同时运行，请将每个实例的日志记录分开到单独的文件中：

   a. 与 映射的诊断上下文 (MDC)周围的 solve （） 调用：

```
MDC.put("tenant.name",tenantName);
MySolution bestSolution = solver.solve(problem);
MDC.remove("tenant.name");
```

   b.

将您的日志记录器配置为为每个 ${tenant.name} 使用不同的文件。例如，在 logback.xml 文件中使用 SiftingAppender ：

```
<appender name="fileAppender" class="ch.qos.logback.classic.sift.SiftingAppender">
  <discriminator>
    <key>tenant.name</key>
    <defaultValue>unknown</defaultValue>
  </discriminator>
  <sift>
    <appender name="fileAppender.${tenant.name}" class="...FileAppender">
      <file>local/log/optaplanner-${tenant.name}.log</file>
      ...
    </appender>
  </sift>
</appender>
```

注意

当运行多个 solvers 或一个多线程解决时，大多数附加程序（包括控制台）会导致 debug 和 trace 日志记录。切换到 async 附加器以避免出现这个问题或关闭 调试日志记录。

7.6. 监控解决方案

OptaPlanner 通过 Micrometer (Java 应用程序的指标检测库)公开指标。您可以使用带有流行监控系统的 Micrometer 来监控 OptaPlanner solver。

7.6.1. 为 Micrometer 配置 Quarkus OptaPlanner 应用程序

要将 OptaPlanner Quarkus 应用程序配置为使用 Micrometer 和指定的监控系统，请将 Micrometer 依赖项添加到 pom.xml 文件中。

先决条件

- 您有一个 Quarkus OptaPlanner 应用程序。

流程

1. 在应用程序的 pom.xml 文件中添加以下依赖项，其中 < MONITORING_SYSTEM& gt; 是 Micrometer 和 Quarkus 支持的监控系统：

注意

Prometheus 目前是唯一由 Quarkus 支持的监控系统。

```
<dependency>
 <groupId>io.quarkus</groupId>
 <artifactId>quarkus-micrometer-registry-<MONITORING_SYSTEM></artifactId>
</dependency>
```

2. **要在开发模式下运行应用程序，请输入以下命令：**

```
mvn compile quarkus:dev
```

3. **要查看应用程序的指标，请在浏览器中输入以下 URL：**

```
http://localhost:8080/q/metrics
```

7.6.2. 为 Micrometer 配置 Spring Boot OptaPlanner 应用程序

要将 Spring Boot OptaPlanner 应用配置为使用 Micrometer 和指定的监控系统，请将 Micrometer 依赖项添加到 pom.xml 文件中。

先决条件

- **您有一个 Spring Boot OptaPlanner 应用程序。**

流程

1. **将以下依赖项添加到应用程序的 pom.xml 文件中，其中 < MONITORING_SYSTEM& gt; 是 Micrometer 和 Spring Boot 支持的监控系统：**

```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
 <groupId>io.micrometer</groupId>
 <artifactId>micrometer-registry-<MONITORING_SYSTEM></artifactId>
</dependency>
```

2.

在应用程序的 application.properties 文件中添加配置信息。如需更多信息，请参阅 Micrometer 网站。

3. 要运行应用程序，请输入以下命令：

```
mvn spring-boot:run
```

4. 要查看应用程序的指标，请在浏览器中输入以下 URL：

http://localhost:8080/actuator/metrics

注意

使用以下 URL 作为 Prometheus scraper 路径：
http://localhost:8080/actuator/prometheus

7.6.3. 为 Micrometer 配置普通 Java OptaPlanner 应用程序

要将普通 Java OptaPlanner 应用配置为使用 Micrometer 应用程序，您必须将您选择的监控系统的 Micrometer 依赖项和配置信息添加到项目的 POM.XML 文件中。

先决条件

- 您有一个普通 Java OptaPlanner 应用程序。

流程

1. 将以下依赖项添加到应用程序的 pom.xml 文件中，其中 < MONITORING_SYSTEM& gt; 是一个监控系统，它被配置为 Micrometer，<VERSION > 是您要使用的 Micrometer 的版本：

```
<dependency>
 <groupId>io.micrometer</groupId>
 <artifactId>micrometer-registry-<MONITORING_SYSTEM></artifactId>
 <version><VERSION></version>
</dependency>
<dependency>
 <groupId>io.micrometer</groupId>
 <artifactId>micrometer-core</artifactId>
 <version><VERSION></version>
</dependency>
```

2. 将监控系统的 Micrometer 配置信息添加到项目的 pom.xml 文件的开头。如需更多信息，请参阅 Micrometer 网站。

3. 在配置信息下添加以下行，其中 <MONITORING_SYSTEM> 是您添加的监控系统：

```
Metrics.addRegistry(<MONITORING_SYSTEM>);
```

以下示例演示了如何添加 Prometheus 监控系统：

```
PrometheusMeterRegistry prometheusRegistry = new
PrometheusMeterRegistry(PrometheusConfig.DEFAULT);
try {
    HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
    server.createContext("/prometheus", httpExchange -> {
        String response = prometheusRegistry.scrape();
        httpExchange.sendResponseHeaders(200, response.getBytes().length);
        try (OutputStream os = httpExchange.getResponseBody()) {
            os.write(response.getBytes());
        }
    });
    new Thread(server::start).start();
} catch (IOException e) {
    throw new RuntimeException(e);
}
Metrics.addRegistry(prometheusRegistry);
```

4. 打开您的监控系统，以查看您的 OptaPlanner 项目的指标。公开以下指标：

注意

指标的名称和格式因 registry 而异。

- OptaPlanner.solver.errors.total ：自测量开始以来发生的错误总数。
- OptaPlanner.solver.solve-length.active-count: 当前解决的 solvers 的数量。
- OptaPlanner.solver.solve-length.seconds-max: 最长运行的当前活跃 addressr 的运行时。

- OptaPlanner.solver.solve-length.seconds-duration-sum: 每个活跃的 solver 的 solve 持续时间总和。例如，如果有两个活跃的解决问题，则运行三分钟，另一个在一分钟内，总解决时间为四分钟。

7.6.4. 其他指标

如需更详细的监控，您可以在 solver 配置中配置 OptaPlanner，以性能成本监控其他指标。以下示例使用 BEST_SCORE 和 SCORE_CALCULATION_COUNT 指标：

```
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <monitoring>
    <metric>BEST_SCORE</metric>
    <metric>SCORE_CALCULATION_COUNT</metric>
    ...
  </monitoring>
  ...
</solver>
```

您可以在此配置中启用以下指标：

- SOLVE_DURATION （默认启用）（默认为 Micrometer 量表 ID: optaplanner.solver.solve.duration）：衡量在进行最长的活跃解决者、活跃解决的问题者数以及所有活跃解决问题者的累计持续时间。
- ERROR_COUNT （默认启用的 Micrometer 量表 ID: optaplanner.solver.errors）：衡量在解决时发生的错误数量。
- SCORE_CALCULATION_COUNT （默认启用），Micrometer 计量 ID: optaplanner.solver.score.calculation.count：衡量所执行的分数计算的数量。
- BEST_SCORE (Micrometer meter ID: optaplanner.solver.best.score mdadm)：衡量目前 OptaPlanner 的最佳解决方案分数。每个分数级别都有单独的量表。例如，对于 HardSoftScore，有 optaplanner.solver.best.score.hard.score 和 optaplanner.solver.best.score.soft.score 量表。
- STEP_SCORE (Micrometer meter ID: optaplanner.solver.step.scoremdadm)：衡量 OptaPlanner 执行的每个步骤分数。每个分数级别都有单独的量表。例如，对于

HardSoftScore，有 optaplanner.solver.step.score.hard.score 和 optaplanner.solver.step.score.soft.score 量表。

- *BEST_SOLUTION_MUTATION (Micrometer 计量 ID: optaplanner.solver.best.solution.mutation)：连续最佳解决方案之间更改的规划变量数量。*

- *MOVE_COUNT_PER_STEP (Micrometer meter ID: optaplanner.solver.step.move.count)：衡量步骤中评估的移动数量。*

- *MEMORY_USE (Micrometer 量表 ID： jvm.memory.used)：衡量 JVM 中使用的内存量。此指标不测量 solver 使用的内存量；同一 JVM 上的两个 solvers 将报告此指标的值。*

- *CONSTRAINT_MATCH_TOTAL_BEST_SCORE (Micrometer meter ID: optaplanner.solver.constraint.match.best.score)：衡量 OptaPlanner 上每个约束的分数影响。每个分数级别都有单独的量表，每个约束的标签。例如，对于软件包 "com.example" 的限制 "Minimize Cost" 的 HardSoftScore，软件包 "com.example" 有 optaplanner.solver.constraint.match.best.score.hard .score 和 optaplanner.solver.constraint.match.best.score 量表，标签为 "constraint.package=com.example" 和 "constraint.name=Minimize"。*

- *CONSTRAINT_MATCH_TOTAL_STEP_SCORE (Micrometer 计量 ID: optaplanner.solver.constraint.step.step.score)：衡量当前步骤中每个约束的分数影响。每个分数级别都有单独的量表，每个约束的标签。例如，对于软件包 "com.example" 的限制 "Minimize Cost" 的 HardSoftScore，带有标签 "constraint.package=com .constraint.step.step.score.hard.score 和 optaplanner.solver.constraint.match.step.score.soft.score 量表，标签为 "constraint.package=com.example" 和 "constraint.name=Minize"。*

- *PICKED_MOVE_TYPE_BEST_SCORE_DIFF （Micrometer meter ID: optaplanner.solver.move.type.best.score.diffmdadm：测量特定类型可以改进最佳解决方案。每个分数级别都有单独的计量，移动类型的标签。例如，对于一个进程的计算机，对于 HardSoftScore 和 ChangeMove，有一个带有标签 move.type. bester.move.type.best.score.diff.hard.score 和 optaplanner.solver.move.type.best.score.diff.soft.score 量表，标签为 move.type=ChangeMove (Process.computer)。*

- *PICKED_MOVE_TYPE_STEP_SCORE_DIFF （Micrometer meter ID: optaplanner.solver.move.type.step.score.diffruby：测量特定移动类型可以改进最佳解决方案。每个分数级别都有单独的计量，移动类型的标签。例如，对于一个进程的计算机，对于 HardSoftScore 和 ChangeMove，有一个带有 tag move.type. solver.move.type.step.score.diff.hard.score 和*

optaplanner.solver.move.type.step.score.diff.soft.score 量(label) move.type=ChangeMove (Process.computer)。

7.7. 配置随机数字生成器

许多 Heuristics 和 metaheuristics 依赖于伪随机数字生成器来移动选择，以解决分数绑定、基于迁移接受的可能性等。在解决期间，重复利用相同的随机实例，以提高随机值的可重复性、性能和统一分布。

随机 seed 是一个用于初始化伪随机数生成器的数字。

流程

1. 可选：要更改随机实例的随机 seed，请指定一个 randomSeed ：

```
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <randomSeed>0</randomSeed>
  ...
</solver>
```

2. 可选：要更改伪随机数生成器实现，请为以下 solver 配置文件中列出的 randomType 属性指定一个值，其中 < RANDOM_NUMBER_GENERATOR& gt; 是一个伪随机数生成器：

```
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <randomType><RANDOM_NUMBER_GENERATOR></randomType>
  ...
</solver>
```

支持以下伪随机数生成器：

- JDK （默认）：标准随机数生成器实施(java.util.Random)
- MERSENNE_TWISTER: Random number generator implementation by Commons Math

- *WELL512A,WELL1024A,WELL19937A,WELL19937C,WELL44497A 和 WELL44497B: Random number generator implementation by Commons Math*

对于大多数用例，randomType 属性的值不会影响到多个数据集上最佳解决方案的平均质量。

第 8 章 OPTAPLANNER SOLVERMANAGER

Solver Manager 是用于一个或多个 Solver 实例的传真，简化了 REST 和其他企业服务中的规划问题。

与 Solver.solve (…) 方法不同，SolverManager 具有以下特征：

- SolverManager.solve (…) 立即返回：它会调度问题以异步解决，而不阻止调用线程。这可避免 HTTP 和其他技术的超时问题。
- SolverManager.solve (…) 可以并行解决同一域的多个规划问题。

在内部，SolverManager 管理一个 solver 线程的线程池，该线程调用 Solver.solve (…)，以及处理最佳解决方案更改事件的线程池。

在 Quarkus 和 Spring Boot 中，SolverManager 实例会自动注入您的代码中。如果您使用 Quarkus 或 Spring Boot 以外的平台，请使用 create (…) 方法构建 SolverManager 实例：

```
SolverConfig solverConfig =
SolverConfig.createFromXmlResource(".../cloudBalancingSolverConfig.xml");
SolverManager<CloudBalance, UUID> solverManager = SolverManager.create(solverConfig,
new SolverManagerConfig());
```

提交到 SolverManager.solve (…) 方法的每个问题都必须具有唯一的问题 ID。之后调用 getSolverStatus (problemId) 或 terminateEarly (problemId) 使用问题 ID 来区分计划问题。问题 ID 必须是不可变类，如 Long、String 或 java.util.UUID。

SolverManagerConfig 类具有一个 parallelSolverCount 属性，用于控制并行运行多少个 solvers。例如，如果 parallelSolverCount 属性设置为 4，并且您提交五个问题，则四个问题会立即解决，并且第五个问题在其中一个问题结束时开始。如果这些问题解决五分钟，则第五个问题需要花费 10 分钟才能完成。默认情况下，parallelSolverCount 设置为 AUTO，它解析为一半的 CPU 内核，而不考虑 solvers 的 moveThreadCount。

要检索最佳解决方案，在解决时通常会使用 SolverJob.getFinalBestSolution （） ：

```
CloudBalance problem1 = ...;
UUID problemId = UUID.randomUUID();
```

```
// Returns immediately
SolverJob<CloudBalance, UUID> solverJob = solverManager.solve(problemId, problem1);
...
CloudBalance solution1;
try {
    // Returns only after solving terminates
    solution1 = solverJob.getFinalBestSolution();
} catch (InterruptedException | ExecutionException e) {
    throw ...;
}
```

但是，有更好的方法，在用户需要解决方案前解决批处理问题，以及在用户主动等待解决方案时进行实时解决。

当前的 SolverManager 实施在单个计算机节点上运行，但未来的工作旨在在云之间分发 solver 负载。

8.1. 批处理解决问题

批量解决正在并行解决多个数据集。批量解决在夜间特别有用：

- 夜中通常有一些或没有问题更改。有些机构强制使用截止时间，例如，在 午夜前提交所有日期的请求。
- solvers 可以运行更长的时间（通常小时），因为 nobody 正在等待结果和 CPU 资源通常更便宜。
- 当员工达到下一个工作时，可以使用解决方案。

流程

要并行解决问题，请受 parallelSolverCount 的限制，请对每个数据集调用 solve (...) 创建了以下类：

```
public class TimeTableService {

    private SolverManager<TimeTable, Long> solverManager;

    // Returns immediately, call it for every data set
    public void solveBatch(Long timeTableId) {
        solverManager.solve(timeTableId,
                // Called once, when solving starts
                this::findById,
```

```
            // Called once, when solving ends
            this::save);
    }

    public TimeTable findById(Long timeTableId) {...}

    public void save(TimeTable timeTable) {...}

}
```

8.2. 解决并侦听显示进度

当用户在等待解决方案时运行解决者时，用户可能需要等待几分钟或几小时才能收到结果。为确保一切正常，通过显示最佳解决方案并获得最佳分数来显示进度。

流程

1. 要处理中间最佳解决方案，请使用 solveAndListen (...)：

```
public class TimeTableService {

    private SolverManager<TimeTable, Long> solverManager;

    // Returns immediately
    public void solveLive(Long timeTableId) {
        solverManager.solveAndListen(timeTableId,
                // Called once, when solving starts
                this::findById,
                // Called multiple times, for every best solution change
                this::save);
    }

    public TimeTable findById(Long timeTableId) {...}

    public void save(TimeTable timeTable) {...}

    public void stopSolving(Long timeTableId) {
        solverManager.terminateEarly(timeTableId);
    }

}
```

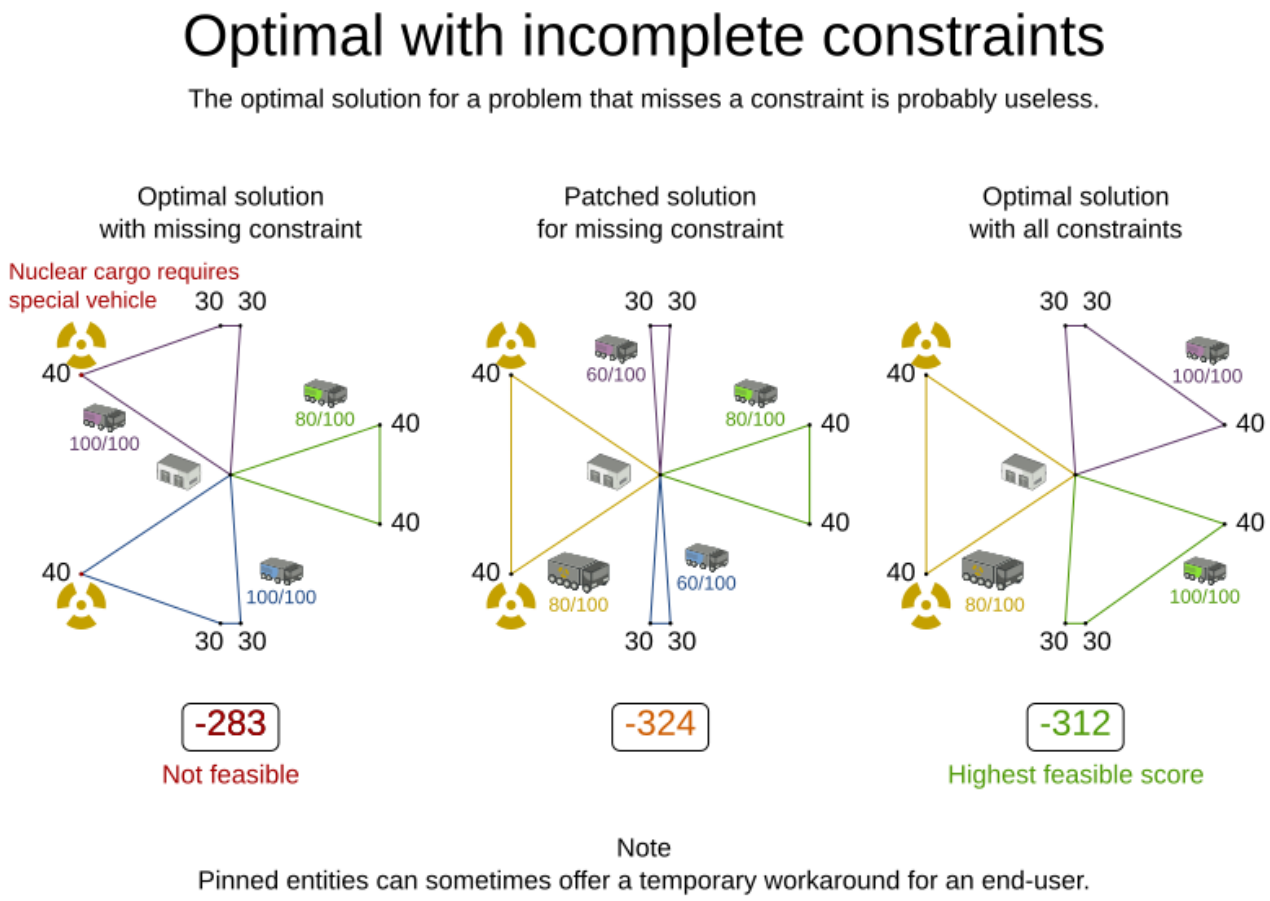
此实施是使用数据库与 UI 通信，它会轮询数据库。更高级的实现将最佳解决方案直接推送到 UI 或消息传递队列。

2. 当用户满足中间最佳解决方案且不想等待更好解决方案时，请调用 SolverManager.terminateEarly (problemId)。

部分 IV. OPTAPLANNER 分数计算

每个 @PlanningSolution 类都有一个分数。分数是比较两个解决方案的目标方法。分数较高的解决方案更好。该方案旨在找到所有可能解决方案中最高分数的解决方案。最佳解决方案 是解决解决者在解决过程中遇到的最高分数 的解决方案，这可能是最佳解决方案。

OptaPlanner 无法自动知道哪个解决方案最适合您的业务，因此您必须告诉如何根据您的业务计算指定 @PlanningSolution 实例的分数。如果您忘记或无法实现重要的业务约束，解决方案可能无用处，如下图所示：



第 9 章 OPTAPLANNER 中的业务限制

使用业务限制来限制方案中的条件。这些条件可能基于现有业务合同、资源可用性、员工偏好或业务规则。要在 OptaPlanner 中实施业务约束，业务约束必须正式化为分数约束。OptaPlanner 中提供的以下分数属性提供灵活的解决方案：

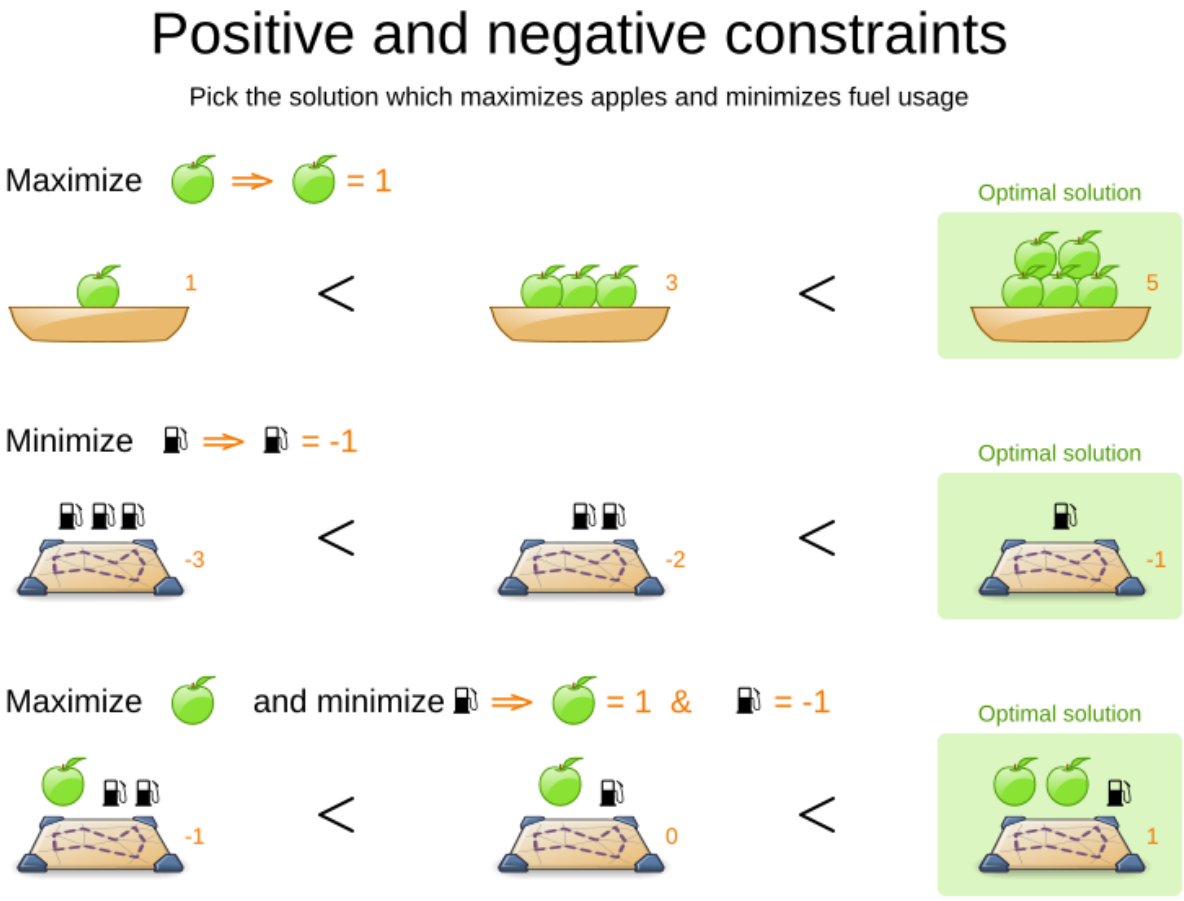
- score signum: 使约束类型正或负数
- 分数权重 ：将成本或利润置于约束类型
- 分数级别（硬、软等）：对一组约束类型的优先级

注意

不假定您的企业事先知道其所有分数限制。期望在第一个发行版本后添加、更改或删除分数限制。

9.1. 负和正分数限制

所有分数技术都基于约束。约束可以是一个简单的模式，如 Maximize the apple harvest in the solution or a more complex pattern。约束可以是负的或正数。正约束是您要最大化的限制。负约束是您要最小化的限制。



此镜像说明了最佳解决方案始终具有最高的分数，无论约束是正还是负数。

大多数规划问题仅具有负限制，因此分数为负数。在这种情况下，分数是负约束的权重总和，完美分数为 0。例如，在 N Queens 问题中，分数是可以相互攻击的 queen 对数的负数。您可以组合负和正限制，即使在同一分数级别也是如此。

当约束在特定计划实体集上激活时，因为负约束中断或满足正约束，它称为 约束匹配。

9.2. 分数约束权重

并非所有分数限制都同样重要。如果一次中断一个约束，与多次破坏另一个约束相同，则这两个限制具有不同的权重，即使它们处于相同的分数级别。

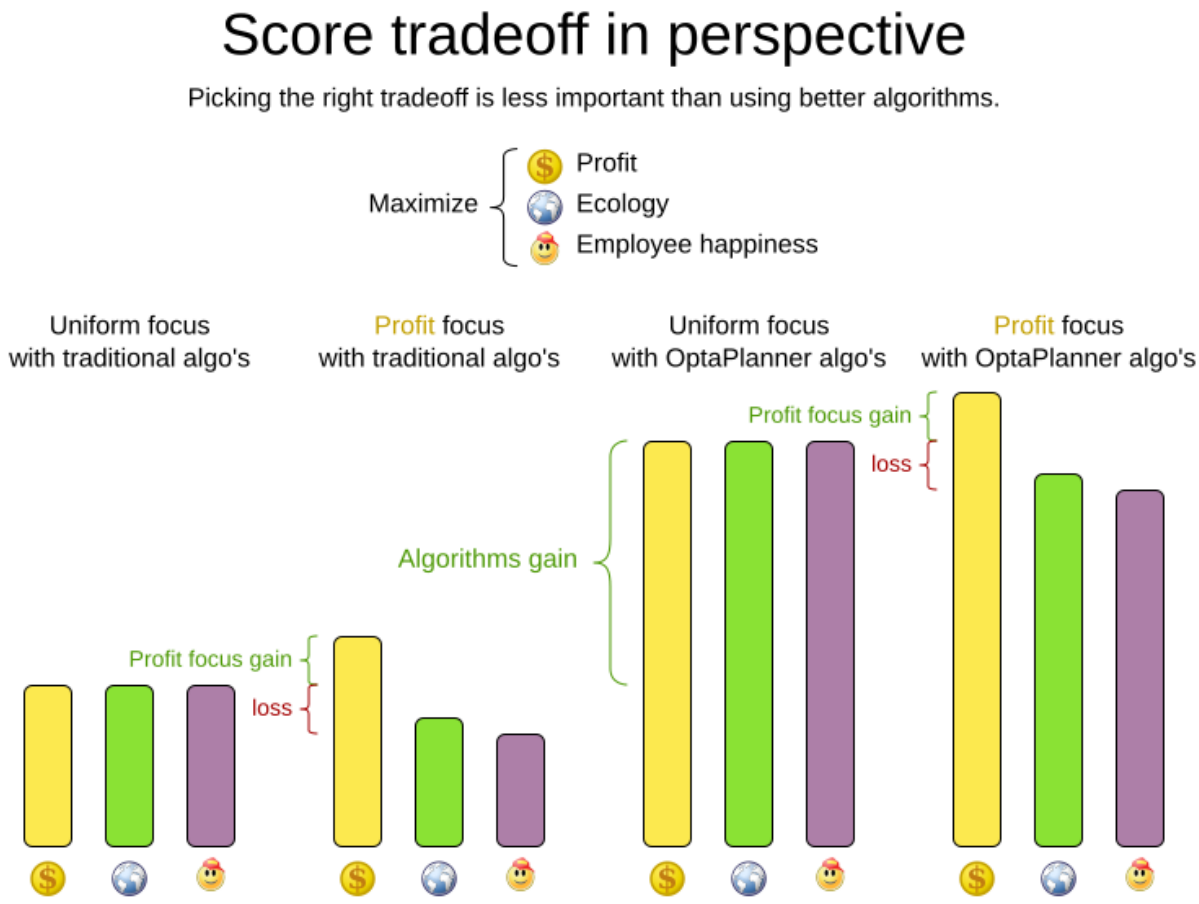
在您可以为所有用例分配成本时，分数权重非常简单。在这种情况下，正限制最大收益，负限制可最大程度降低费用，并将它们一起最大限度地提高利润。

或者，分数权重也通常用于创建社交公平。例如，要求免费一天向新年度支付更高的权重，超过正常日期。

约束匹配的权重可能取决于涉及的计划实体。例如，在云平衡中，软约束的权重与活跃计算机匹配的权重是该计算机的维护成本，每个计算机的维护成本不同。

在约束上放置良好权重通常是非常困难的决定，因为它对于其他限制做出选择和权衡。不同的利益相关者有不同的优先级。

在实施开始时，不要浪费约束权重讨论。相反，添加 @constraintConfiguration 注释，并允许用户通过 UI 更改它们。不准确的权重比 mediocre 算法小，如下图所示：



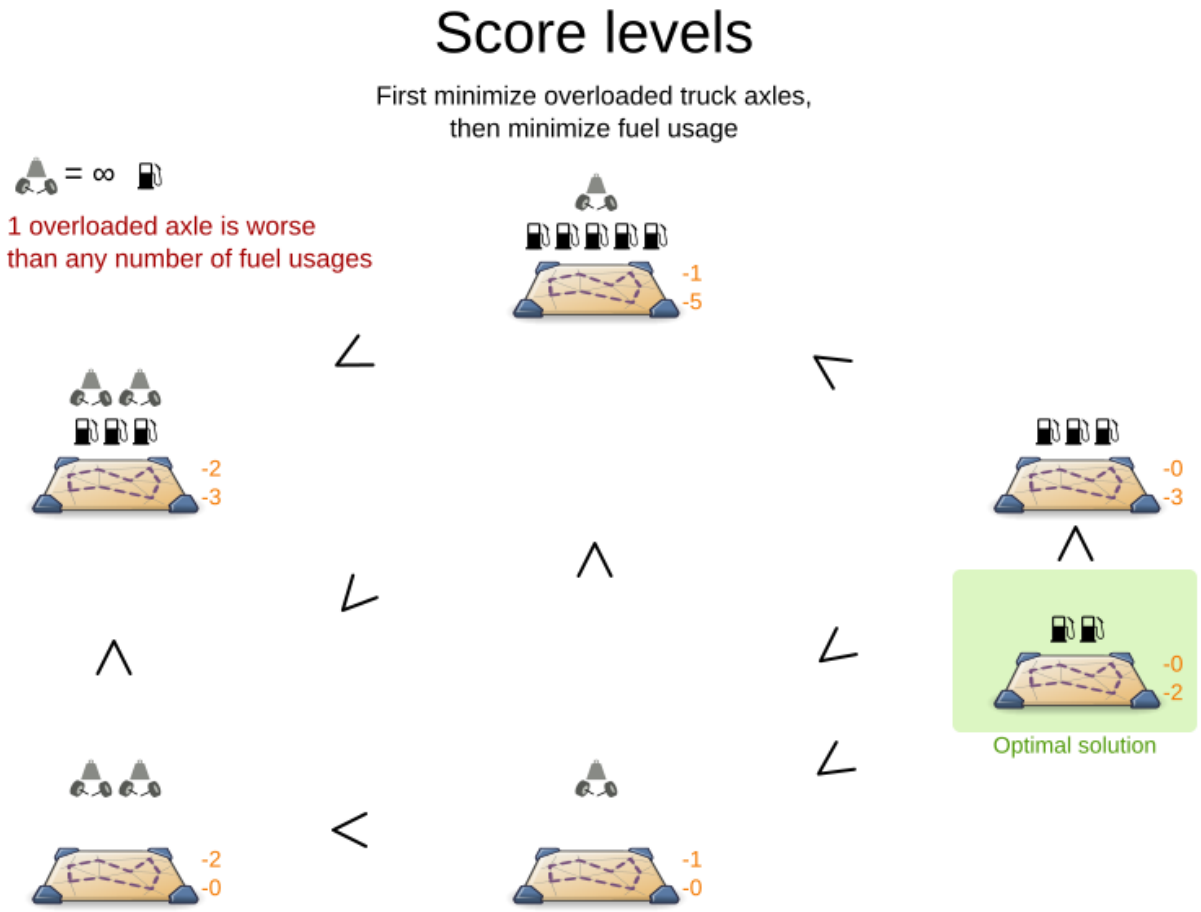
大多数用例都使用带有 int 权重的分数，如 HardSoftScore。

9.3. 分数约束级别

有时，分数约束为另一个分数约束，无论后者中断了多少。在这种情况下，这些分数限制在不同的级

别。例如，由于物理现实的限制，nurse 无法同时进行两次转换，因此此约束会降低所有 nurse happiness 约束。

大多数用例仅有两个分数级别，即 hard 和 soft。按顺序比较两个分数的级别。首先比较第一个分数级别。如果两个分数不同，则忽略剩余的分数级别。例如，破坏 0 个硬约束和 1000000 软限制的分数比中断 1 硬约束和 0 软限制的分数更高。

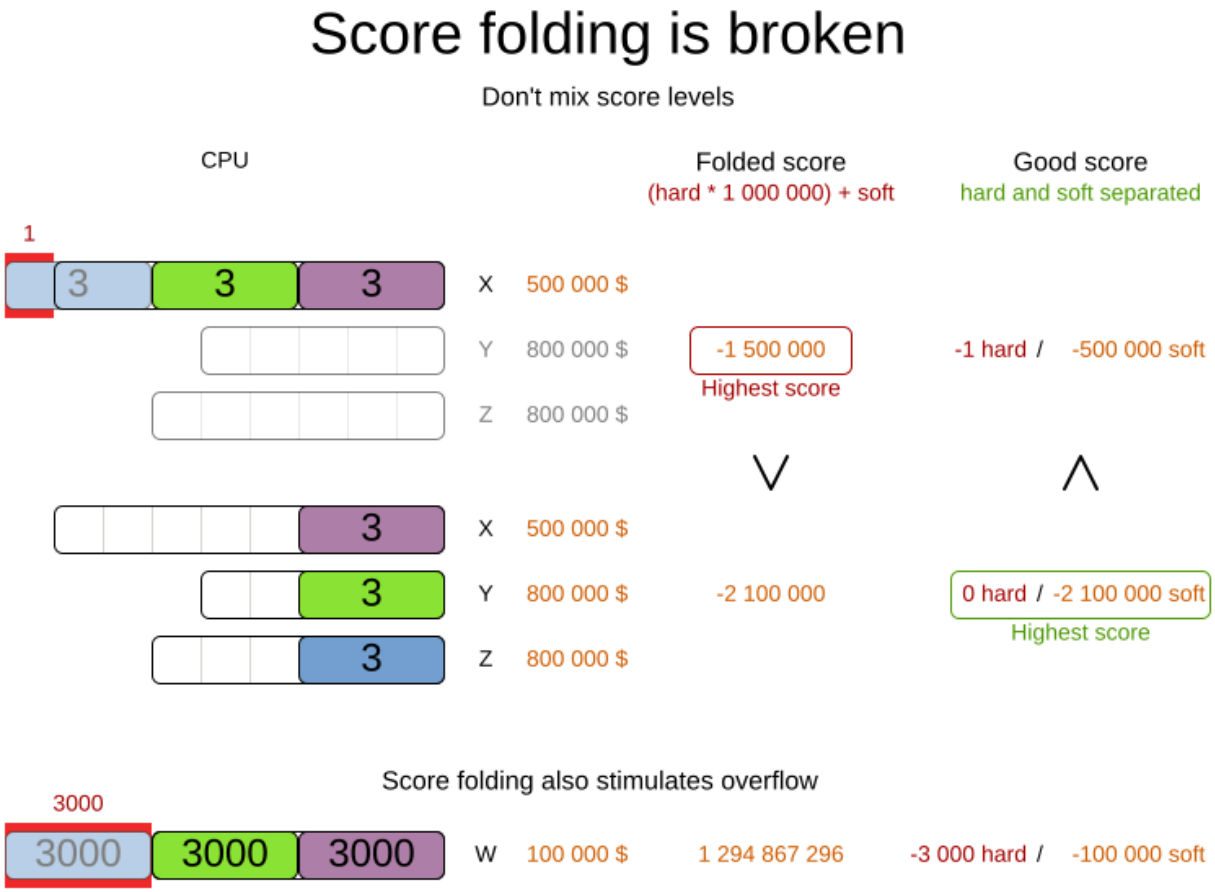


如果分数级别或两个以上分数级别，如果没有硬限制，则分数是 可行的。

注意

默认情况下，OptaPlanner 为计划值分配所有计划变量。如果没有可行的解决方案，这意味着最佳解决方案是不可行的。要保留某些计划实体没有被分配，请对受限的计划应用。

对于每个约束，您必须选择一个分数级别、分数权重和分数号。例如，-1 soft 的分数级别为，权重为 1，负符号为。当您的业务确实需要不同的分数级别时，请勿使用大约束权重。这个临时解决方案（称为分数折叠）无法正常工作：

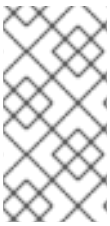


注意

您的业务可能会告诉您所有硬限制都有相同的权重，因为它们无法被破坏，因此权重无关紧要。这不是正确的。如果特定数据集不存在可行的解决方案，则业务可以使用最小的解决方案来估算其缺少多少个业务资源。例如，在云平衡问题中，最可行的解决方案可以发现需要多少新计算机。

另外，如果您的所有硬约束都有相同的权重，您可能会创建一个分数陷阱。例如，如果计算机在其进程中有 7 个太多 CPU，那么在云平衡问题中，如果计算机在其进程中，必须有 7 倍，就像它只有一个 CPU 太少。

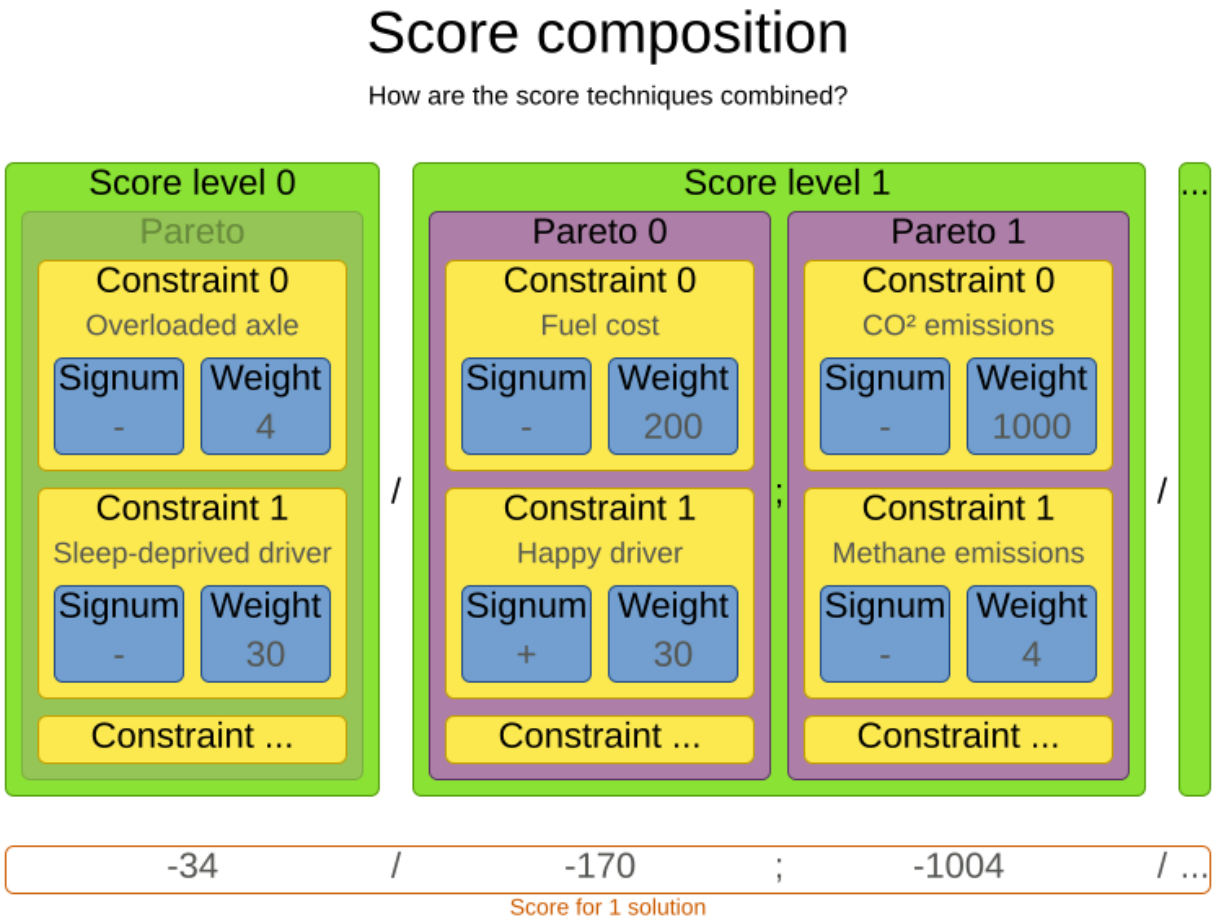
OptaPlanner 也支持三个或更多分数级别。例如，公司可能决定营收员工满意度，反之亦然，而这两个限制则受到物理现实的局限性。



注意

要建模公平或负载平衡，您不需要使用大量分数级别，即使 OptaPlanner 可以处理许多分数级别。

大多数用例都使用具有两个或三个权重的分数，如 HardSoftScore 和 HardMediumSoftScore。您可以组合所有这些技术，看似无疑：

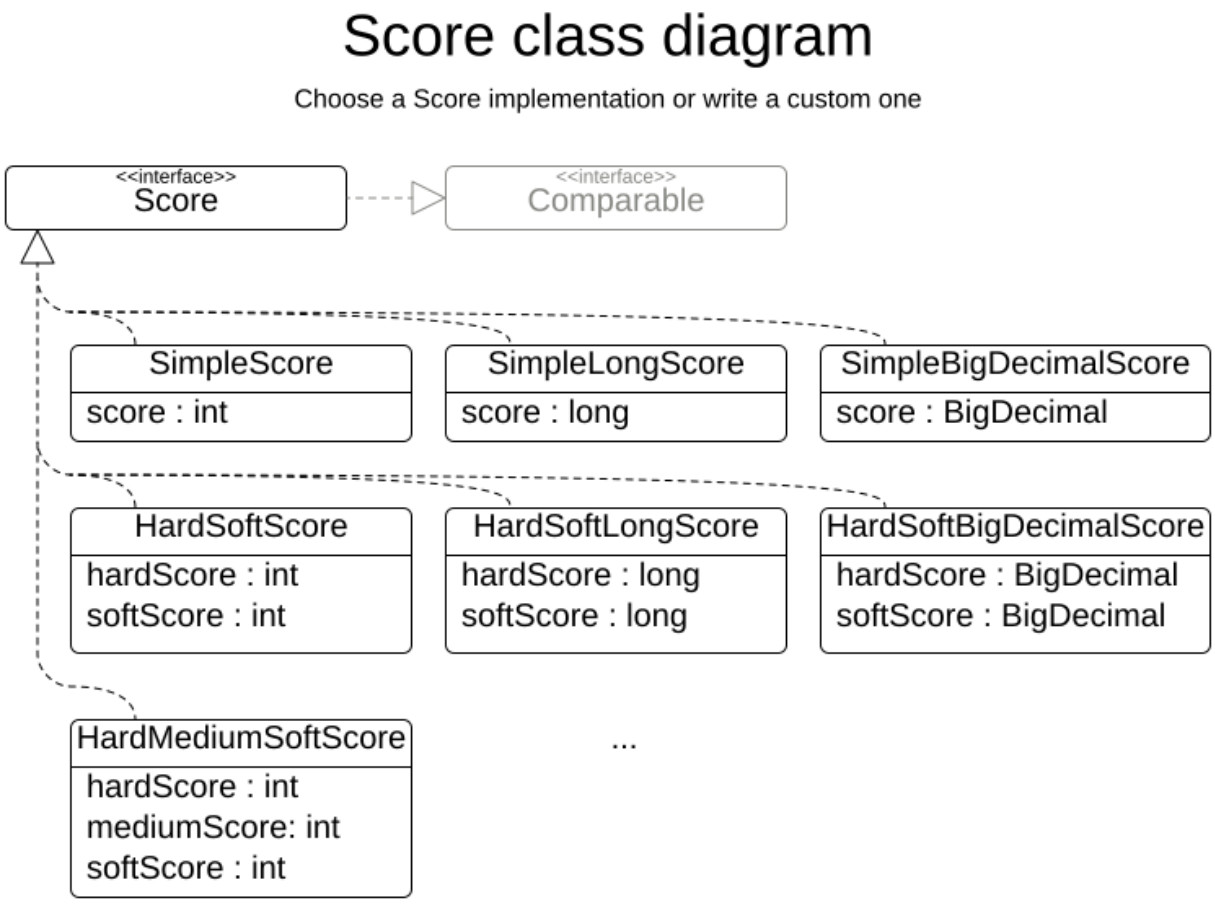


第 10 章 OPTAPLANNER SCORE 接口

分数由 Score 接口表示，它扩展了 Comparable 接口：

```
public interface Score<...> extends Comparable<...> {
    ...
}
```

要使用的分数实施取决于您的用例。您的分数可能不适用于一个 长 值。OptaPlanner 有几个内置分数实现，但您也可以实施自定义分数。大多数用例都使用内置的 HardSoftScore 分数。



所有分数实施也都有一个 initScore (int)。它主要用于内部使用 OptaPlanner：它是未初始化的计划变量的负数。从用户的角度来说，这是 0， 除非在它能够初始化所有规划变量之前终止建筑高度。在本例中， Score .isSolutionInitialized （） 返回 false。

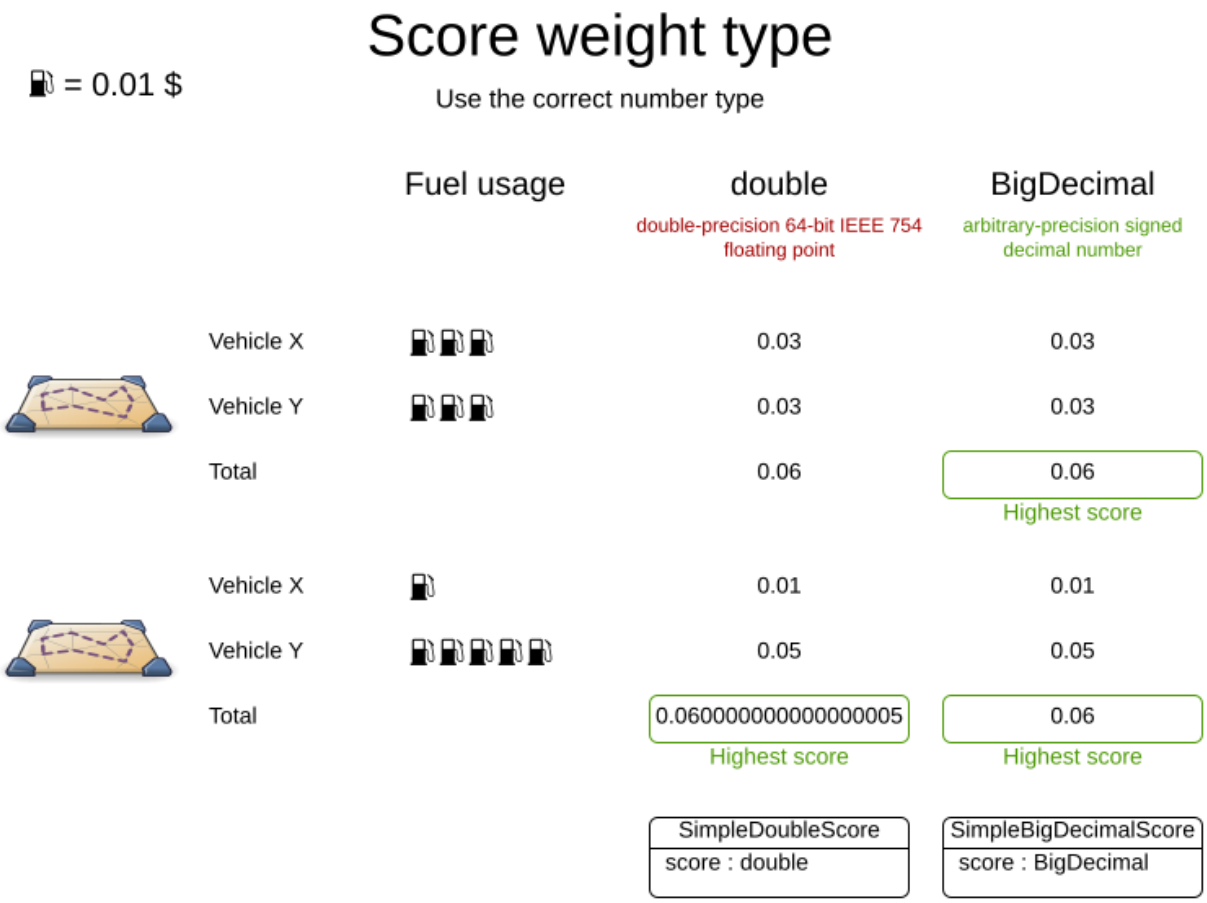
在解析器运行时，分数实施（如 HardSoftScore）必须相同。分数实现在解决方案域类中配置：

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

```
    ...

    @PlanningScore
    private HardSoftScore score;

}
```

10.1. 分数计算中的浮点号

避免使用浮点数类型 float 或分数 计算中的双倍。改为使用 BigDecimal 或 scale long。浮点号无法正确表示十进制数字。例如，双引号 无法正确包含 0.05 值。相反，它包含最接近代表的值。算术，包括增加和减去，使用浮点号（特别是规划问题）会导致决策不正确，如下图所示：



另外，浮动点数添加不关联：

```
System.out.println( ((0.01 + 0.02) + 0.03) == (0.01 + (0.02 + 0.03)) ); // returns false
```

这会导致 分数损坏。

十进制数字(BigDecimal)没有这些问题。

注意

巨型算术性比 int、长 或 双 算算术要慢得多。在某些试验中，分数计算需要更长的时间。

因此，在很多情况下，可能需要将单个分数权重 的所有 数字乘以十十，因此分数权重适合扩展的 int 或 长。例如，如果您将所有权重乘以 1000，则 0.07 的 fuelCost 变得 fuelCostMillis 为 70，并且不再使用十进制分数权重。

10.2. 分数计算类型

计算解决方案分数的方法有几种：

- 简单 Java 分数计算 ：以 Java 或其他 JVM 语言通过单一方法实施所有限制。这个方法无法扩展。
- 约束 流分数计算 ：将每个约束实施为 Java 或其他 JVM 语言的独立约束流。这个方法快速且可扩展的。
- 增量 Java 分数计算 （不推荐）：使用 Java 或其他 JVM 语言实施多种低级别方法。这个方法快速、可扩展，但很难实施和维护。
- Drools 分数计算（已弃用）：将每个约束作为 DRL 中的单独分数规则实施。这个方法可扩展。

每个分数计算类型都可以用于任何分数定义，如 HardSoftScore 或 HardMediumSoftScore。所有分数计算类型都面向对象，并可重复利用现有的 Java 代码。

重要

分数计算必须为只读。它不得以任何方式更改计划实体或问题事实。例如，分数计算不得对分数计算中的规划实体调用 setter 方法。

如果可以预测，在启用了 environmentMode 断言时，OptaPlanner 不会重新计算解决方案分数。例如，在完成获奖步骤后，无需计算分数，因为之前已完成并撤消。因此，无法保证在分数计算过程中应用的更改实际发生。

要在规划变量更改时更新计划实体，请使用 shadow 变量。

10.2.1. Implenting the Easy Java 分数计算类型

Easy Java 分数计算类型提供了一种在 Java 中实施分数计算的简单方法。您可以使用单一方法以 Java 或其他 JVM 语言实施所有限制。

- 优点：
  - 使用普通旧 Java，因此没有学习曲线
  - 提供一个将分数计算委托给现有代码库或旧系统的机会
- 缺点：
  - 最慢的计算类型
  - 不扩展，因为没有增量分数计算

流程

1. 实现 EasyScoreCalculator 接口：

```
public interface EasyScoreCalculator<Solution_, Score_ extends Score<Score_>> {
```

```
    Score_ calculateScore(Solution_ solution);

}
```

以下示例在 N Queens 问题中实施这个接口：

```
public class NQueensEasyScoreCalculator
    implements EasyScoreCalculator<NQueens, SimpleScore> {

    @Override
    public SimpleScore calculateScore(NQueens nQueens) {
        int n = nQueens.getN();
        List<Queen> queenList = nQueens.getQueenList();

        int score = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                Queen leftQueen = queenList.get(i);
                Queen rightQueen = queenList.get(j);
                if (leftQueen.getRow() != null && rightQueen.getRow() != null) {
                    if (leftQueen.getRowIndex() == rightQueen.getRowIndex()) {
                        score--;
                    }
                    if (leftQueen.getAscendingDiagonalIndex() ==
rightQueen.getAscendingDiagonalIndex()) {
                        score--;
                    }
                    if (leftQueen.getDescendingDiagonalIndex() ==
rightQueen.getDescendingDiagonalIndex()) {
                        score--;
                    }
                }
            }
        }
        return SimpleScore.valueOf(score);
    }

}
```

2. 在 solver 配置中配置 EasyScoreCalculator 类。以下示例演示了如何在 N Queens 问题中实施这个接口：

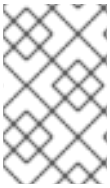
```
  <scoreDirectorFactory>

<easyScoreCalculatorClass>org.optaplanner.examples.nqueens.optional.score.NQueen
sEasyScoreCalculator</easyScoreCalculatorClass>
  </scoreDirectorFactory>
```

3. 要在 solver 配置中动态配置 EasyScoreCalculator 方法的值，以便基准器可以调整这些参数，添加 easyScoreCalculatorCustomProperties 元素并使用自定义属性：

```
<scoreDirectorFactory>
  <easyScoreCalculatorClass>...MyEasyScoreCalculator</easyScoreCalculatorClass>
  <easyScoreCalculatorCustomProperties>
    <property name="myCacheSize" value="1000" />
  </easyScoreCalculatorCustomProperties>
</scoreDirectorFactory>
```

10.2.2. 实施增量 Java 分数计算类型

Incremental Java 分数计算类型提供了一种方式，可以在 Java 中逐步实施分数计算。



注意

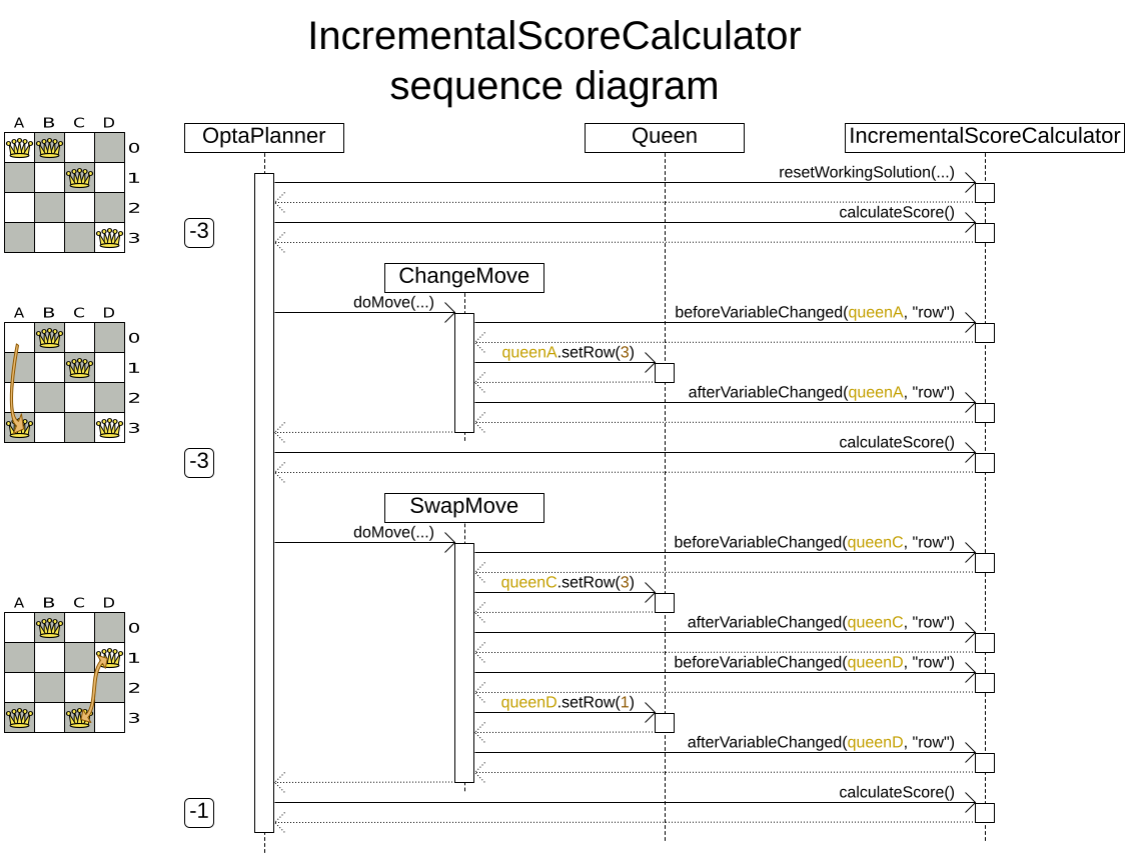
不建议使用这个类型。

- 优点：
  - 非常快速且可扩展的。如果正确实施，这是目前最快的类型。
- 缺点：
  - 很难写入。
    - 一个可扩展的实现，它大量使用映射、索引等。
    - 您必须自行了解、设计、编写并改进所有这些性能优化。
  - 读取困难。常规分数约束更改可能会导致高维护成本。

流程

1. 实现 IncrementalScoreCalculator 接口的所有方法：

```
public interface IncrementalScoreCalculator<Solution_, Score_ extends
Score<Score_>> {

    void resetWorkingSolution(Solution_ workingSolution);

    void beforeEntityAdded(Object entity);

    void afterEntityAdded(Object entity);

    void beforeVariableChanged(Object entity, String variableName);

    void afterVariableChanged(Object entity, String variableName);

    void beforeEntityRemoved(Object entity);

    void afterEntityRemoved(Object entity);

    Score_ calculateScore();

}
```



以下示例在 N Queens 问题中实施这个接口：

```
public class NQueensAdvancedIncrementalScoreCalculator
    implements IncrementalScoreCalculator<NQueens, SimpleScore> {

    private Map<Integer, List<Queen>> rowIndexMap;
```

```
    private Map<Integer, List<Queen>> ascendingDiagonalIndexMap;
    private Map<Integer, List<Queen>> descendingDiagonalIndexMap;

    private int score;

    public void resetWorkingSolution(NQueens nQueens) {
        int n = nQueens.getN();
        rowIndexMap = new HashMap<Integer, List<Queen>>(n);
        ascendingDiagonalIndexMap = new HashMap<Integer, List<Queen>>(n * 2);
        descendingDiagonalIndexMap = new HashMap<Integer, List<Queen>>(n * 2);
        for (int i = 0; i < n; i++) {
            rowIndexMap.put(i, new ArrayList<Queen>(n));
            ascendingDiagonalIndexMap.put(i, new ArrayList<Queen>(n));
            descendingDiagonalIndexMap.put(i, new ArrayList<Queen>(n));
            if (i != 0) {
                ascendingDiagonalIndexMap.put(n - 1 + i, new ArrayList<Queen>(n));
                descendingDiagonalIndexMap.put((-i), new ArrayList<Queen>(n));
            }
        }
        score = 0;
        for (Queen queen : nQueens.getQueenList()) {
            insert(queen);
        }
    }

    public void beforeEntityAdded(Object entity) {
        // Do nothing
    }

    public void afterEntityAdded(Object entity) {
        insert((Queen) entity);
    }

    public void beforeVariableChanged(Object entity, String variableName) {
        retract((Queen) entity);
    }

    public void afterVariableChanged(Object entity, String variableName) {
        insert((Queen) entity);
    }

    public void beforeEntityRemoved(Object entity) {
        retract((Queen) entity);
    }

    public void afterEntityRemoved(Object entity) {
        // Do nothing
    }

    private void insert(Queen queen) {
        Row row = queen.getRow();
        if (row != null) {
            int rowIndex = queen.getRowIndex();
            List<Queen> rowIndexList = rowIndexMap.get(rowIndex);
            score -= rowIndexList.size();
            rowIndexList.add(queen);
```

```
            List<Queen> ascendingDiagonalIndexList =
ascendingDiagonalIndexMap.get(queen.getAscendingDiagonalIndex());
            score -= ascendingDiagonalIndexList.size();
            ascendingDiagonalIndexList.add(queen);
            List<Queen> descendingDiagonalIndexList =
descendingDiagonalIndexMap.get(queen.getDescendingDiagonalIndex());
            score -= descendingDiagonalIndexList.size();
            descendingDiagonalIndexList.add(queen);
        }
    }

    private void retract(Queen queen) {
        Row row = queen.getRow();
        if (row != null) {
            List<Queen> rowIndexList = rowIndexMap.get(queen.getRowIndex());
            rowIndexList.remove(queen);
            score += rowIndexList.size();
            List<Queen> ascendingDiagonalIndexList =
ascendingDiagonalIndexMap.get(queen.getAscendingDiagonalIndex());
            ascendingDiagonalIndexList.remove(queen);
            score += ascendingDiagonalIndexList.size();
            List<Queen> descendingDiagonalIndexList =
descendingDiagonalIndexMap.get(queen.getDescendingDiagonalIndex());
            descendingDiagonalIndexList.remove(queen);
            score += descendingDiagonalIndexList.size();
        }
    }

    public SimpleScore calculateScore() {
        return SimpleScore.valueOf(score);
    }

}
```

2. 在 solver 配置中配置 incrementalScoreCalculatorClass 类。以下示例演示了如何在 N Queens 问题中实施这个接口：

```
  <scoreDirectorFactory>

<incrementalScoreCalculatorClass>org.optaplanner.examples.nqueens.optional.score.NQueensAdvancedIncrementalScoreCalculator</incrementalScoreCalculatorClass>
  </scoreDirectorFactory>
```

重要

编写和审核增量分数计算器代码可能比较困难。通过使用 EasyScoreCalculator 来满足 environmentMode 触发的断言，以保证其正确性。

3. 要在 solver 配置中动态配置 IncrementalScoreCalculator 的值，以便基准器可以调整这些

参数，添加 incrementalScoreCalculProperties 元素并使用自定义属性：

```
<scoreDirectorFactory>

<incrementalScoreCalculatorClass>...MyIncrementalScoreCalculator</incrementalScoreCalculatorClass>
  <incrementalScoreCalculatorCustomProperties>
    <property name="myCacheSize" value="1000"/>
  </incrementalScoreCalculatorCustomProperties>
</scoreDirectorFactory>
```

4. 可选：实现 ConstraintMatchAwareIncrementalScoreCalculator 接口，以促进以下目标：

- 使用 ScoreExplanation.getConstraintMatchTotalMap （） 对每个分数约束进行分割，以说明分数。
- 通过 ScoreExplanation.getIndictmentMap （） 中断来视觉化或排序规划实体。
- 如果在 FAST_ASSERT 或 FULL_ASSERT environmentMode 中损坏 IncrementalScoreCalculator，您会收到详细的分析。

```
public interface ConstraintMatchAwareIncrementalScoreCalculator<Solution_,
Score_ extends Score<Score_>> {

  void resetWorkingSolution(Solution_ workingSolution, boolean
constraintMatchEnabled);

  Collection<ConstraintMatchTotal<Score_>> getConstraintMatchTotals();

  Map<Object, Indictment<Score_>> getIndictmentMap();
}
```

例如，在机器重新分配中，为每个约束类型创建一个 ConstraintMatchTotal，并为每个约束匹配调用 addConstraintMatch （） ：

```
public class MachineReassignmentIncrementalScoreCalculator
    implements
ConstraintMatchAwareIncrementalScoreCalculator<MachineReassignment,
HardSoftLongScore> {
  ...

  @Override
  public void resetWorkingSolution(MachineReassignment workingSolution,
```

```
        boolean constraintMatchEnabled) {
            resetWorkingSolution(workingSolution);
            // ignore constraintMatchEnabled, it is always presumed enabled
        }

        @Override
        public Collection<ConstraintMatchTotal<HardSoftLongScore>>
    getConstraintMatchTotals() {
            ConstraintMatchTotal<HardSoftLongScore> maximumCapacityMatchTotal =
    new DefaultConstraintMatchTotal<>(CONSTRAINT_PACKAGE,
                "maximumCapacity", HardSoftLongScore.ZERO);
            ...
            for (MrMachineScorePart machineScorePart : machineScorePartMap.values())
    {
                for (MrMachineCapacityScorePart machineCapacityScorePart :
    machineScorePart.machineCapacityScorePartList) {
                    if (machineCapacityScorePart.maximumAvailable < 0L) {
                        maximumCapacityMatchTotal.addConstraintMatch(
                                Arrays.asList(machineCapacityScorePart.machineCapacity),

    HardSoftLongScore.valueOf(machineCapacityScorePart.maximumAvailable, 0));
                    }
                }
            }
            ...
            List<ConstraintMatchTotal<HardSoftLongScore>> constraintMatchTotalList =
    new ArrayList<>(4);
            constraintMatchTotalList.add(maximumCapacityMatchTotal);
            ...
            return constraintMatchTotalList;
        }

        @Override
        public Map<Object, Indictment<HardSoftLongScore>> getIndictmentMap() {
            return null; // Calculate it non-incrementally from getConstraintMatchTotals()
        }
    }
```

getConstraintMatchTotals （） 代码通常会复制 normal IncrementalScoreCalculator 方法的一些逻辑。约束流和 Drools Score Calculation 没有这种缺点，因为它们在不需要任何额外的域代码时会自动匹配。

第 11 章 INITIALIZINGSCORETREND 类

您可以将 InitializingScoreTrend 类添加到优化算法中，以指定在附加变量初始化时分数的变化，并且已初始化的变量不会改变。某些优化算法，如结构 Heuristics 和 Exhaustive Search，在这些信息可用时更快地运行。

您可以为分数或每个分数级别分别指定以下趋势之一：

- ANY （默认）：初始化额外变量可能会以正数或负面的方式改变分数。这种趋势不提供性能提升。
- ONLY_UP (rare)：初始化额外变量只能以正的形式更改分数。ONLY_UP 趋势需要以下条件：
  - 只有正限制。
  - 初始化下一个变量无法匹配之前初始化的变量匹配的正约束。
- ONLY_DOWN ：初始化额外的变量只能更改分数。ONLY_DOWN 需要以下条件：
  - 只有负限制。
  - 初始化下一个变量无法匹配之前初始化的变量匹配的负约束。

大多数用例只有负限制。许多用例都有仅停机的 InitializingScoreTrend 类，如下例所示：

```
<scoreDirectorFactory>
<constraintProviderClass>org.optaplanner.examples.cloudbalancing.score.CloudBalancingConstraintProvider</constraintProviderClass>
  <initializingScoreTrend>ONLY_DOWN</initializingScoreTrend>
</scoreDirectorFactory>
```

另外，您还可以单独指定每个分数级别的趋势，如下例所示：

```
  <scoreDirectorFactory>

<constraintProviderClass>org.optaplanner.examples.cloudbalancing.score.CloudBalancingCon
straintProvider</constraintProviderClass>
    <initializingScoreTrend>ONLY_DOWN/ONLY_DOWN</initializingScoreTrend>
  </scoreDirectorFactory>
```

如果您使用 environmentMode 类，并将值指定为 FULL_ASSERT 或 FAST_ASSERT，则环境模式会在增量分数计算中检测分数崩溃。

但是，这样做不会验证您的分数计算器是否实施了您的分数限制。例如，一个约束可能会与错误模式匹配。要根据独立实施验证约束，请配置 assertionScoreDirectorFactory 类：

```
<environmentMode>FAST_ASSERT</environmentMode>
...
<scoreDirectorFactory>

<constraintProviderClass>org.optaplanner.examples.nqueens.optional.score.NQueensConstraintProvider</constraintProviderClass>
  <assertionScoreDirectorFactory>

<easyScoreCalculatorClass>org.optaplanner.examples.nqueens.optional.score.NQueensEasyScoreCalculator</easyScoreCalculatorClass>
  </assertionScoreDirectorFactory>
</scoreDirectorFactory>
```

在本例中，NQueensConstraintProvider 实现由 EasyScoreCalculator 验证。

注意

这种技术非常适合隔离分数损坏，但为了验证约束实施了实际业务需求，通常最好使用 ConstraintVerifier 进行单元测试。

第 13 章 分数计算性能技巧

解决者的大部分执行时间都涉及运行分数计算，该计算在 solver 的深度循环中调用。更快的分数计算以相同的算法较少的时间返回相同的解决方案。这通常在同一时间内提供更好的解决方案。使用以下技术提高分数计算性能。

13.1. 分数计算速度

当您提高分数计算时，请关注最大化分数计算速度，而不是最大化最佳分数。分数计算的显著改进有时可能会有所改进，例如当算法处于本地或全局 optima 中时。如果您正在监视计算速度，则分数计算改进会更加可见。

每秒分数计算速度是可靠的分数计算性能，即使它受到非核心计算执行时间的影响。结果取决于问题数据集的问题扩展。通常，即使在大规模问题中，每秒分数计算速度高于 1000，除非您使用 EasyScoreCalculator 类。

通过观察计算速度，您可以删除或添加分数限制，并将最新的计算速度与原始计算速度进行比较。

注意

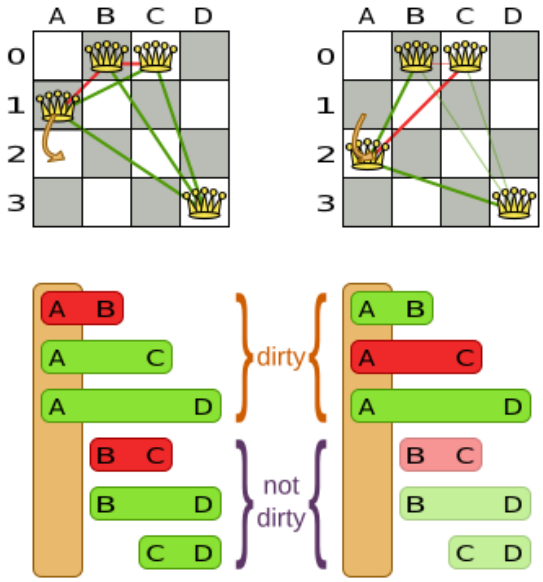
将最佳分数与原始最佳分数进行比较是无分。它与 apples 和 oranges 进行比较。

13.2. 增量分数计算

增量分数计算也称为基于 delta 的分数计算。当解决方案发生变化时，增量分数计算通过评估当前状态和之前状态之间的更改来获取新的分数，而不是在每次解决方案评估中重新计算整个分数。例如，在 N Queens 问题中，当 queen A 从行 1 移动到 2 行时，incrementalScoreCalculation 类不会检查 queen B 和 C 是否可以相互攻击，因为它们都更改了位置，如以下图所示：

Incremental score calculation

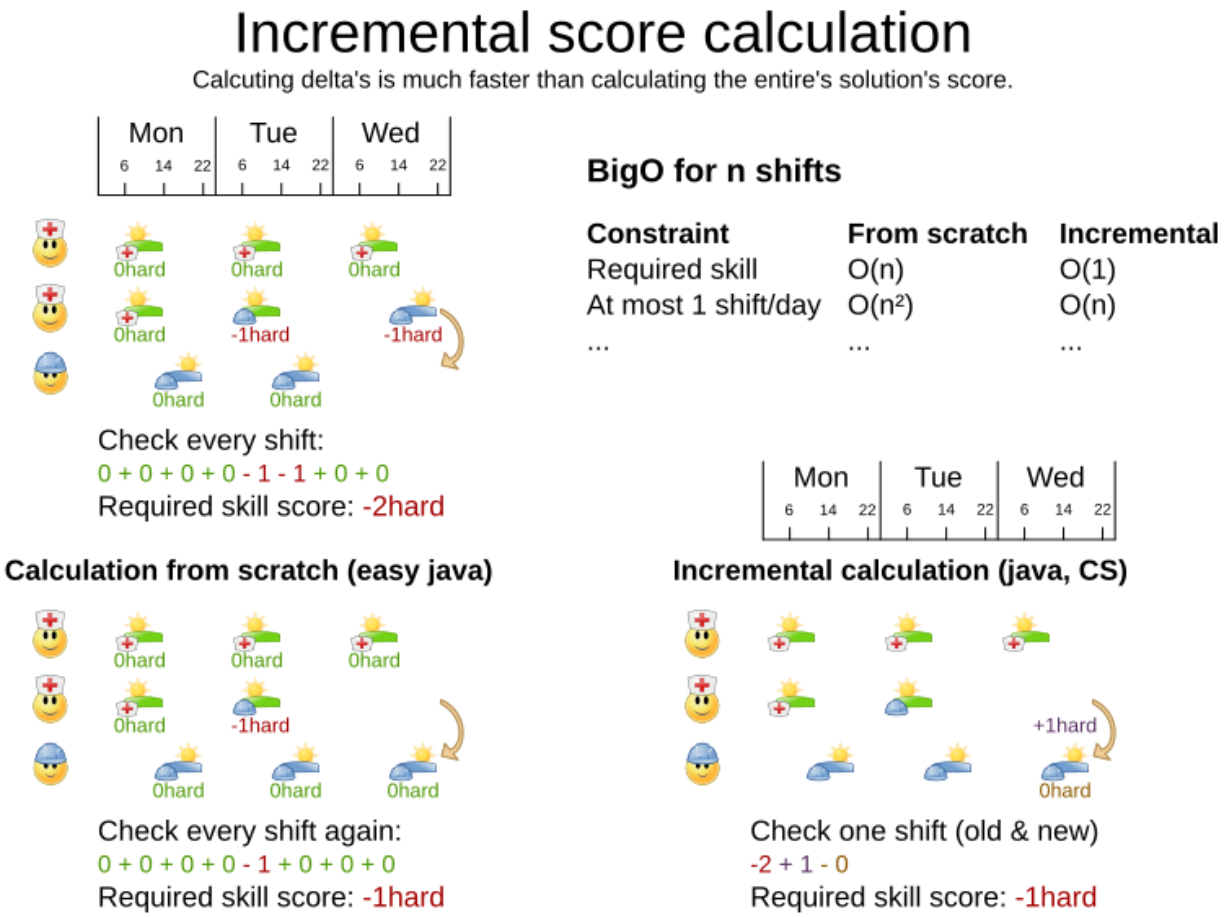
Incremental score calculation is much more scalable because only the delta is calculated.



The rule engine
(with forward chaining)
only recalculates dirty tuples.

queens	dirty		total	speedup
4	3	of	6	time / 2
8	7	of	28	time / 4
16	15	of	120	time / 8
32	31	of	496	time / 16
64	63	of	2016	time / 32
n	*n-1*	*of*	*n*(n-1)/2*	time / *(n/2)*

以下示例显示了员工购买的增量分数计算：



增量分数计算提供了显著的性能和可扩展性。约束流或 Drools 分数计算提供了此可扩展性收益，而不强制您编写复杂的增量分数计算算法。只需让引擎执行硬性工作。

请注意，计算速度的增长相对于您的规划问题的大小（您的 n）。这使得增量分数计算可扩展。

13.3. 远程服务

除非将 EasyScoreCalculator 类桥接到旧系统，否则不要在您的分数计算中调用远程服务。网络延迟会显著降低您的分数计算性能。如果可能，缓存这些远程服务的结果。

如果约束的某些部分可以被计算一次，当 solver 启动时，且永远不会在解决过程中改变，请将其转换为缓存的问题事实。

13.4. 无状态限制

如果您知道特定约束不能被破坏，或者它总是被破坏，请不要为其写入分数约束。例如，在 N Queens 问题中，分数计算不会检查多个 queens occupy 相同列，因为 queen 栏永远不会更改，并且每个解决

方案都从不同列上的每个 queen 开始。

注意

不要过度使用这种技术。如果某些数据集没有使用特定的约束，但其他数据，只要您可以立即退出约束。不需要根据数据集动态更改分数计算。

13.5. 内置硬性限制

硬约束有时可以构建在内，而不是实现硬约束。例如，在 school timetabling 示例中，如果 Lecture A 不应分配给 Room X，但它使用 Solution 上的 ValueRangeProvider 类，Solver 通常会尝试将其分配给 Room X，以发现它会破坏硬约束。在规划实体或过滤的选择中使用 ValueRangeProvider，以定义 Lecture A 应该只分配与 X 不同的 Room A。

这可在一些用例中获得良好的性能，不仅是因为分数计算速度更快，但大多数优化算法将花费较少的时间评估不良的解决方案。但是，这通常不是个好主意，因为交易短期的好处会带来长期的危害。

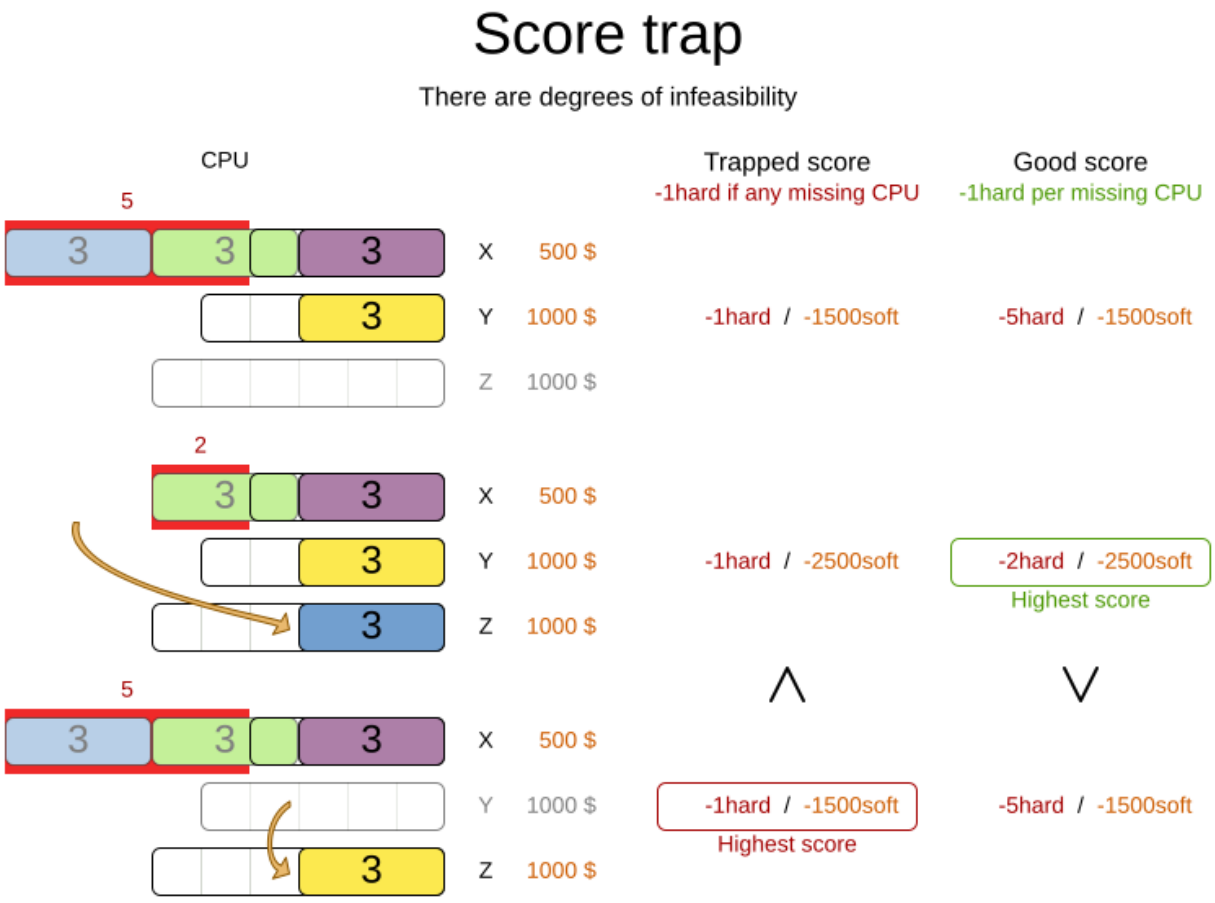
- 许多优化算法依赖于自由在更改计划实体时中断硬限制，而不是利用本地 optima。
- 这两种实施方法都有限制，如功能兼容性和禁用自动性能优化。

13.6. 分数陷阱

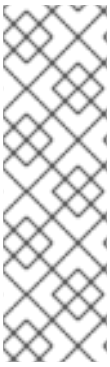
确保您的任何分数限制都不会导致分数陷阱。陷阱的 score 约束对多个约束匹配应用相同的权重。将这个组约束匹配在一起，并为该约束创建一个 flatlined score 功能。这可能导致解决方案状态，其中必须完成几项移动才能解决或降低该单一约束的权重。以下示例演示了分数陷阱：

- 每个操作表都需要两个文档，但您一次只移动一个文档。solver 没有激励地将文档组织移到没有文档人员的表中。要修复这个问题，请在分数函数中仅有一个 doctors 以上的表。
- 两个考试必须同时进行，但您一次只能移动一个考试。解决者必须在不移动同一迁移的情况下，将一项考试移至另一个时间插槽。要解决此问题，请添加一个粗粒度迁移，同时移动两个考试。

下图显示了分数陷阱：



如果蓝色项目从过载计算机移到空计算机，则硬分数应该提高。但是，陷阱得分的实施无法做到这一点。solver 最终应该会遇到这个陷阱，但会花费大量努力，特别是在过载计算机上有更多进程时。在这段操作之前，实际上可能开始将更多的进程移到该超载计算机中，因为这样做没有损失。



注意

避免分数陷阱并不意味着您的分数功能应该足够智能，以避免本地 optima。保留为优化算法以处理本地 optima。

避免分数陷阱意味着可以单独避免每个分数约束。

重要

始终指定责任程度。业务经常表示"如果解决方案可行，则这并不重要"。 虽然这对业务而言是如此，对分数计算而言并不重要，因为从了解解决方案不良的分数计算中受益。在实践中，软限制通常以自然方式执行此操作，而这只是出于硬限制而做的一个问题。

可以通过多种方式处理分数陷阱：

- 改进 score 约束，以区分分数权重。例如，为每个缺少的 CPU 而不是 -1hard （如果没有任何 CPU ）进行 penalize -1hard。
- 如果业务视角不允许更改分数约束，请添加一个较低分数，分数约束。例如，如果缺少任何 CPU，在每个缺少的 CPU 上 penalize -1 subsoft。业务会忽略子软分数级别。
- 使用现有的细粒度移动添加粗粒度移动和取消选择它们。粗粒度移动有效执行多次移动，通过单一移动直接获得分数陷阱。例如，将多个项从同一容器移动到另一个容器。

13.7. STEPLIMIT 基准

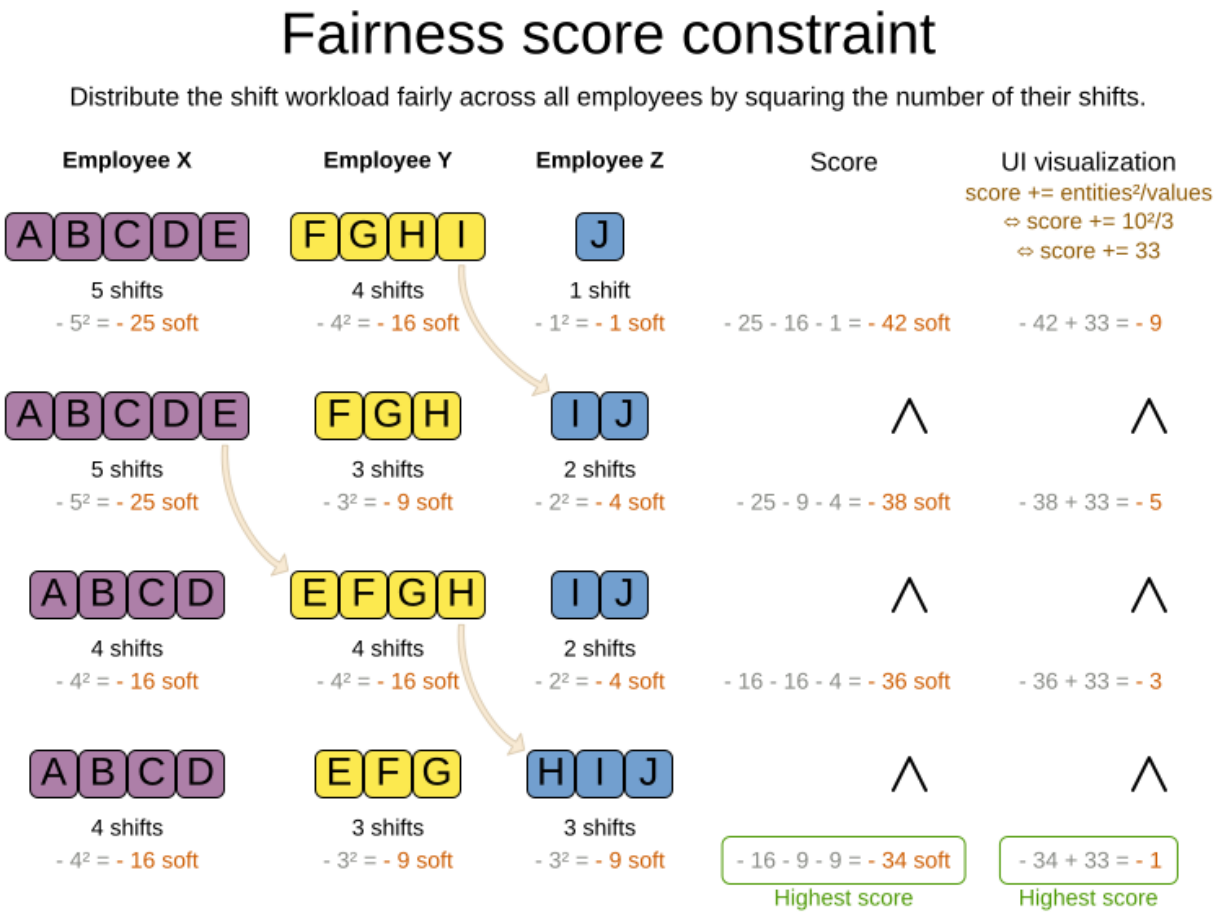
不是所有分数限制都有相同的性能成本。有时，一个分数约束可能会终止分数计算性能。使用基准器执行一分钟运行，然后在您注释掉所有分数限制但其中一个分数限制时，检查分数计算速度会怎样。

13.8. 平等分数限制

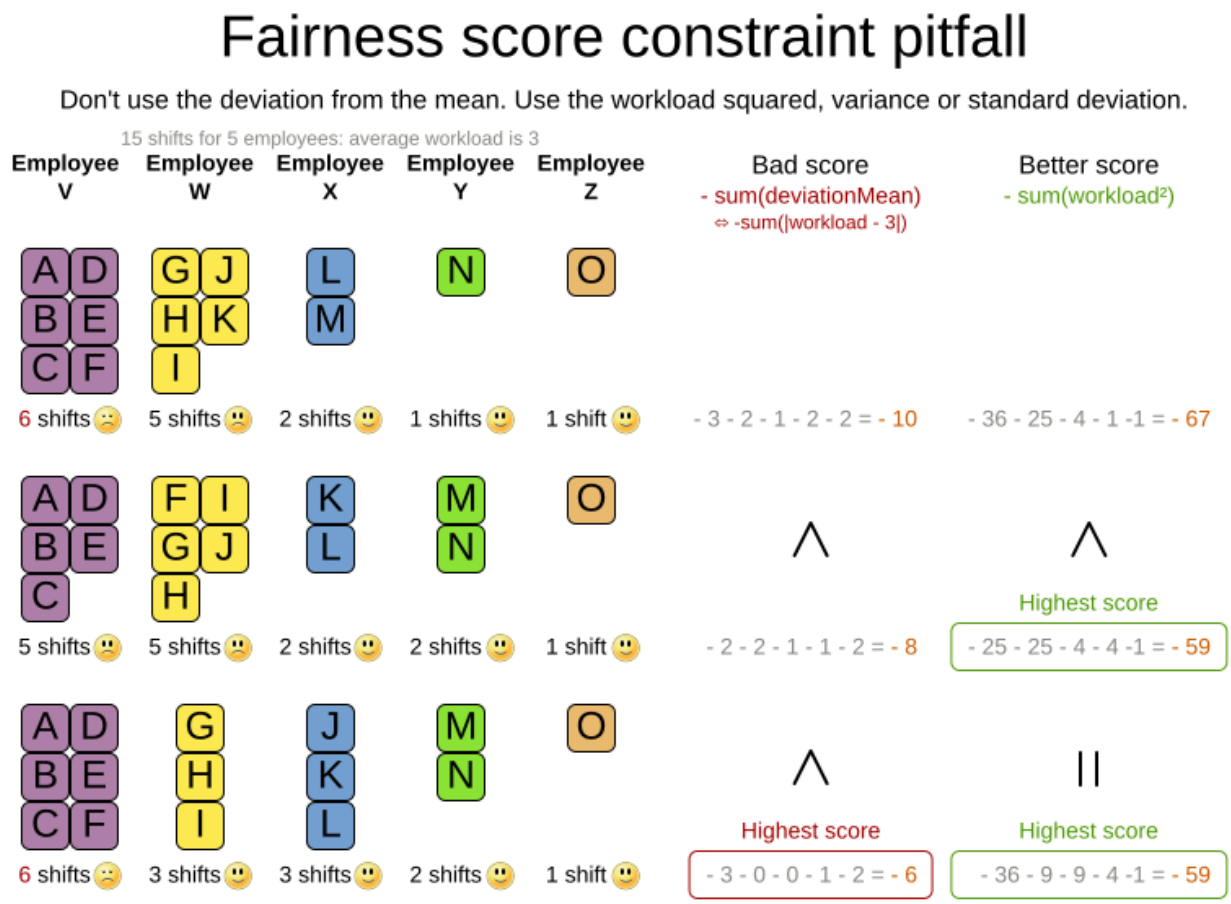
有些用例需要业务来提供公平调度，通常作为软分数约束，例如：

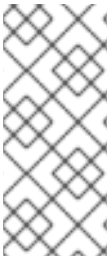
- 为避免环境，在员工中公平地分发工作负载。
- 为提高可靠性，请在资产间平均分配工作负载。

实施此类约束可能看似困难，特别是因为有不同方法来正式 化公平，但通常以最理想的方式实现方式工作负载 实施。对于每个员工或资产，将工作负载指定为 w，从分数中减去 w zFCP。



平方工作负载实施保证，如果您从指定解决方案中选择两个员工，并在这两个员工公平间分布，则生成的新解决方案将具有更好的总体分数。不要只使用与平均工作负载的不同，因为这可能会导致不公平，如下图所示：





注意

除了 方括号工作负载 实施外，也可以使用不同（平均差异差异）或标准 deviation (square root of the variance)。这对分数比较没有影响，因为计划期间平均不会改变。它只是实施更多工作，因为平均需要已知且简单慢，因为计算需要更长的时间。

当工作负载完全平衡时，用户通常希望看到 0 分数，而不是 distracting -34soft。这显示在前面的镜像中，其中几乎是完全均衡的最后一个解决方案。要为空，请在分数中添加平均乘以实体数，或者在 UI 中显示差异或标准偏差。

13.9. 其他分数计算性能提示

使用以下提示进一步提高分数计算性能：

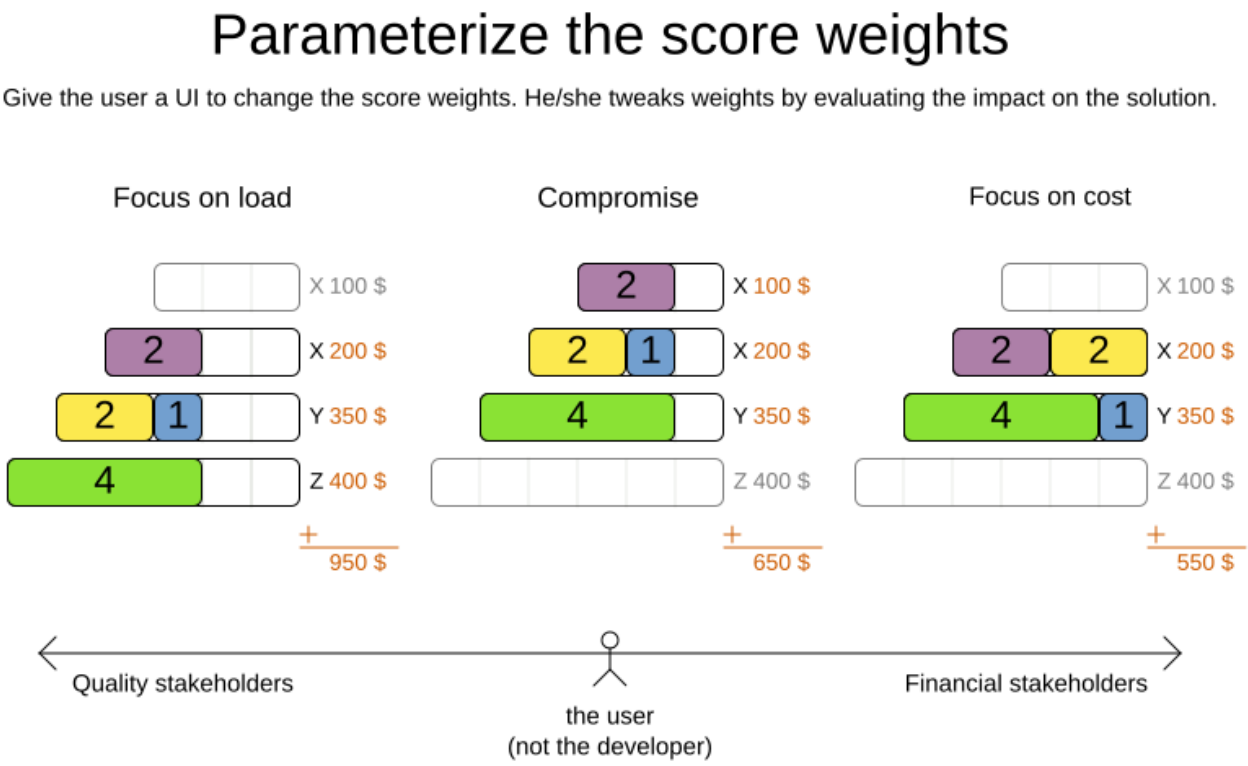
- *验证您的分数计算在正确的数字类型中。例如，如果您要添加类型为 int 的值，请不要存储类型 double （计算）的结果。*
-

为获得最佳性能，请使用最新的 Java 版本。例如，您可以通过从 Java 11 切换到 17 来提高大约 10 % 的性能。

- 始终记住，预先考虑的优化是非常不可靠的。确保您的设计足够灵活，以便进行基于配置的调整。

13.10. 配置限制

决定每个约束的正确权重和级别并不容易。它通常涉及不同的拥有者及其优先级。此外，量化软约束的影响通常是业务经理的新体验，因此他们还需要大量的迭代才能获得它。为了更容易做到这一点，请使用 @ConstraintConfiguration 类及约束 weights 和参数。然后，提供 UI，以便业务管理器可以调整约束权重，并视觉化生成的解决方案，如下图所示：



例如，在会议调度问题中，最小暂停约束具有约束权重，但也有一个 constraint 参数，用于定义同一 speaker 的两个对话之间的时间长度。暂停长度取决于会议：在某些大会议中，20 分钟没有足够时间来从一个房间进入一个空间，在小的会议 10 分钟内可以有足够的时间。暂停长度是约束配置中的一个字段，没有 @ConstraintWeight 注释。

每个约束都有一个约束软件包和约束名称，它们组成约束 ID。约束 ID 将约束权重与约束实现连接。对

于每个约束 weight，必须使用相同软件包和名称相同的约束实现。

- @ConstraintConfiguration 注释具有一个 constraintPackage 属性，默认为约束配置类的软件包。带有约束流的情况通常不需要指定它。

- @ConstraintWeight 注释具有一个 值，即约束名称（例如"REGION 冲突"）。它继承了来自 @ConstraintConfiguration 的 constraint 软件包，但它可以覆盖它，例如 @ConstraintWeight (constraintPackage = "...region.france", ...) 使用不同于其他权重的约束软件包。

因此，每个约束权重都以约束软件包和约束名称结束。每个约束权重链接带有约束实现，例如在约束流中：

```
public final class ConferenceSchedulingConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory factory) {
        return new Constraint[] {
                speakerConflict(factory),
                themeTrackConflict(factory),
                contentConflict(factory),
                ...
        };
    }

    protected Constraint speakerConflict(ConstraintFactory factory) {
        return factory.forEachUniquePair(...)
                ...
                .penalizeConfigurable("Speaker conflict", ...);
    }

    protected Constraint themeTrackConflict(ConstraintFactory factory) {
        return factory.forEachUniquePair(...)
                ...
                .penalizeConfigurable("Theme track conflict", ...);
    }

    protected Constraint contentConflict(ConstraintFactory factory) {
        return factory.forEachUniquePair(...)
                ...
                .penalizeConfigurable("Content conflict", ...);
    }

    ...

}
```

每个约束 weights 定义其约束的分数级别和分数权重。约束实现调用 rewardConfigurable （） 或

penalizeConfigurable （），约束权重会自动应用。

如果约束实现提供了匹配的权重，则匹配 weight 乘以约束权重。例如，内容冲突 约束权重默认为 100soft，约束实现根据共享内容标签的数量和两个通信的重叠持续时间来分配每个匹配项：

```
@ConstraintWeight("Content conflict")
private HardMediumSoftScore contentConflict = HardMediumSoftScore.ofSoft(100);
```

```
Constraint contentConflict(ConstraintFactory factory) {
    return factory.forEachUniquePair(Talk.class,
        overlapping(t -> t.getTimeslot().getStartDateTime(),
            t -> t.getTimeslot().getEndDateTime()),
        filtering((talk1, talk2) -> talk1.overlappingContentCount(talk2) > 0))
        .penalizeConfigurable("Content conflict",
                (talk1, talk2) -> talk1.overlappingContentCount(talk2)
                        * talk1.overlappingDurationInMinutes(talk2));
}
```

因此，当 2 重叠仅与 1 个内容标签进行交流，并且与 60 分钟重叠时，分数会受到 -6000soft 的影响。但是，当 2 重叠与共享 3 内容标签通信时，匹配权重为 180，因此分数受到 -18000soft 的影响。

流程

1. *创建一个新类来存放约束 weight 和其他约束参数，例如 Conversation ConstraintConfiguration。*

2. *使用 @ConstraintConfiguration 注解此类：*

```
@ConstraintConfiguration
public class ConferenceConstraintConfiguration {
    ...
}
```

3. *在规划解决方案中添加约束配置，并使用 @ConstraintConfigurationProvider 注解该字段或属性：*

```
@PlanningSolution
public class ConferenceSolution {

    @ConstraintConfigurationProvider
    private ConferenceConstraintConfiguration constraintConfiguration;

    ...
}
```

4. 在约束配置类中，为每个约束添加一个 @ConstraintWeight 属性，并为每个约束赋予一个默认值：

```
@ConstraintConfiguration(constraintPackage = "...conferencescheduling.score")
public class ConferenceConstraintConfiguration {

    @ConstraintWeight("Speaker conflict")
    private HardMediumSoftScore speakerConflict = HardMediumSoftScore.ofHard(10);

    @ConstraintWeight("Theme track conflict")
    private HardMediumSoftScore themeTrackConflict =
HardMediumSoftScore.ofSoft(10);
    @ConstraintWeight("Content conflict")
    private HardMediumSoftScore contentConflict = HardMediumSoftScore.ofSoft(100);

    ...
}
```

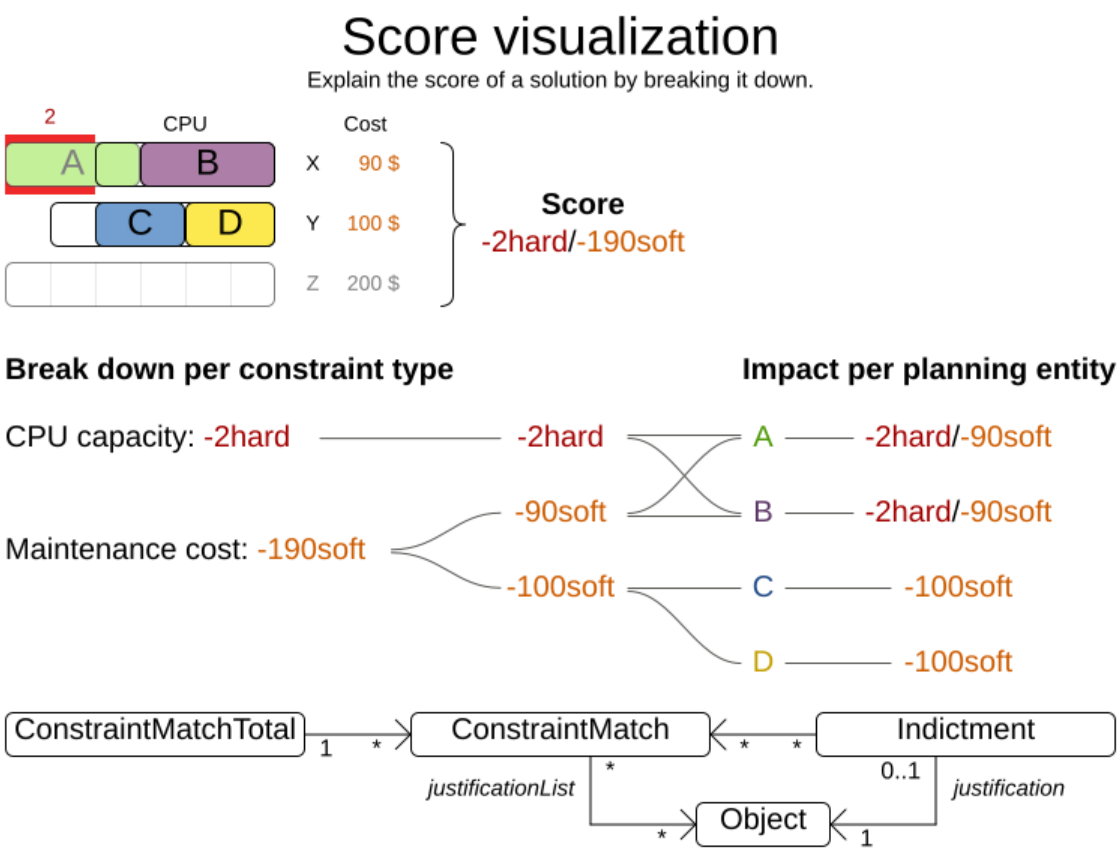
@ConstraintConfigurationProvider 注释会自动将约束配置作为问题事实公开。不需要添加 @ProblemFactProperty 注释。约束权重不能为空。

5. 在 UI 中公开约束权重，以便业务用户可以调整值。前面的示例使用 ofHard （）、$Medium （） 和 Soft （） 方法进行此操作。请注意，它如何 将内容冲突 约束默认设置为十倍，而不是主题跟踪冲突 约束。通常，约束权重只使用一个分数级别，但可以使用多个分数级别（以较小的性能成本）。

13.11. 解释分数

有几种方法显示了 OptaPlanner 分数是如何派生的。这称为解释分数：

- 打印 getSummary （） 的返回值。这是在开发期间解释分数的最简单方法，但仅将此方法用于诊断目的。
- 在应用程序或 Web UI 中使用 ScoreManager API。
- 划分每个约束的分数，以获得更精细的视图。



流程

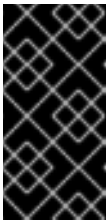
- *使用以下方法之一解释分数：*
  - *打印 getSummary （） 的返回值：*

```
System.out.println(scoreManager.getSummary(solution));
```

以下会议调度示例显示，与 S51 通信负责破坏硬约束：

```
Explanation of score (-1hard/-806soft):
    Constraint match totals:
        -1hard: constraint (Speaker required room tag) has 1 matches:
            -1hard: justifications ([S51])
        -340soft: constraint (Theme track conflict) has 32 matches:
            -20soft: justifications ([S68, S66])
            -20soft: justifications ([S61, S44])
            ...
        ...
    Indictments (top 5 of 72):
        -1hard/-22soft: justification (S51) has 12 matches:
            -1hard: constraint (Speaker required room tag)
```

```
        -10soft: constraint (Theme track conflict)
        ...
    ...
```



重要

不要尝试解析此字符串，或者在 UI 或公开的服务中使用它。反之，使用 ConstraintMatch API。

- *在应用程序或 Web UI 中使用 ScoreManager API。*

  a. *输入类似以下示例的代码：*

```
ScoreManager<CloudBalance, HardSoftScore> scoreManager =
ScoreManager.create(solverFactory);
ScoreExplanation<CloudBalance, HardSoftScore> scoreExplanation =
scoreManager.explainScore(cloudBalance);
```

  b. *当您需要计算解决方案分数时使用此代码：*

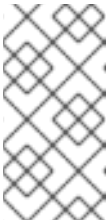
```
HardSoftScore score = scoreExplanation.getScore();
```

- *按约束划分分数：*

  a. *从 ScoreExplanation 中获取 ConstraintMatchTotal 值：*

```
Collection<ConstraintMatchTotal<HardSoftScore>> constraintMatchTotals =
scoreExplanation.getConstraintMatchTotalMap().values();
for (ConstraintMatchTotal<HardSoftScore> constraintMatchTotal :
constraintMatchTotals) {
    String constraintName = constraintMatchTotal.getConstraintName();
    // The score impact of that constraint
    HardSoftScore totalScore = constraintMatchTotal.getScore();

    for (ConstraintMatch<HardSoftScore> constraintMatch :
constraintMatchTotal.getConstraintMatchSet()) {
        List<Object> justificationList = constraintMatch.getJustificationList();
        HardSoftScore score = constraintMatch.getScore();
        ...
    }
}
```

每个 ConstraintMatchTotal 代表一个约束，它属于整体分数。所有 ConstraintMatchTotal.getScore （） 的总和等于整个分数。



注意

约束流和 Drools 分数计算支持约束自动匹配，但增量 Java 分数计算需要实施额外的接口。

13.12. 视觉化热计划实体

在 UI 中显示 heat 映射，它突出显示了对分数影响的规划实体和问题事实。

流程

- 从 ScoreExplanation 中获取 Indictment 映射：

```
Map<Object, Indictment<HardSoftScore>> indictmentMap =
scoreExplanation.getIndictmentMap();
for (CloudProcess process : cloudBalance.getProcessList()) {
   Indictment<HardSoftScore> indictment = indictmentMap.get(process);
   if (indictment == null) {
      continue;
   }
   // The score impact of that planning entity
   HardSoftScore totalScore = indictment.getScore();

   for (ConstraintMatch<HardSoftScore> constraintMatch :
indictment.getConstraintMatchSet()) {
      String constraintName = constraintMatch.getConstraintName();
      HardSoftScore score = constraintMatch.getScore();
      ...
   }
}
```

每个字典 都是涉及合理对象的所有限制总和。所有 Indictment.getScoreTotal （） 的总和与整体分数不同，因为多个字典实体可以共享相同的 ConstraintMatch。

注意

约束流和 Drools 分数计算支持约束自动匹配，但增量 Java 分数计算需要实施额外的接口。

13.13. 分数限制测试

不同的分数计算类型有不同的测试工具。单独为每个分数约束编写单元测试，以检查它的行为是否正确。

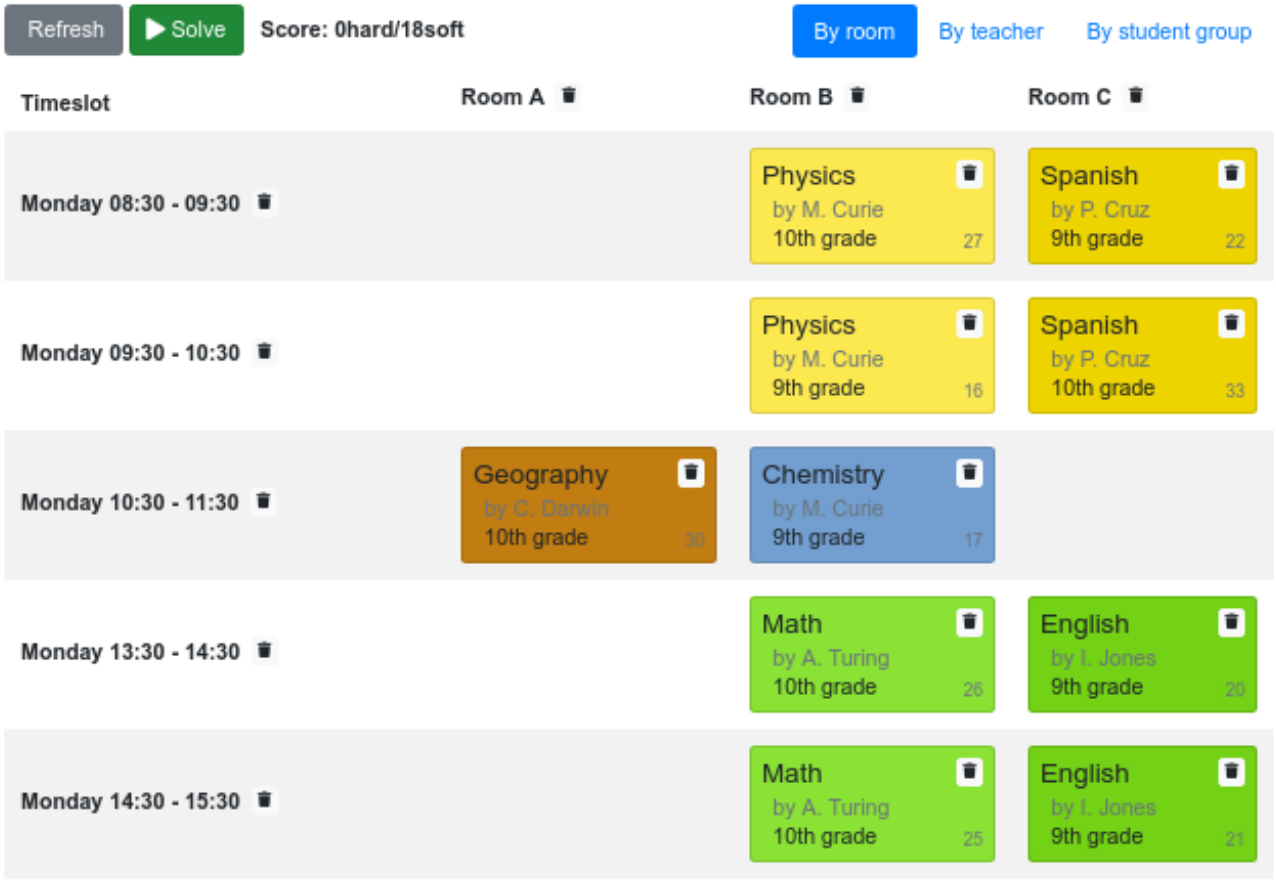
部分 V. 红帽构建的 OPTAPLANNER 快速启动指南

红帽构建的 OptaPlanner 提供了以下快速入门指南，以演示如何将 OptaPlanner 与不同的 techologies 集成：

- Red Hat build of Quarkus 平台的 Red Hat build of OptaPlanner: 一个 medium timetable 的快速开始指南
- Red Hat build of Quarkus 平台的 Red Hat build of OptaPlanner: vaccination appointment 调度程序快速启动指南
- 红帽构建的 Quarkus 平台上的 OptaPlanner 构建：员工调度程序快速启动指南
- Red Hat build of OptaPlanner on Spring Boot: a tutorial timeable quick start 指南
- 红帽构建的带有 Java solvers 的 OptaPlanner：一个中型快速开始指南

第 14 章 RED HAT BUILD OF QUARKUS 平台的 RED HAT BUILD OF OPTAPLANNER: 一个 MEDIUM TIMETABLE 的快速开始指南

本指南指导您使用 Red Hat Build of OptaPlanner 约束创建红帽构建的 Quarkus 应用程序来帮助您解决人工智能(AI)。您将构建一个 REST 应用程序，为教师和教师优化一个生存时间



您的服务将使用 AI 自动将 lesson 实例分配给 Timeslot 和 Room 实例，以遵循以下硬和软 调度限制：

- 房间最多可以有一门。
- 教员可以在大多数课程同时进行指导。
- 学员最多可同时参加一门课程。
- 教员更喜欢在单一房间发言。

- 教师希望检测后续课程和课程之间的差别。

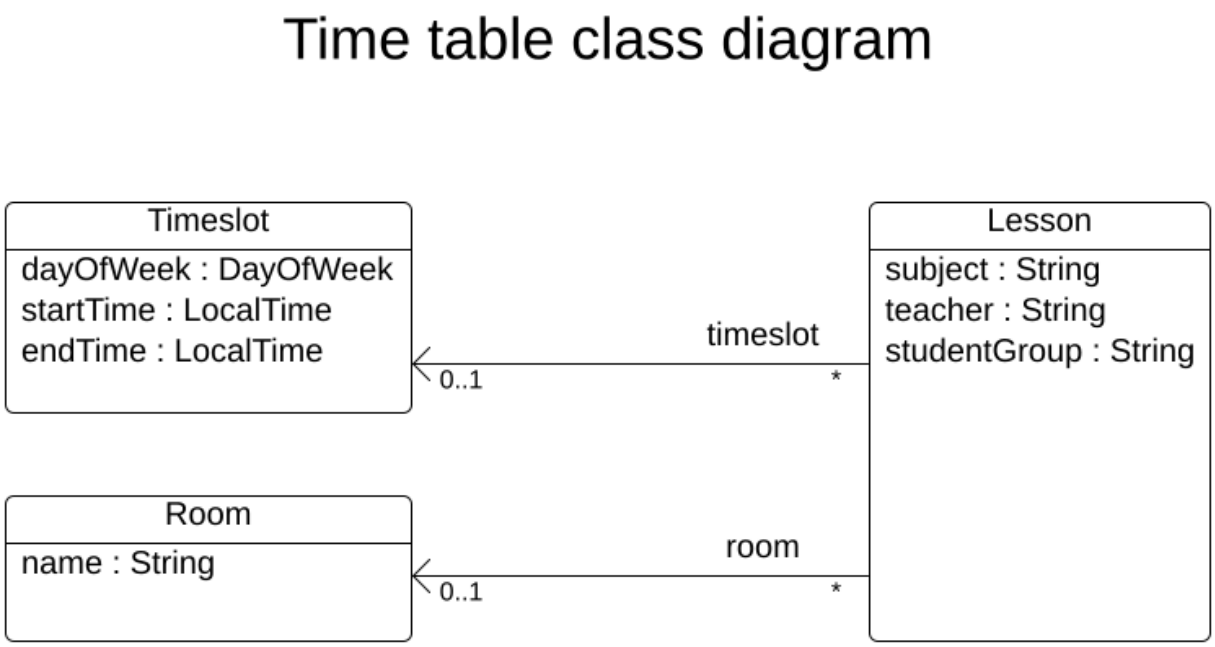
以数学方式说，中型时间稳定是一个 NP-hard 的问题。这意味着扩展比较困难。简单地迭代所有带有 brute 强制的可能组合，即使是超级计算机，则需要数百万年时间。SriovIBNetwork， AI 约束解决者（如红帽构建的 OptaPlanner）具有在合理的时间内提供接近最佳解决方案的高级算法。认为是合理的时间，这取决于您的问题的目标。

先决条件

- 已安装了 OpenJDK 11 或更高版本。红帽构建的 Open JDK 可通过红帽客户门户网站中的 Software Downloads 页面（需要登录）。
- Apache Maven 3.8 或更高版本已安装。Maven 位于 Apache Maven Project 网站。
- 提供了 IDE，如 IntelliJ IDEA、VSCode 或 Eclipse。
- 提供了 Red Hat build of OptaPlanner Red Hat build of Quarkus 项目。有关创建 Red Hat build of OptaPlanner Red Hat build of Quarkus 项目的说明，请参阅" 开始使用 OptaPlanner"部分的 OptaPlanner 部分。

14.1. 对域对象建模

红帽构建的 OptaPlanner timetable 项目的目标是为每个课程分配时间插槽和房间。要做到这一点，请添加三个类，Timeslot , lesson , 和 Room，如下图所示：



timeslot

***Timeslot** 类代表了在课程时的间隔，例如 Monday 10:30 - 11:30 或 Tuesday 13:30 - 14:30。在这个示例中，所有时间插槽都有相同的持续时间，在 lunch 或其他间没有时间插槽。*

时间段没有日期，因为高校计划每周都会重复。不需要 持续规划。一个 timeslot 被称为 问题，因为在解决过程中没有 Timeslot 实例改变。此类类不需要任何 OptaPlanner 特定注解。

房间

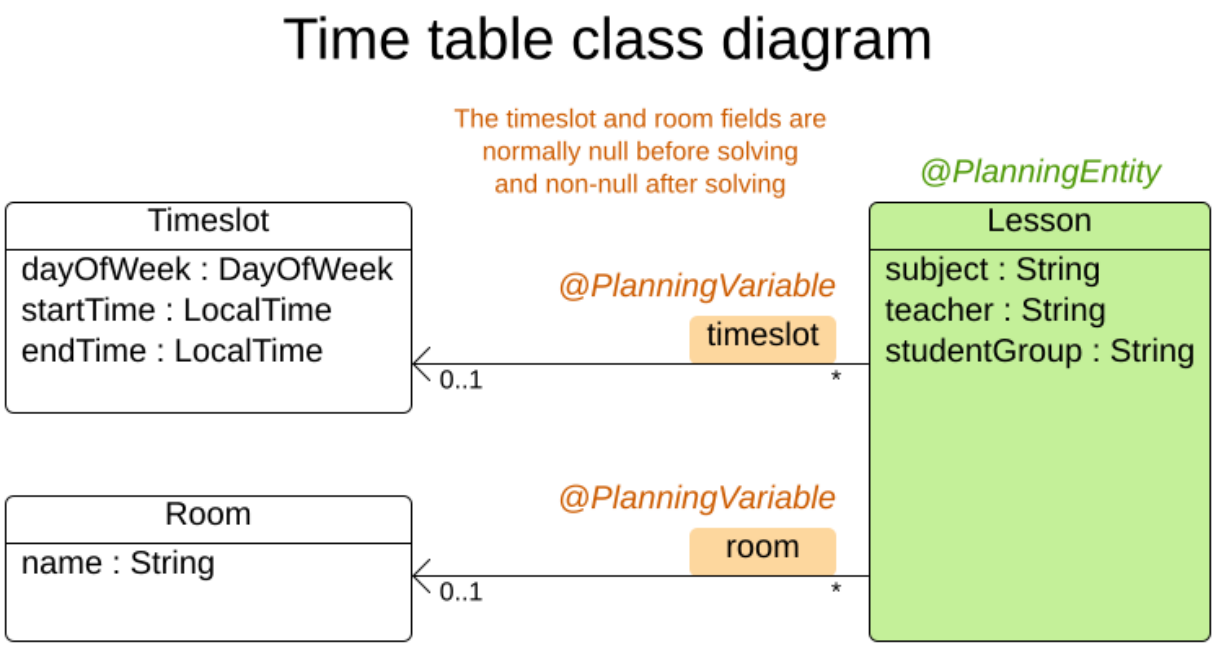
***Room** 类代表一个跟踪课程的位置，例如 Room A 或 Room B。在这个示例中，所有房间都没有容量限制，它们可以容纳所有课程。*

房间 实例在解决时 不会改变，因此也 是一个问题。

lesson

在短时间内，由 Lesson 类表示，教员向一组学员提供主题，例如 Math by A.Turing for 9th grade 或 Chemistry by M.Curie for 10th grade。如果每周由同一生组多次学习一个主题，则有多个仅通过 id 区分的 lesson 实例。例如，第 9 个评级每周有 6 个数个。

在解决期间，OptaPlanner 更改了 Lesson 类的 timeslot 和 room 字段，为每个课程分配时间插槽和房间。因为 OptaPlanner 更改这些字段，所以 lesson 是一个 规划实体 ：



上图中的大部分字段包含输入数据，但 orange 字段除外。一个 lesson 的 timeslot 和 room 字段在输入数据中未分配（空），并在输出数据中分配（非 null）。OptaPlanner 在解决过程中更改这些字段。此类字段称为计划变量。为了让 OptaPlanner 识别它们，timeslot 和 room 字段都需要 @PlanningVariable 注释。它包含类 lesson 需要一个 @PlanningEntity 注释。

流程

1. *创建 src/main/java/com/example/domain/Timeslot.java 类：*

```
package com.example.domain;

import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;

    private Timeslot() {
    }

    public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
        this.dayOfWeek = dayOfWeek;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    @Override
```

```
    public String toString() {
        return dayOfWeek + " " + startTime.toString();
    }

    // ********************************
    // Getters and setters
    // ********************************

    public DayOfWeek getDayOfWeek() {
        return dayOfWeek;
    }

    public LocalTime getStartTime() {
        return startTime;
    }

    public LocalTime getEndTime() {
        return endTime;
    }

}
```

注意 toString （） 方法保持输出短，以便更轻松地阅读 OptaPlanner 的 DEBUG 或 TRACE 日志，如稍后所示。

2. 创建 src/main/java/com/example/domain/Room.java 类：

```
package com.example.domain;

public class Room {

    private String name;

    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public String getName() {
        return name;
```

```
    }

}
```

3. 创建 src/main/java/com/example/domain/Lesson.java 类：

```
package com.example.domain;

import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
    }

    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // ********************************
    // Getters and setters
    // ********************************

    public Long getId() {
        return id;
    }

    public String getSubject() {
        return subject;
    }

    public String getTeacher() {
```

```
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {
        return timeslot;
    }

    public void setTimeslot(Timeslot timeslot) {
        this.timeslot = timeslot;
    }

    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
        this.room = room;
    }

}
```

Lesson 类具有一个 @PlanningEntity 注释，因此 OptaPlanner 知道此类在解决过程中发生了变化，因为它包含一个或多个计划变量。

timeslot 字段具有一个 @PlanningVariable 注释，因此 OptaPlanner 知道它可以更改其值。为了找到潜在的 Timeslot 实例来分配给此字段，OptaPlanner 使用 valueRangeProviderRefs 属性连接到提供 List<Timeslot> 的值范围供应商。有关值范围供应商的信息，请参阅 第 14.3 节 “在规划解决方案中收集域对象”。

room 字段还具有 @PlanningVariable 注释，原因相同。

14.2. 定义限制并计算分数

在解决问题时，分数 代表特定解决方案的质量。分数越高。红帽构建的 OptaPlanner 寻求最佳解决方案，这是在可用时间内获得最高分数的解决方案。这可能是 最佳解决方案。

因为 timetable 示例用例具有硬和软限制，所以请使用 HardSoftScore 类来代表分数：

- 硬约束不能被破坏。例如 ：一个空间最多可以同时有一个。

- 软限制不应中断。例如 ：教员更倾向于在单一房间进行学习。

硬约束会根据其他硬限制来加权。软限制是针对其他软限制的权重。硬限制始终超过软约束，无论其对应的权重是什么。

要计算分数，您可以实施 EasyScoreCalculator 类：

```
public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

    @Override
    public HardSoftScore calculateScore(TimeTable timeTable) {
        List<Lesson> lessonList = timeTable.getLessonList();
        int hardScore = 0;
        for (Lesson a : lessonList) {
            for (Lesson b : lessonList) {
                if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
                        && a.getId() < b.getId()) {
                    // A room can accommodate at most one lesson at the same time.
                    if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                        hardScore--;
                    }
                    // A teacher can teach at most one lesson at the same time.
                    if (a.getTeacher().equals(b.getTeacher())) {
                        hardScore--;
                    }
                    // A student can attend at most one lesson at the same time.
                    if (a.getStudentGroup().equals(b.getStudentGroup())) {
                        hardScore--;
                    }
                }
            }
        }
        int softScore = 0;
        // Soft constraints are only implemented in the "complete" implementation
        return HardSoftScore.of(hardScore, softScore);
    }

}
```

不幸的是，这个解决方案无法很好地扩展，因为它并没有递增：每次将课程分配给不同的时间插槽或房间，所有课程都会被重新评估以计算新的分数。

更好的解决方法是创建一个 src/main/java/com/example/solver/TimeTableConstraintProvider.java 类，以执行增量分数计算。此类使用 OptaPlanner 的 ConstraintStream API，该 API 由 Java 8 Streams 和 SQL 推进。ConstraintProvider 比 EasyScoreCalculator 增加了一个比 EasyScoreCalculator 更强的 magnitude 顺序：O(n)而不是 O(nbusybox)。

流程

创建以下 src/main/java/com/example/solver/TimeTableConstraintProvider.java 类：

```
package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
                // Hard constraints
                roomConflict(constraintFactory),
                teacherConflict(constraintFactory),
                studentGroupConflict(constraintFactory),
                // Soft constraints are only implemented in the "complete" implementation
        };
    }

    private Constraint roomConflict(ConstraintFactory constraintFactory) {
        // A room can accommodate at most one lesson at the same time.

        // Select a lesson ...
        return constraintFactory.forEach(Lesson.class)
                // ... and pair it with another lesson ...
                .join(Lesson.class,
                        // ... in the same timeslot ...
                        Joiners.equal(Lesson::getTimeslot),
                        // ... in the same room ...
                        Joiners.equal(Lesson::getRoom),
                        // ... and the pair is unique (different id, no reverse pairs)
                        Joiners.lessThan(Lesson::getId))
                // then penalize each pair with a hard weight.
                .penalize(HardSoftScore.ONE_HARD)
                .asConstraint("Room conflict");
    }

    private Constraint teacherConflict(ConstraintFactory constraintFactory) {
        // A teacher can teach at most one lesson at the same time.
        return constraintFactory.forEach(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getTeacher),
                        Joiners.lessThan(Lesson::getId))
                        .penalize(HardSoftScore.ONE_HARD)
                        .asConstraint("Teacher conflict");
    }
```

```
    private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
        // A student can attend at most one lesson at the same time.
        return constraintFactory.forEach(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getStudentGroup),
                        Joiners.lessThan(Lesson::getId))
                        .penalize(HardSoftScore.ONE_HARD)
                        .asConstraint("Student group conflict");
    }

}
```

14.3. 在规划解决方案中收集域对象

TimeTable 实例会打包单个 数据集 的所有 TimeTable、Room 和 lesson 实例。另外，因为它包含所有课程，每个课程都有特定的规划变量状态，所以它是一个 规划解决方案，它分数如下：

- 如果尚未分配课程，则它是一个 未初始化的 解决方案，例如，分数为 -4init/0hard/0soft 的解决方案。
- 如果它中断了硬限制，则它是一个 不可避免的 解决方案，例如分数为 -2hard/-3soft 的解决方案。
- 如果它遵循所有硬限制，则它是一个可行的解决方案，例如分数为 0hard/-7soft 的解决方案。

TimeTable 类具有 @PlanningSolution 注解，因此红帽构建的 OptaPlanner 知道此类包含所有输入和输出数据。

特别是，这个类是问题的输入：

- 包含所有时间插槽的 timeslotList 字段
  - 这是问题事实列表，因为它们在解决过程中不会改变。
- 带有所有房间的 roomList 字段

- 这是问题事实列表，因为它们在解决过程中不会改变。

- 具有所有课程的 lessonList 字段
  - 这是规划实体列表，因为它们在解决过程中发生了变化。
  - 每个 lesson：
    - timeslot 和 room 字段的值通常仍然为 null，因此未分配。它们是规划变量。
    - 其他字段（如 主题、sonr 和 studentGroup ）已填写。这些字段是问题属性。

但是，这个类也是解决方案的输出：

- 在解决解决后，每个 Lesson 实例都有非null 时间slot 和 room 字段的 lessonList 字段
- 代表输出解决方案的质量的 score 字段，例如 0hard/-5soft

流程

创建 src/main/java/com/example/domain/TimeTable.java 类：

```
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;

@PlanningSolution
public class TimeTable {

   @ValueRangeProvider(id = "timeslotRange")
```

```
    @ProblemFactCollectionProperty
    private List<Timeslot> timeslotList;

    @ValueRangeProvider(id = "roomRange")
    @ProblemFactCollectionProperty
    private List<Room> roomList;

    @PlanningEntityCollectionProperty
    private List<Lesson> lessonList;

    @PlanningScore
    private HardSoftScore score;

    private TimeTable() {
    }

    public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
            List<Lesson> lessonList) {
        this.timeslotList = timeslotList;
        this.roomList = roomList;
        this.lessonList = lessonList;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public List<Timeslot> getTimeslotList() {
        return timeslotList;
    }

    public List<Room> getRoomList() {
        return roomList;
    }

    public List<Lesson> getLessonList() {
        return lessonList;
    }

    public HardSoftScore getScore() {
        return score;
    }

}
```

值范围供应商

timeslotList 字段是一个值范围供应商。它保存 OptaPlanner 可以从中选择的 Timeslot 实例，以分配给 lesson 实例的 timeslot 字段。timeslotList 字段具有一个 @ValueRangeProvider 注释，用于将 id 与 lesson 中 @PlanningVariable 的 @PlanningVariable 的 id 匹配。

遵循相同的逻辑后，roomList 字段也具有 @ValueRangeProvider 注释。

问题事实和规划实体属性

此外，OptaPlanner 需要知道它可以更改哪些更少的实例，以及如何检索用于您的 TimeTableConstraintProvider 分数计算的 Timeslot 和 Room 实例。

timeslotList 和 roomList 字段具有一个 @ProblemFactCollectionProperty 注释，因此您的 TimeTableConstraintProvider 可以从这些实例中选择。

lessonList 有一个 @PlanningEntityCollectionProperty 注释，因此 OptaPlanner 可以在解决期间更改它们，您的 TimeTableConstraintProvider 也可以从它们中选择。

14.4. 创建 SOLVER 服务

解决 REST 线程上的规划问题会导致 HTTP 超时问题。因此，Quarkus 扩展注入一个 SolverManager，它在单独的线程池中运行 solvers，并可并行解决多个数据集。

流程

创建 src/main/java/org/acme/optaplanner/rest/TimeTableResource.java 类：

```
package org.acme.optaplanner.rest;

import java.util.UUID;
import java.util.concurrent.ExecutionException;
import javax.inject.Inject;
import javax.ws.rs.POST;
import javax.ws.rs.Path;

import org.acme.optaplanner.domain.TimeTable;
import org.optaplanner.core.api.solver.SolverJob;
import org.optaplanner.core.api.solver.SolverManager;

@Path("/timeTable")
public class TimeTableResource {

    @Inject
    SolverManager<TimeTable, UUID> solverManager;

    @POST
    @Path("/solve")
    public TimeTable solve(TimeTable problem) {
        UUID problemId = UUID.randomUUID();
        // Submit the problem to start solving
        SolverJob<TimeTable, UUID> solverJob = solverManager.solve(problemId, problem);
        TimeTable solution;
        try {
```

```
            // Wait until the solving ends
            solution = solverJob.getFinalBestSolution();
        } catch (InterruptedException | ExecutionException e) {
            throw new IllegalStateException("Solving failed.", e);
        }
        return solution;
    }

}
```

这个初始实施会等待 solver 完成，这仍然会导致 HTTP 超时。完整的实现可以避免 HTTP 超时很多。

14.5. 设置 SOLVER 终止时间

如果您的计划应用程序没有终止设置或终止事件，则理论上会永久运行，实际上最终会导致 HTTP 超时错误。要防止这种情况的发生，请使用 optaplanner.solver.termination.spent-limit 参数指定应用程序终止的时间长度。在大多数应用程序中，将时间设置为至少五分钟(5m)。但是，在 Timetable 示例中，将处理时间限制为 5 秒，这足以避免 HTTP 超时。

流程

使用以下内容创建 src/main/resources/application.properties 文件：

```
quarkus.optaplanner.solver.termination.spent-limit=5s
```

14.6. 运行 AMP TIMETABLE 应用程序

在创建了 school timetable 项目后，将其以开发模式运行。在开发模式中，您可以在应用程序运行时更新应用程序源和配置。您的更改将出现在正在运行的应用程序中。

先决条件

- 您已创建了 amp timetable 项目。

流程

1. 要编译应用程序为 development 模式，请从项目目录中输入以下命令：

```
./mvnw compile quarkus:dev
```

2.

测试 **REST** 服务。您可以使用任何 **REST** 客户端。以下示例使用 **Linux** 命令 **curl** 发送 **POST** 请求：

```
$ curl -i -X POST http://localhost:8080/timeTable/solve -H "Content-Type:application/json" -d
'{"timeslotList":[{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"}],"roomList":
[{"name":"Room A"},{"name":"Room B"}],"lessonList":[{"id":1,"subject":"Math","teacher":"A.
Turing","studentGroup":"9th grade"},{"id":2,"subject":"Chemistry","teacher":"M.
Curie","studentGroup":"9th grade"},{"id":3,"subject":"French","teacher":"M.
Curie","studentGroup":"10th grade"},{"id":4,"subject":"History","teacher":"I.
Jones","studentGroup":"10th grade"}]}'
```

在终止中指定的时间超过 **application.properties** 文件中定义的时间后，服务会返回类似以下示例的输出：

```
HTTP/1.1 200
Content-Type: application/json
...

{"timeslotList":...,"roomList":...,"lessonList":[{"id":1,"subject":"Math","teacher":"A.
Turing","studentGroup":"9th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":
{"name":"Room A"}},{"id":2,"subject":"Chemistry","teacher":"M. Curie","studentGroup":"9th
grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":
{"name":"Room A"}},{"id":3,"subject":"French","teacher":"M. Curie","studentGroup":"10th
grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":
{"name":"Room B"}},{"id":4,"subject":"History","teacher":"I. Jones","studentGroup":"10th
grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":
{"name":"Room B"}}],"score":"0hard/0soft"}
```

请注意，您的应用程序会将所有四个课程分配给两个时间插槽之一，两个两个房间一个。另请注意，它符合所有硬约束。例如，**M. Curie** 的两个课程位于不同的时间插槽中。

3. 要查看解决时间期间 **OptaPlanner** 执行的操作，请查看服务器端的信息日志。以下是 **info** 日志输出示例：

```
... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score
calculation speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score
calculation speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed
(28524/sec), phase total (2), environment mode (REPRODUCIBLE).
```

14.7. 测试应用程序

良好的应用程序包括测试覆盖。测试您的 timetable 项目中的限制和 solver。

14.7.1. 测试 education 时间表限制

要以隔离方式测试 timetable 项目的每个约束，请在单元测试中使用 ConstraintVerifier。这会测试每个约束的基写情况与其他测试隔离，这可在添加新的约束时降低维护。

此测试会验证当在同一房间给出三个课程时，这个测试会验证 constraint TimeTableConstraintProvider::roomConflict，两个课程具有相同的时间slot，并用匹配权重 1 来节省。因此，如果约束 weight 为 10hard，它将分数减少 -10hard。

流程

创建 src/test/java/org/acme/optaplanner/solver/TimeTableConstraintProviderTest.java 类：

```
package org.acme.optaplanner.solver;

import java.time.DayOfWeek;
import java.time.LocalTime;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.Room;
import org.acme.optaplanner.domain.TimeTable;
import org.acme.optaplanner.domain.Timeslot;
import org.junit.jupiter.api.Test;
import org.optaplanner.test.api.score.stream.ConstraintVerifier;

@QuarkusTest
class TimeTableConstraintProviderTest {

    private static final Room ROOM = new Room("Room1");
    private static final Timeslot TIMESLOT1 = new Timeslot(DayOfWeek.MONDAY,
LocalTime.of(9,0), LocalTime.NOON);
    private static final Timeslot TIMESLOT2 = new Timeslot(DayOfWeek.TUESDAY,
LocalTime.of(9,0), LocalTime.NOON);

    @Inject
    ConstraintVerifier<TimeTableConstraintProvider, TimeTable> constraintVerifier;

    @Test
    void roomConflict() {
        Lesson firstLesson = new Lesson(1, "Subject1", "Teacher1", "Group1");
```

```
        Lesson conflictingLesson = new Lesson(2, "Subject2", "Teacher2", "Group2");
        Lesson nonConflictingLesson = new Lesson(3, "Subject3", "Teacher3", "Group3");

        firstLesson.setRoom(ROOM);
        firstLesson.setTimeslot(TIMESLOT1);

        conflictingLesson.setRoom(ROOM);
        conflictingLesson.setTimeslot(TIMESLOT1);

        nonConflictingLesson.setRoom(ROOM);
        nonConflictingLesson.setTimeslot(TIMESLOT2);

        constraintVerifier.verifyThat(TimeTableConstraintProvider::roomConflict)
                .given(firstLesson, conflictingLesson, nonConflictingLesson)
                .penalizesBy(1);
    }

}
```

请注意，ConstraintVerifier 在测试过程中如何忽略约束权重，即使这些约束权重在 ConstraintProvider 中被硬编码。这是因为在进入生产前定期更改约束权重。这样，约束 weight tweaking 不会破坏单元测试。

14.7.2. 测试 Central timetable solver

本例在红帽构建的 Quarkus 平台上测试红帽构建的 OptaPlanner education timetable 项目。它使用 JUnit 测试来生成测试数据集并将其发送到 TimeTableController 以解决。

流程

1. 使用以下内容创建 src/test/java/com/example/rest/TimeTableResourceTest.java 类：

```
package com.exmaple.optaplanner.rest;

import java.time.DayOfWeek;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import com.exmaple.optaplanner.domain.Room;
import com.exmaple.optaplanner.domain.Timeslot;
import com.exmaple.optaplanner.domain.Lesson;
import com.exmaple.optaplanner.domain.TimeTable;
import com.exmaple.optaplanner.rest.TimeTableResource;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
```

```
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@QuarkusTest
public class TimeTableResourceTest {

    @Inject
    TimeTableResource timeTableResource;

    @Test
    @Timeout(600_000)
    public void solve() {
        TimeTable problem = generateProblem();
        TimeTable solution = timeTableResource.solve(problem);
        assertFalse(solution.getLessonList().isEmpty());
        for (Lesson lesson : solution.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(solution.getScore().isFeasible());
    }

    private TimeTable generateProblem() {
        List<Timeslot> timeslotList = new ArrayList<>();
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30),
LocalTime.of(9, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>();
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
        lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
        lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
        lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
        return new TimeTable(timeslotList, roomList, lessonList);
```

```
    }

}
```

此测试会验证在解决后，所有课程都分配给一个时间插槽和房间。它还会验证是否发现一种可行的解决方案（无硬限制）。

2. 在 src/main/resources/application.properties 文件中添加测试属性：

```
# The solver runs only for 5 seconds to avoid a HTTP timeout in this simple implementation.
# It's recommended to run for at least 5 minutes ("5m") otherwise.
quarkus.optaplanner.solver.termination.spent-limit=5s

# Effectively disable this termination in favor of the best-score-limit
%test.quarkus.optaplanner.solver.termination.spent-limit=1h
%test.quarkus.optaplanner.solver.termination.best-score-limit=0hard/*soft
```

通常，该方案在 200 毫秒内找到可行的解决方案。注意 application.properties 文件在测试过程中如何覆盖 solver 终止，以便在找到可行的解决方案 (0hardAttrsoft) 时立即终止。这可避免硬编码代码，因为单元测试可能会在任意硬件上运行。这种方法可确保测试运行时间足够长，以找到可行的解决方案，即使在速度较慢的系统上也是如此。但是，在快速系统中，它不会运行比严格必须长的 millisecond。

14.8. 日志记录

完成 Red Hat Build of OptaPlanner education timetable 项目后，您可以使用日志信息微调 ConstraintProvider 中的限制。查看 info 日志文件中的分数计算速度，以评估对您约束的更改的影响。以调试模式运行应用程序，以显示应用程序所采用的每个步骤，或使用 trace logging 来记录每个步骤和每个移动。

流程

1. 以固定时间（例如 5 分钟）运行 amp time。

2. 查看日志文件中的分数计算速度，如下例所示：

```
... Solving ended: ..., score calculation speed (29455/sec), ...
```

3. 更改约束，再次运行 planning 应用程序以相同的时间，并检查日志文件中记录的分数计算速度。

4. 以 debug 模式运行应用程序以记录应用程序所做的每个步骤：

   - 要从命令行运行调试模式，请使用 -D 系统属性。

   - 要永久启用 debug 模式，请在 application.properties 文件中添加以下行：

     ```
     quarkus.log.category."org.optaplanner".level=debug
     ```

     以下示例显示了日志文件中的 debug 模式的输出：

     ```
     ... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode (REPRODUCIBLE), random (JDK with seed 0).
     ...     CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15), picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
     ...     CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15), picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
     ...
     ```

5. 使用 trace logging 显示每个步骤和每个步骤。

14.9. 将数据库与您的 QUARKUS OPTAPLANNER RESEARCH TIMETABLE 应用程序集成

创建 Quarkus OptaPlanner PROFILE timetable 应用程序后，您可以将其与数据库集成，并创建基于 Web 的用户界面来显示可时间性。

先决条件

- 您有一个 Quarkus OptaPlanner school timetable 应用程序。

流程

1. 使用 Hibernate 和 Panache 将 Timeslot、Room 和 lesson 实例存储在数据库中。如需更多信息，请参阅使用 Panache 简化 Hibernate ORM。

2. 通过 REST 公开实例。如需更多信息，请参阅 编写 JSON REST 服务。

3.

*更新 **TimeTableResource** 类，在单个事务中读取和写入 **TimeTable** 实例：*

```
package org.acme.optaplanner.rest;

import javax.inject.Inject;
import javax.transaction.Transactional;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;

import io.quarkus.panache.common.Sort;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.Room;
import org.acme.optaplanner.domain.TimeTable;
import org.acme.optaplanner.domain.Timeslot;
import org.optaplanner.core.api.score.ScoreManager;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.solver.SolverManager;
import org.optaplanner.core.api.solver.SolverStatus;

@Path("/timeTable")
public class TimeTableResource {

    public static final Long SINGLETON_TIME_TABLE_ID = 1L;

    @Inject
    SolverManager<TimeTable, Long> solverManager;
    @Inject
    ScoreManager<TimeTable, HardSoftScore> scoreManager;

    // To try, open http://localhost:8080/timeTable
    @GET
    public TimeTable getTimeTable() {
        // Get the solver status before loading the solution
        // to avoid the race condition that the solver terminates between them
        SolverStatus solverStatus = getSolverStatus();
        TimeTable solution = findById(SINGLETON_TIME_TABLE_ID);
        scoreManager.updateScore(solution); // Sets the score
        solution.setSolverStatus(solverStatus);
        return solution;
    }

    @POST
    @Path("/solve")
    public void solve() {
        solverManager.solveAndListen(SINGLETON_TIME_TABLE_ID,
                this::findById,
                this::save);
    }

    public SolverStatus getSolverStatus() {
        return solverManager.getSolverStatus(SINGLETON_TIME_TABLE_ID);
    }
```

```
    @POST
    @Path("/stopSolving")
    public void stopSolving() {
        solverManager.terminateEarly(SINGLETON_TIME_TABLE_ID);
    }

    @Transactional
    protected TimeTable findById(Long id) {
        if (!SINGLETON_TIME_TABLE_ID.equals(id)) {
            throw new IllegalStateException("There is no timeTable with id (" + id + ").");
        }
        // Occurs in a single transaction, so each initialized lesson references the same
timeslot/room instance
        // that is contained by the timeTable's timeslotList/roomList.
        return new TimeTable(
                Timeslot.listAll(Sort.by("dayOfWeek").and("startTime").and("endTime").and("id")),
                Room.listAll(Sort.by("name").and("id")),
                Lesson.listAll(Sort.by("subject").and("teacher").and("studentGroup").and("id")));
    }

    @Transactional
    protected void save(TimeTable timeTable) {
        for (Lesson lesson : timeTable.getLessonList()) {
            // TODO this is awfully naive: optimistic locking causes issues if called by the
SolverManager
            Lesson attachedLesson = Lesson.findById(lesson.getId());
            attachedLesson.setTimeslot(lesson.getTimeslot());
            attachedLesson.setRoom(lesson.getRoom());
        }
    }

}
```

本例包含一个 TimeTable 实例。但是，您可以为并行多个 ultra 启用多租户和处理 TimeTable 实例。

getTimeTable （） 方法返回数据库的最新时间。它使用 ScoreManager 方法（自动注入）来计算可时间的分数，并使其可用于 UI。

solve （） 方法启动一个作业，以解决当前的可时间表，并将时间插槽和房间分配存储在数据库中。它使用 SolverManager.solveAndListen （） 方法侦听中间的解决方案，并相应地更新数据库。UI 在后端仍然解决时使用它来显示进度。

4. ***更新 TimeTableResourceTest 类，以反映 solve （） 方法立即返回，并轮询最新的解决方案，直到解决者完成解决为止：***

```
package org.acme.optaplanner.rest;
```

```
import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.TimeTable;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.optaplanner.core.api.solver.SolverStatus;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@QuarkusTest
public class TimeTableResourceTest {

    @Inject
    TimeTableResource timeTableResource;

    @Test
    @Timeout(600_000)
    public void solveDemoDataUntilFeasible() throws InterruptedException {
        timeTableResource.solve();
        TimeTable timeTable = timeTableResource.getTimeTable();
        while (timeTable.getSolverStatus() != SolverStatus.NOT_SOLVING) {
            // Quick polling (not a Test Thread Sleep anti-pattern)
            // Test is still fast on fast machines and doesn't randomly fail on slow machines.
            Thread.sleep(20L);
            timeTable = timeTableResource.getTimeTable();
        }
        assertFalse(timeTable.getLessonList().isEmpty());
        for (Lesson lesson : timeTable.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(timeTable.getScore().isFeasible());
    }

}
```

5. 在这些 REST 方法之上构建 Web UI，以提供可时间的可视化表示。

6. 查看 快速入门源代码。

14.10. 使用 MICROMETER 和 PROMETHEUS 监控您的 PROFILE OPTAPLANNER QUARKUS 应用程序

OptaPlanner 通过 Micrometer (Java 应用程序的指标检测库)公开指标。您可以将 Micrometer 与 Prometheus 搭配使用，以监控教育时应用程序中的 OptaPlanner solver。

先决条件

- *您已创建了 Quarkus OptaPlanner school timetable 应用程序。*
- *已安装 Prometheus。有关安装 Prometheus 的详情，请查看 Prometheus 网站。*

流程

1. *将 Micrometer Prometheus 依赖项添加到 Central timetable pom.xml 文件中：*

```
<dependency>
 <groupId>io.quarkus</groupId>
 <artifactId>quarkus-micrometer-registry-prometheus</artifactId>
</dependency>
```

2. *启动 amp timetable 应用程序：*

```
mvn compile quarkus:dev
```

3. *在 Web 浏览器中打开 http://localhost:8080/q/metric。*

第 15 章 红帽构建的 QUARKUS 上的 OPTAPLANNER 构建： VACCINATION APPOINTMENT 调度程序快速启动指南

您可以使用 OptaPlanner vaccination appointment 调度程序快速启动来开发高效和公平的 vaccination 调度。vaccination appointment 调度程序使用人工智能(AI)来优先选择人员并根据多个限制和优先级分配时间插槽。

先决条件

- 已安装了 OpenJDK 11 或更高版本。红帽构建的 Open JDK 可通过红帽客户门户网站中的 Software Downloads 页面（需要登录）。
- Apache Maven 3.8 或更高版本已安装。Maven 位于 Apache Maven Project 网站。
- 提供了 IDE，如 IntelliJ IDEA、VSCode 或 Eclipse。
- 您已在红帽构建的 Quarkus 平台项目上创建了一个 OptaPlanner 项目，如 第 5 章 在 Red Hat build of Quarkus 平台上开始使用 Red Hat Build of OptaPlanner 所述。

15.1. OPTAPLANNER VACCINATION APPOINTMENT 调度程序的工作方式

调度识别的方法主要有两种。系统可以让个人选择一个单点插槽（用户选择插槽），或者系统分配一个插槽，并告知个人参加的时间和位置（系统自动分配）。OptaPlanner vaccination appointment 调度程序使用 system-automatically-assigns 方法。通过 OptaPlanner vaccination appointment 调度程序，您可以创建一个应用程序，其中人员为其系统提供信息，系统分配了一个点数。

这个方法的特性：

- Appointment 插槽会根据优先级分配。
- 系统根据预配置的规划限制分配最佳评估时间和位置。
- 该系统并不是大量用户对数量有限数量的竞争。

这种方法通过使用计划限制为各个人创建分数，从而解决了尽可能多的人准确。人的分数决定何时获得 appointment。人的分数越高，他们获得早期的几分的机会越好。

15.1.1. 红帽构建的 OptaPlanner vaccination appointment 调度程序限制

Red Hat build of OptaPlanner vaccination appointment 调度程序限制是 hard, medium, 或 soft:

- 硬约束无法中断。如果任何硬约束被破坏，则计划不可取，无法执行：
  - 容量： 在任何位置都不要过度注册容量。
  - Vaccine 最长期限：如果 vaccine 有最长期限，则不要管理它给首次执行 vaccination 的人员不会超过 vaccine 最长期限。确保人们提供适合其年龄的 vaccine 类型。例如，不要为 vaccine 分配 7 年的旧人员，其最长期限限制为 65 年。
  - 必需 vaccine 类型：使用所需的 vaccine 类型。例如， vaccine 的第二个操作必须是与第一个 dose 相同的 vaccine 类型。
  - 就绪日期：管理指定日期或之后的 vaccine。例如，如果某人收到第二个操作，不要在特定 vaccine 类型的建议最早可能 vaccination 日期之前对其进行管理，例如，首次完成后的 26 天。
  - 到期日期：在指定日期或之前管理 vaccine。例如，如果某人收到第二个操作，请在特定 vaccine 的具体 vaccine 到期日期之前对其进行管理，例如，在首次执行之后 3 个月。
  - 限制最大旅行距离：将每个人员分配到最接近的 vaccination 数据中心之一。这通常是三个中心之一。此限制是通过差旅时间而不是距离计算的，因此位于某个房间的个人通常比农业领域有比个人的最大距离更小。
- 中等限制决定当没有足够容量来为每个人都分配 appointments 时，谁不会获得一个状况。这称为受限规划：
  - schedule second dose vaccinations：除非理想的日期不在计划窗口之外，否则不要使任何第二个 dose vaccination appointments 没有被分配。

- 根据优先级评级计划人员：每个人都有一个优先级评级。这通常是它们的年龄，但如果它们是健康的 worker，则它可能会更高。只有具有最低优先级评级的人员没有被分配。下一次运行时将考虑它们。这个约束比以前的约束软，因为第二个限制始终优先于优先级评级。

- 软限制不应中断：
  - 首选 vaccination Center ：如果个人具有首选的 vaccination Center，请给他们在该中心提供建议。
  - 距离：最小化个人必须前往其分配的票务中心的距离。
  - 理想日期：管理 vaccine on 或与指定日期接近。例如，如果某人收到第二个操作，则根据特定 vaccine 的理想日期对其进行管理，例如，首次执行 28 天。这个约束比 distance 约束的软者，以避免在国家/地区间向人发送半路，仅有一天更接近其理想的日期。
  - 优先级评级：调度之前在规划窗口中具有更高的优先级评级的人。这个约束比 distance 约束的软者，以避免在国家/地区间发送人员的一半。这个约束也比理想的日期约束较低，因为第二个操作优先于优先级评级。

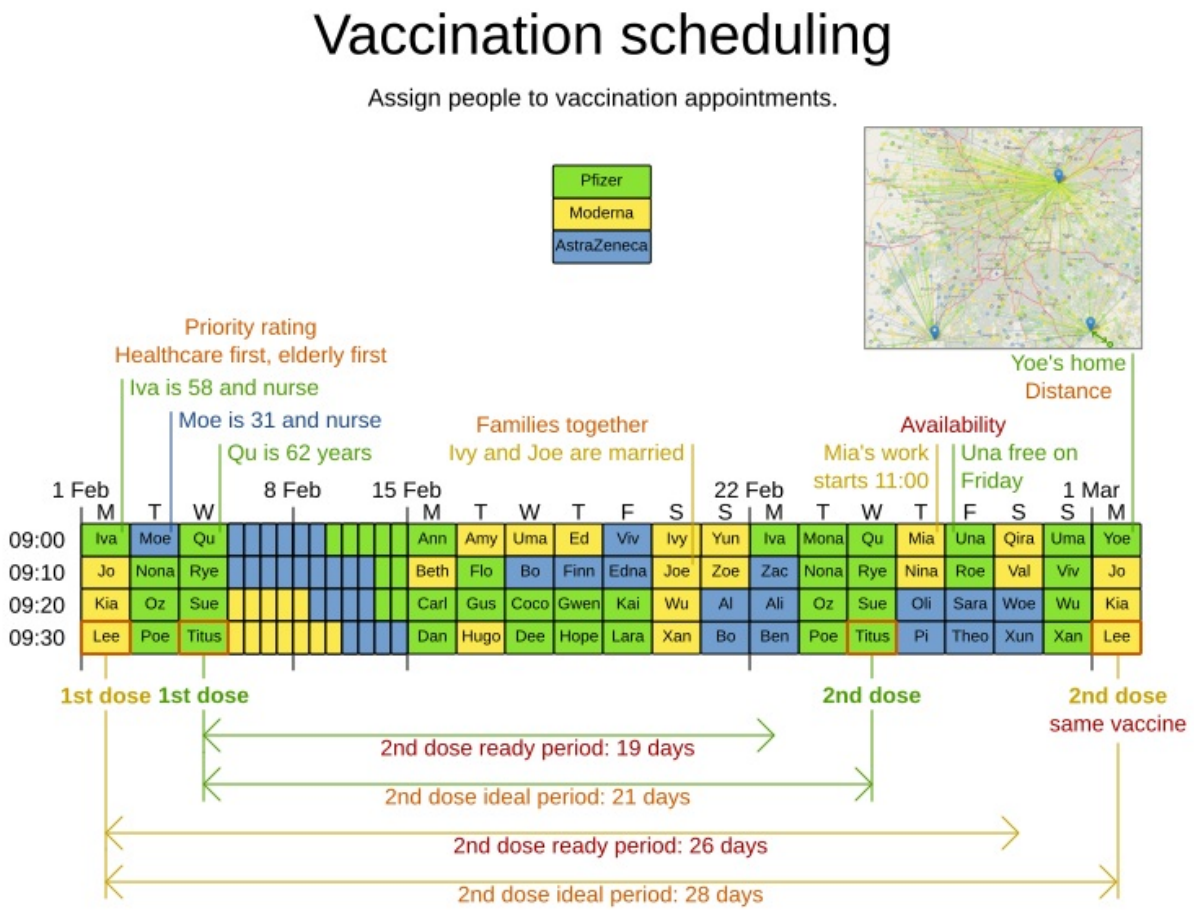
硬约束会根据其他硬限制来加权。软限制是针对其他软限制的权重。但是，硬限制总是优先于中型和软限制。如果硬约束被破坏，则计划不可行。但是，如果没有硬约束问题，则软和中等约束会被考虑以确定优先级。因为用户通常比可用的单点插槽更多，所以您必须优先选择。第二天首先被分配为始终分配点数，以避免创建稍后给您的系统造成放大的积压。之后，会根据优先级评级来分配人员。每个人都以优先级评级开头，即其年龄。这样做会优先选择较早的人。之后，位于特定优先级组的人员收到，例如，几百个额外的点。这因组的优先级而异。例如，nurses 可能会收到额外的 1000 个点。这样，较早的 nurses 优先考虑不久的人。下表描述了这个概念：

表 15.1. 优先级分级表

年龄	作业（job）	优先级评级
60	nurse	1060
33	nurse	1033
71	弃用	71
52	办公室工作者	52

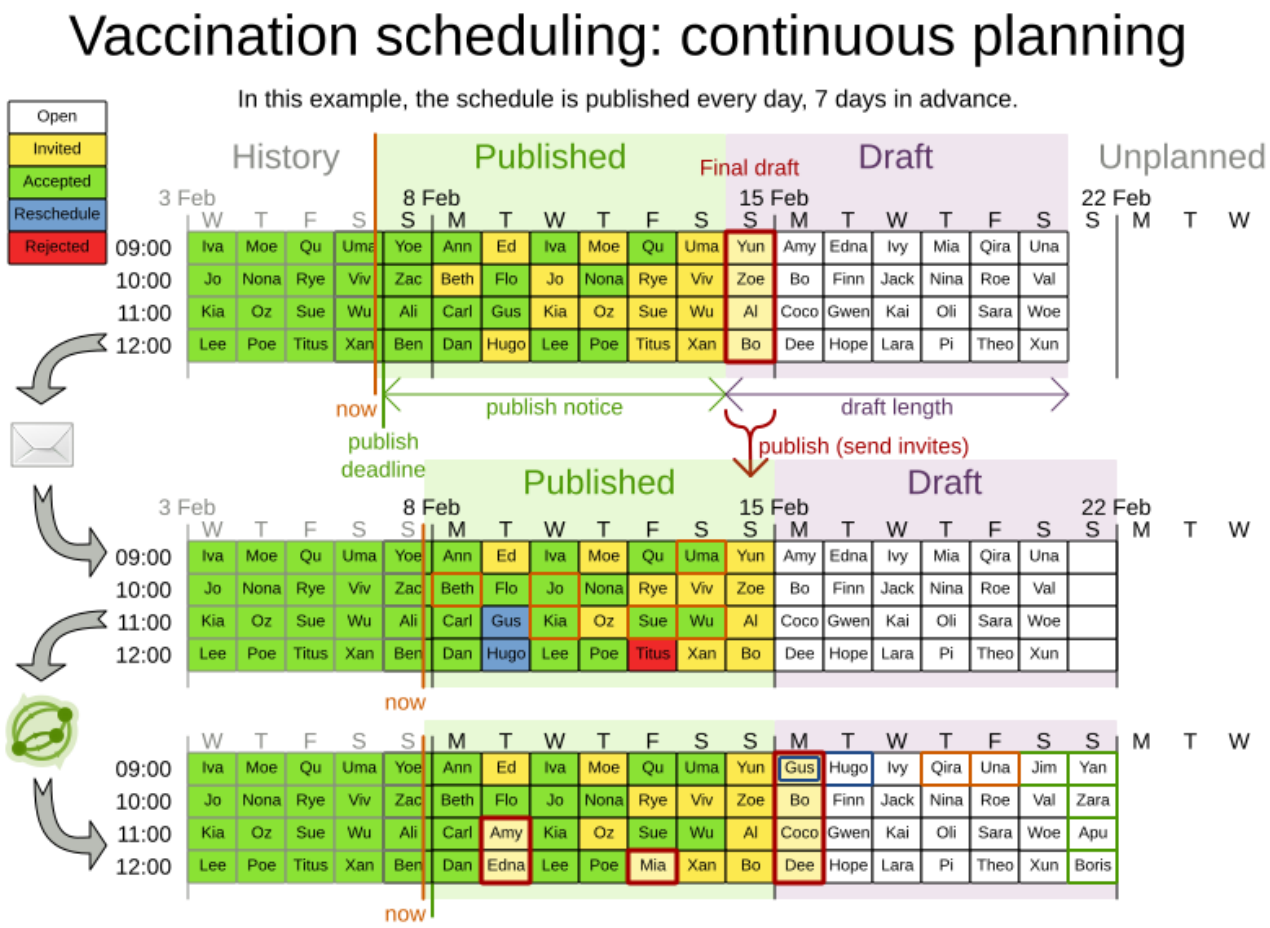
15.1.2. 红帽构建的 OptaPlanner solver

在 OptaPlanner 的核心是 solver，引擎会获取问题数据集并覆盖规划限制和配置。问题数据集包含有关人员、vaccines 和 vaccination 数据中心的所有信息。解决者通过各种数据组合工作，最终确定了特定中心分配给 vaccination appointments 的人员的优化情况。下图显示了 solver 创建的调度：



15.1.3. 持续规划

持续规划是同时管理一个或多个即将推出的规划周期的技术，并重复这个过程每月、每周、每天、每小时甚至更频繁地重复该过程。规划窗口以指定间隔逐步推进。下图显示了每天更新的两周规划窗口：



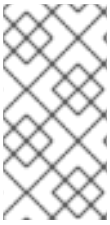
两周规划窗口将划分成一半。第一周处于 published 状态，第二周处于草案状态。人们在规划窗口的出版和草案部分中都分配到了一场点。但是，只有发布了计划窗口一部分的人员才会获得他们的意见通知。在下一次运行中，其他 appoints 仍然可以变化。这样做可让 OptaPlanner 在您再次运行解决方案时，灵活地更改草案中的点子部分。例如，如果需要第二天的个人准备日期为 Monday，并且理想的日期为 Wednesday，OptaPlanner 不必为他们留出一场点，如果您可以证明 OptaPlanner 可以演示它以后可以于周的草案。

您可以确定 planning 窗口的大小，但只了解问题空间的大小。问题空间是创建计划的各种元素。您提前规划的天数越大，问题空间越大。

15.1.4. 固定计划实体

如果您每天都要不断规划，那么在已经分配给了人的两周期间，将有点数。为确保 appointments 没有双书，OptaPlanner 将现有的 appointments 标记为固定它们所分配的。固定用于分配一个或多个特定分配，并强制 OptaPlanner 根据这些固定分配进行调度。固定计划实体（如 appointment）在解决过程中不会改变。

实体是固定的，是否由 appointment 状态决定。Appointment 可以有五种状态：Open、Invited、Accepted、Rejected 或 Rescheduled。



注意

您实际上不会在快速启动演示代码中直接看到这些状态，因为 OptaPlanner 引擎仅对 appointment 是否固定是否感兴趣。

您需要规划已计划的一个点数。已固定带有 Invited 或 Accepted 状态的 appointment。带有 Open、Reschedule 和 Rejected 状态的点数不会被固定，并可用于调度。

在本例中，当 solver 运行它在发布和草案范围内的整个双周规划窗口时，除了未计划输入数据外，该方案还考虑了任何未固定的实体，以及 开放、重新排期 或拒绝状态的任何非固定实体，以找到最佳解决方案。如果解析器每天运行，您将在运行 solver 前看到添加到计划的新日期。

请注意，新日的点数已被分配，并且之前在规划窗口草案中的 Amy 和 Edna 调度到该窗口的发布部分。这是因为 Gus 和 Hugo 请求重新调度。这不会造成混淆，因为 Amy 和 Edna 永远不会收到其草案日期的通知。现在，因为它们在计划窗口的公布的部分中有点，因此它们会被通知并要求接受或拒绝其 Appointments，现在被固定。

15.2. 下载并运行 OPTAPLANNER VACCINATION APPOINTMENT 调度程序

下载 OptaPlanner vaccination appointment 调度程序快速启动存档，以 Quarkus 开发模式启动它，并在浏览器中查看应用程序。Quarkus 开发模式允许您在应用程序运行时进行更改和更新。

流程

1. 进入红帽客户门户网站中的 Software Downloads 页面（需要登录），然后从下拉菜单中选择产品和版本：
   - 产品： 红帽构建的 OptaPlanner
   - Version: 8.38
2. 下载 红帽构建的 OptaPlanner 8.38 快速入门。
3. 提取 rhbop-8.38.0-optaplanner-quickstarts-sources.zip 文件。

提取的 org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/use-cases/vaccination-scheduling 目录包含示例源代码。

4. 导航到 org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/use-cases/vaccination-scheduling 目录。

5. 输入以下命令在开发模式中启动 OptaPlanner vaccination appointment 调度程序：

```
$ mvn quarkus:dev
```

6. 要查看 OptaPlanner vaccination appointment 调度程序，请在网页浏览器中输入以下 URL：

```
http://localhost:8080/
```

7. 要运行 OptaPlanner vaccination appointment 调度程序，请单击 Solve。

8. 对源代码进行更改，然后按 F5 键刷新浏览器。请注意，您所做的更改现已可用。

15.3. 软件包并运行 OPTAPLANNER VACCINATION APPOINTMENT 调度程序

当您在 quarkus:dev 模式中完成对 OptaPlanner vaccination appointment 调度程序的开发工作后，将应用程序作为传统的 jar 文件运行。

先决条件

- 您已下载了 OptaPlanner vaccination appointment 调度程序快速启动。

流程

1. 导航到 /use-cases/vaccination-scheduling 目录。

2. 要编译 OptaPlanner vaccination appointment 调度程序，请输入以下命令：

```
$ mvn package
```

3. *要运行编译的 OptaPlanner vaccination appointment 调度程序，请输入以下命令：*

```
$ java -jar ./target/quarkus-app/quarkus-run.jar
```

注意

要在端口 8081 上运行应用程序，请将 -Dquarkus.http.port=8081 添加到上一命令。

4. *要启动 OptaPlanner vaccination appointment 调度程序，请在网页浏览器中输入以下 URL：*

```
http://localhost:8080/
```

15.4. 其他资源

- *Vaccination appointment scheduling video*

第 16 章 红帽构建的 QUARKUS 上的 OPTAPLANNER 构建：员工调度程序快速启动指南

员工调度程序快速启动应用程序为员工分配在组织的不同位置上转换的员工。例如，您可以使用应用程序在 nurses、保护职责跨多个位置转换或在工作程序之间的装配行之间分发转换。

最佳员工调度必须考虑多个变量。例如，在不同位置上转移可能需要不同的技能。另外，一些员工可能不适用于某些时间插槽，或者可能首选使用特定的时间插槽。此外，员工还可以拥有一个合同，限制员工在单一时间段内工作的小时数。

此入门应用程序的红帽构建的 OptaPlanner 规则使用硬和软限制。在优化过程中，Planner 引擎可能无法违反硬限制，例如，如果员工不可用(sick)，或者员工无法在单一转换中处理两个点。Planner 引擎会尝试遵循软限制，如员工更喜欢不工作特定转换，但如果最佳解决方案需要它，则可以违反它们。

先决条件

- 已安装了 OpenJDK 11 或更高版本。红帽构建的 Open JDK 可通过红帽客户门户网站中的 Software Downloads 页面（需要登录）。
- Apache Maven 3.8 或更高版本已安装。Maven 位于 Apache Maven Project 网站。
- 提供了 IDE，如 IntelliJ IDEA、VSCode 或 Eclipse。

16.1. 下载并运行 OPTAPLANNER 员工调度程序

下载 OptaPlanner 员工调度程序快速启动存档，以 Quarkus 开发模式启动它，并在浏览器中查看应用程序。Quarkus 开发模式允许您在应用程序运行时进行更改和更新。

流程

1. 进入红帽客户门户网站中的 Software Downloads 页面（需要登录），然后从下拉菜单中选择产品和版本：
   - 产品：红帽构建的 OptaPlanner
   - Version: 8.38

2. 下载 红帽构建的 OptaPlanner 8.38 快速入门。

3. 提取 rhbop-8.38.0-optaplanner-quickstarts-sources.zip 文件。

4. 导航到 org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/use-cases/employee-scheduling 目录。

5. 输入以下命令在开发模式下启动 OptaPlanner employee 调度程序：

```
$ mvn quarkus:dev
```

6. 要查看 OptaPlanner 员工调度程序，请在网页浏览器中输入以下 URL：

```
http://localhost:8080/
```

7. 要运行 OptaPlanner 员工调度程序，请单击 Solve。

8. 对源代码进行更改，然后按 F5 键刷新浏览器。请注意，您所做的更改现已可用。

16.2. 软件包并运行 OPTAPLANNER 员工调度程序

当您完成开发工作(quarkus:dev 模式)中的 OptaPlanner 员工调度程序后，将应用作为传统的 jar 文件运行。

先决条件

- 您已下载了 OptaPlanner 员工快速调度。

流程

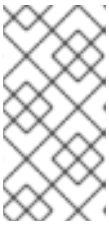
1. 导航到 /use-cases/vaccination-scheduling 目录。

2. 要编译 OptaPlanner 员工调度程序，请输入以下命令：

```
$ mvn package
```

3. 要运行编译的 OptaPlanner employee 调度程序，请输入以下命令：

```
$ java -jar ./target/quarkus-app/quarkus-run.jar
```



注意

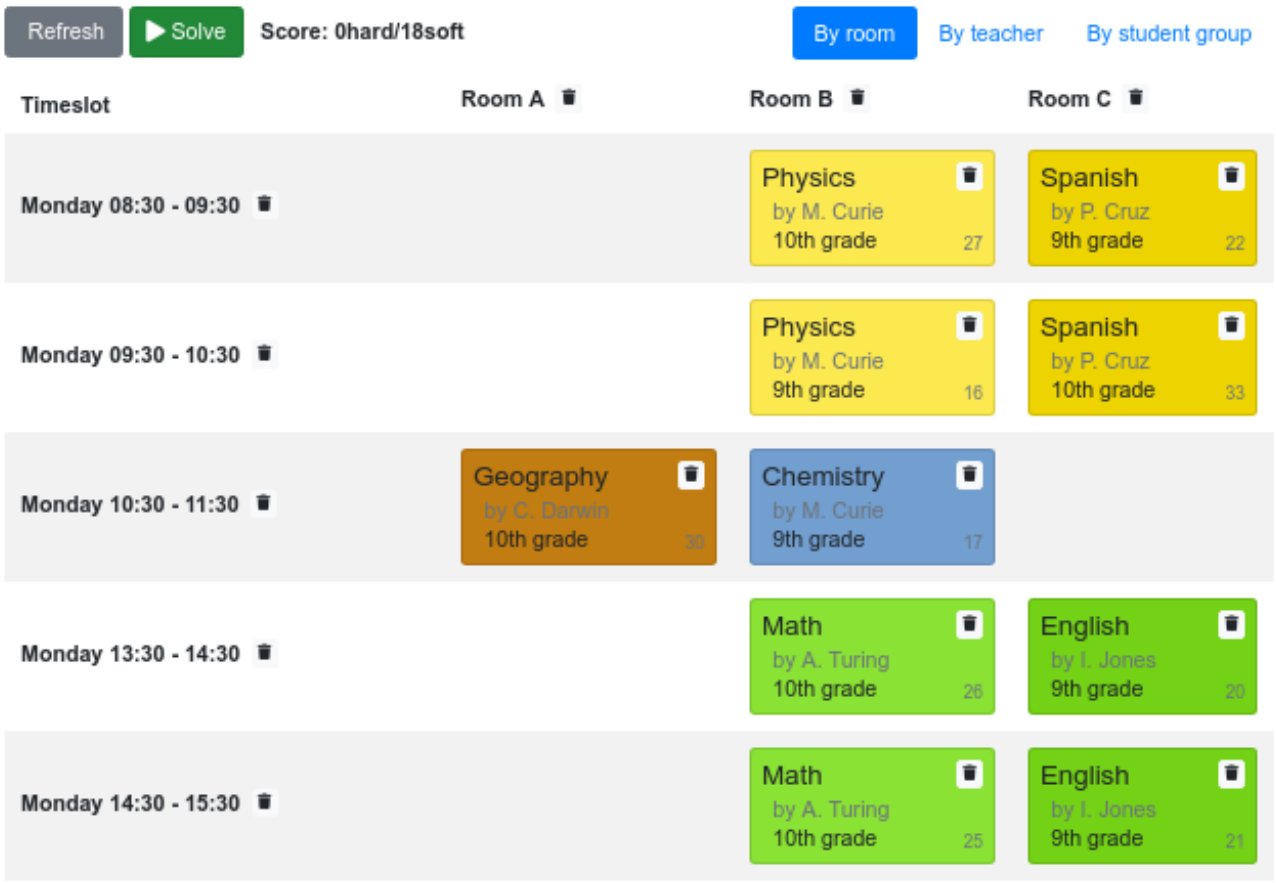
要在端口 8081 上运行应用程序，请将 -Dquarkus.http.port=8081 添加到上一命令。

4. 要启动 OptaPlanner 员工调度程序，请在网页浏览器中输入以下 URL。

```
http://localhost:8080/
```

第 17 章 RED HAT BUILD OF OPTAPLANNER ON SPRING BOOT: A TUTORIAL TIMEABLE QUICK START 指南

本指南指导您完成使用 OptaPlanner 的约束创建 Spring Boot 应用程序的过程，用于解决人工智能 (AI)。您将构建一个 REST 应用程序，为教育人员和教员优化一个生存时间。



您的服务将使用 AI 自动将 lesson 实例分配给 Timeslot 和 Room 实例，以遵循以下硬和软 调度限制：

- 房间最多可以有一门。
- 教员可以在大多数课程同时进行指导。
- 学员最多可同时参加一门课程。
- 教员更喜欢在单一房间发言。

- 教师希望检测后续课程和课程之间的差别。

以数学方式说，中型时间稳定是一个 NP-hard 的问题。这意味着扩展比较困难。简单地迭代所有带有 brute 强制的可能组合，即使是超级计算机，则需要数百万年时间。satisfy, AI 约束解析器（如 OptaPlanner）具有在合理的时间内提供接近最佳解决方案的高级算法。认为是合理的时间，这取决于您的问题的目标。

先决条件

- 已安装了 OpenJDK 11 或更高版本。红帽构建的 Open JDK 可通过红帽客户门户网站中的 Software Downloads 页面（需要登录）。
- Apache Maven 3.8 或更高版本已安装。Maven 位于 Apache Maven Project 网站。
- 提供了 IDE，如 IntelliJ IDEA、VSCode 或 Eclipse。

17.1. 下载并构建 SPRING BOOT SCHOOL TIMETABLE 快速启动

如果您要看到红帽构建的带有 Spring Boot 产品的 OptaPlanner 的可完成时间示例，请从红帽客户门户网站下载初学者应用程序。

流程

1. 进入红帽客户门户网站中的 Software Downloads 页面（需要登录），然后从下拉菜单中选择产品和版本：
   - 产品： 红帽构建的 OptaPlanner
   - Version: 8.38
2. 下载 红帽构建的 OptaPlanner 8.38 快速入门。
3. 提取 rhbop-8.38.0-optaplanner-quickstarts-sources.zip 文件。

提取的 org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/use-cases/school-timetabling 目录包含示例源代码。

4. *导航到 org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/use-cases/school-timetabling 目录。*

5. *下载 红帽构建的 OptaPlanner 8.38.0 Maven Repositroy (rhbop-8.38.0-optaplanner-maven-repository.zip)。*

6. *提取 rhbop-8.38.0-optaplanner-maven-repository.zip 文件。*

7. *将 rhbop-8.38.0-optaplanner/maven-repository 子目录的内容复制到 ~/.m2/repository 目录中。*

8. *导航到 org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/technical/java-spring-boot 目录。*

9. *输入以下命令构建 Spring Boot school timetabling 项目：*

```
mvn clean install -DskipTests
```

10. *要构建 Spring Boot CLASS timetabling 项目，请输入以下命令：*

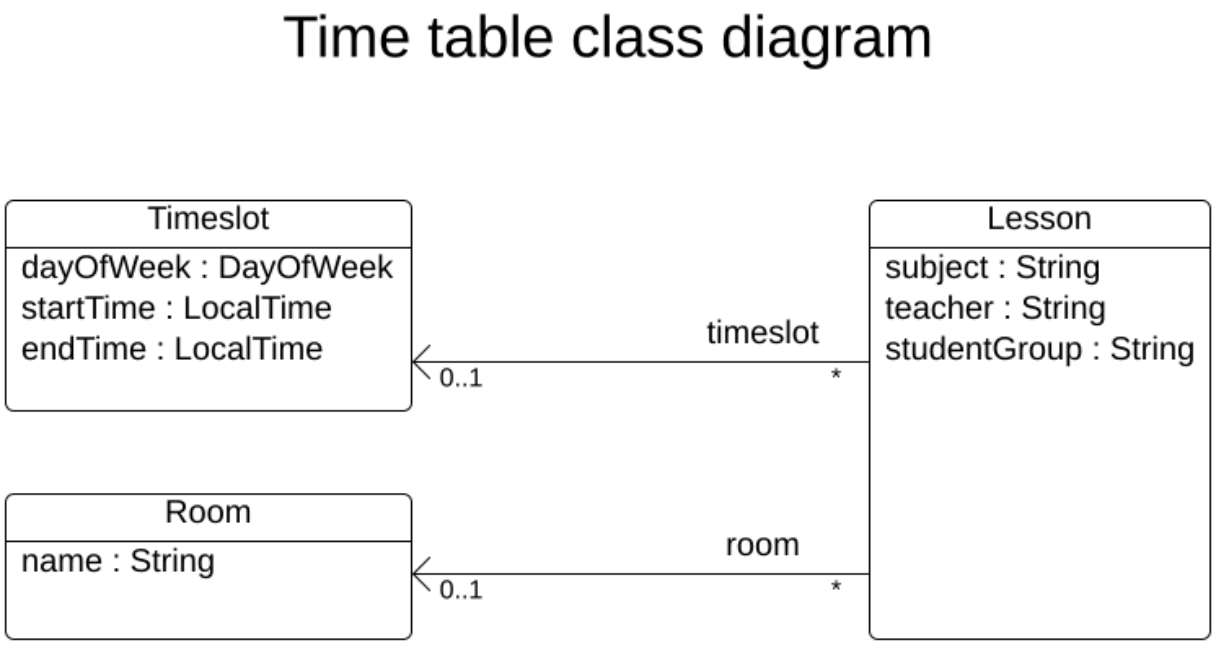
```
mvn spring-boot:run -DskipTests
```

11. *要查看项目，请在网页浏览器中输入以下 URL：*

```
http://localhost:8080/
```

17.2. 对域对象建模

红帽构建的 OptaPlanner timetable 项目的目标是为每个课程分配时间插槽和房间。要做到这一点，请添加三个类，Timeslot , lesson , 和 Room，如下图所示：



timeslot

***Timeslot** 类代表了在课程时的间隔，例如 **Monday 10:30 - 11:30** 或 **Tuesday 13:30 - 14:30**。在这个示例中，所有时间插槽都有相同的持续时间，在 **lunch** 或其他间没有时间插槽。*

*时间段没有日期，因为高校计划每周都会重复。不需要 持续规划。一个 **timeslot** 被称为 问题，因为在解决过程中没有 **Timeslot** 实例改变。此类类不需要任何 **OptaPlanner** 特定注解。*

房间

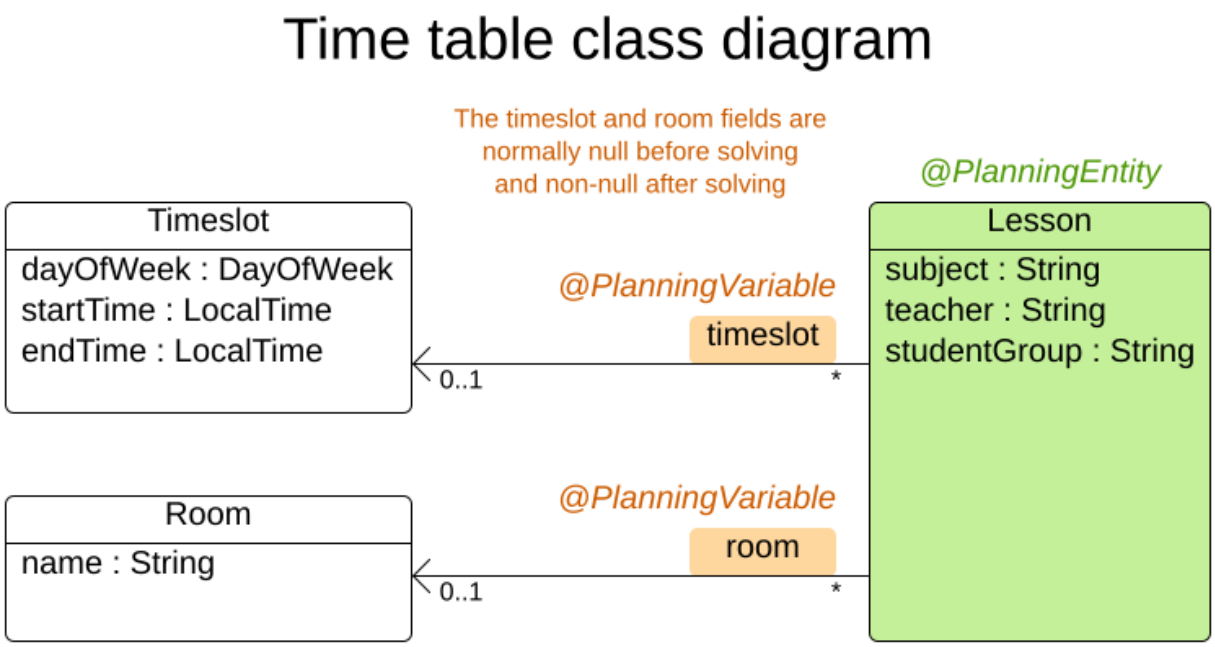
***Room** 类代表一个跟踪课程的位置，例如 **Room A** 或 **Room B**。在这个示例中，所有房间都没有容量限制，它们可以容纳所有课程。*

房间 实例在解决时 不会改变，因此也 是一个问题。

lesson

*在短时间内，由 **Lesson** 类表示，教员向一组学员提供主题，例如 **Math by A.Turing for 9th grade** 或 **Chemistry by M.Curie for 10th grade**。如果每周由同一生组多次学习一个主题，则有多个仅通过 **id** 区分的 **lesson** 实例。例如，第 **9** 个评级每周有 **6** 个数个。*

*在解决期间，**OptaPlanner** 更改了 **Lesson** 类的 **timeslot** 和 **room** 字段，为每个课程分配时间插槽和房间。因为 **OptaPlanner** 更改这些字段，所以 **lesson** 是一个 规划实体 ：*



上图中的大部分字段包含输入数据，但 orange 字段除外。一个 lesson 的 timeslot 和 room 字段在输入数据中未分配（空），并在输出数据中分配（非 null）。OptaPlanner 在解决过程中更改这些字段。此类字段称为计划变量。为了让 OptaPlanner 识别它们，timeslot 和 room 字段都需要 @PlanningVariable 注释。它包含类 lesson 需要一个 @PlanningEntity 注释。

流程

1. *创建 src/main/java/com/example/domain/Timeslot.java 类：*

```
package com.example.domain;

import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;

    private Timeslot() {
    }

    public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
        this.dayOfWeek = dayOfWeek;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    @Override
```

```
    public String toString() {
        return dayOfWeek + " " + startTime.toString();
    }

    // ********************************
    // Getters and setters
    // ********************************

    public DayOfWeek getDayOfWeek() {
        return dayOfWeek;
    }

    public LocalTime getStartTime() {
        return startTime;
    }

    public LocalTime getEndTime() {
        return endTime;
    }

}
```

注意 toString （） 方法保持输出短，以便更轻松地阅读 OptaPlanner 的 DEBUG 或 TRACE 日志，如稍后所示。

2. 创建 src/main/java/com/example/domain/Room.java 类：

```
package com.example.domain;

public class Room {

    private String name;

    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public String getName() {
        return name;
```

```
    }

}
```

3. 创建 src/main/java/com/example/domain/Lesson.java 类：

```
package com.example.domain;

import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
    }

    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // ********************************
    // Getters and setters
    // ********************************

    public Long getId() {
        return id;
    }

    public String getSubject() {
        return subject;
    }

    public String getTeacher() {
```

```
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {
        return timeslot;
    }

    public void setTimeslot(Timeslot timeslot) {
        this.timeslot = timeslot;
    }

    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
        this.room = room;
    }

}
```

Lesson 类具有一个 @PlanningEntity 注释，因此 OptaPlanner 知道此类在解决过程中发生了变化，因为它包含一个或多个计划变量。

timeslot 字段具有一个 @PlanningVariable 注释，因此 OptaPlanner 知道它可以更改其值。为了找到潜在的 Timeslot 实例来分配给此字段，OptaPlanner 使用 valueRangeProviderRefs 属性连接到提供 List<Timeslot> 的值范围供应商。有关值范围供应商的信息，请参阅 第 17.4 节 “在规划解决方案中收集域对象”。

room 字段还具有 @PlanningVariable 注释，原因相同。

17.3. 定义限制并计算分数

在解决问题时，分数 代表特定解决方案的质量。分数越高。红帽构建的 OptaPlanner 寻求最佳解决方案，这是在可用时间内获得最高分数的解决方案。这可能是 最佳解决方案。

因为 timetable 示例用例具有硬和软限制，所以请使用 HardSoftScore 类来代表分数：

- 硬约束不能被破坏。例如 ：一个空间最多可以同时有一个。

- 软限制不应中断。例如 ：教员更倾向于在单一房间进行学习。

硬约束会根据其他硬限制来加权。软限制是针对其他软限制的权重。硬限制始终超过软约束，无论其对应的权重是什么。

要计算分数，您可以实施 EasyScoreCalculator 类：

```
public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

   @Override
   public HardSoftScore calculateScore(TimeTable timeTable) {
      List<Lesson> lessonList = timeTable.getLessonList();
      int hardScore = 0;
      for (Lesson a : lessonList) {
         for (Lesson b : lessonList) {
            if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
                  && a.getId() < b.getId()) {
               // A room can accommodate at most one lesson at the same time.
               if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                  hardScore--;
               }
               // A teacher can teach at most one lesson at the same time.
               if (a.getTeacher().equals(b.getTeacher())) {
                  hardScore--;
               }
               // A student can attend at most one lesson at the same time.
               if (a.getStudentGroup().equals(b.getStudentGroup())) {
                  hardScore--;
               }
            }
         }
      }
      int softScore = 0;
      // Soft constraints are only implemented in the "complete" implementation
      return HardSoftScore.of(hardScore, softScore);
   }

}
```

不幸的是，这个解决方案无法很好地扩展，因为它并没有递增：每次将课程分配给不同的时间插槽或房间，所有课程都会被重新评估以计算新的分数。

更好的解决方法是创建一个 src/main/java/com/example/solver/TimeTableConstraintProvider.java 类，以执行增量分数计算。此类使用 OptaPlanner 的 ConstraintStream API，该 API 由 Java 8 Streams 和 SQL 推进。ConstraintProvider 比 EasyScoreCalculator 增加了一个比 EasyScoreCalculator 更强的 magnitude 顺序：O(n)而不是 O(nbusybox)。

流程

创建以下 src/main/java/com/example/solver/TimeTableConstraintProvider.java 类：

```
package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
                // Hard constraints
                roomConflict(constraintFactory),
                teacherConflict(constraintFactory),
                studentGroupConflict(constraintFactory),
                // Soft constraints are only implemented in the "complete" implementation
        };
    }

    private Constraint roomConflict(ConstraintFactory constraintFactory) {
        // A room can accommodate at most one lesson at the same time.

        // Select a lesson ...
        return constraintFactory.forEach(Lesson.class)
                // ... and pair it with another lesson ...
                .join(Lesson.class,
                        // ... in the same timeslot ...
                        Joiners.equal(Lesson::getTimeslot),
                        // ... in the same room ...
                        Joiners.equal(Lesson::getRoom),
                        // ... and the pair is unique (different id, no reverse pairs)
                        Joiners.lessThan(Lesson::getId))
                // then penalize each pair with a hard weight.
                .penalize(HardSoftScore.ONE_HARD)
                .asConstraint("Room conflict");
    }

    private Constraint teacherConflict(ConstraintFactory constraintFactory) {
        // A teacher can teach at most one lesson at the same time.
        return constraintFactory.forEach(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getTeacher),
                        Joiners.lessThan(Lesson::getId))
                        .penalize(HardSoftScore.ONE_HARD)
                        .asConstraint("Teacher conflict");
    }
```

```
    private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
        // A student can attend at most one lesson at the same time.
        return constraintFactory.forEach(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getStudentGroup),
                        Joiners.lessThan(Lesson::getId))
                        .penalize(HardSoftScore.ONE_HARD)
                        .asConstraint("Student group conflict");
    }

}
```

17.4. 在规划解决方案中收集域对象

TimeTable 实例会打包单个 数据集 的所有 TimeTable、Room 和 lesson 实例。另外，因为它包含所有课程，每个课程都有特定的规划变量状态，所以它是一个 规划解决方案，它分数如下：

- 如果尚未分配课程，则它是一个 未初始化的 解决方案，例如，分数为 -4init/0hard/0soft 的解决方案。
- 如果它中断了硬限制，则它是一个 不可避免的 解决方案，例如分数为 -2hard/-3soft 的解决方案。
- 如果它遵循所有硬限制，则它是一个可行的解决方案，例如分数为 0hard/-7soft 的解决方案。

TimeTable 类具有 @PlanningSolution 注解，因此红帽构建的 OptaPlanner 知道此类包含所有输入和输出数据。

特别是，这个类是问题的输入：

- 包含所有时间插槽的 timeslotList 字段
  - 这是问题事实列表，因为它们在解决过程中不会改变。
- 带有所有房间的 roomList 字段

- 这是问题事实列表，因为它们在解决过程中不会改变。

- 具有所有课程的 lessonList 字段

  - 这是规划实体列表，因为它们在解决过程中发生了变化。

  - 每个 lesson：

    - timeslot 和 room 字段的值通常仍然为 null，因此未分配。它们是规划变量。

    - 其他字段（如 主题、sonr 和 studentGroup ）已填写。这些字段是问题属性。

但是，这个类也是解决方案的输出：

- 在解决解决后，每个 Lesson 实例都有非null 时间slot 和 room 字段的 lessonList 字段

- 代表输出解决方案的质量的 score 字段，例如 0hard/-5soft

流程

创建 src/main/java/com/example/domain/TimeTable.java 类：

```
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;

@PlanningSolution
public class TimeTable {

   @ValueRangeProvider(id = "timeslotRange")
```

```
    @ProblemFactCollectionProperty
    private List<Timeslot> timeslotList;

    @ValueRangeProvider(id = "roomRange")
    @ProblemFactCollectionProperty
    private List<Room> roomList;

    @PlanningEntityCollectionProperty
    private List<Lesson> lessonList;

    @PlanningScore
    private HardSoftScore score;

    private TimeTable() {
    }

    public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
            List<Lesson> lessonList) {
        this.timeslotList = timeslotList;
        this.roomList = roomList;
        this.lessonList = lessonList;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public List<Timeslot> getTimeslotList() {
        return timeslotList;
    }

    public List<Room> getRoomList() {
        return roomList;
    }

    public List<Lesson> getLessonList() {
        return lessonList;
    }

    public HardSoftScore getScore() {
        return score;
    }

}
```

值范围供应商

timeslotList 字段是一个值范围供应商。它保存 OptaPlanner 可以从中选择的 Timeslot 实例，以分配给 lesson 实例的 timeslot 字段。timeslotList 字段具有一个 @ValueRangeProvider 注释，用于将 id 与 lesson 中 @PlanningVariable 的 @PlanningVariable 的 id 匹配。

遵循相同的逻辑后，roomList 字段也具有 @ValueRangeProvider 注释。

问题事实和规划实体属性

此外，OptaPlanner 需要知道它可以更改哪些更少的实例，以及如何检索用于您的 TimeTableConstraintProvider 分数计算的 Timeslot 和 Room 实例。

timeslotList 和 roomList 字段具有一个 @ProblemFactCollectionProperty 注释，因此您的 TimeTableConstraintProvider 可以从这些实例中选择。

lessonList 有一个 @PlanningEntityCollectionProperty 注释，因此 OptaPlanner 可以在解决期间更改它们，您的 TimeTableConstraintProvider 也可以从它们中选择。

17.5. 创建 TIMETABLE 服务

现在，您已准备好将所有内容放在一起并创建 REST 服务。但解决 REST 线程上的规划问题会导致 HTTP 超时问题。因此，Spring Boot 初学者注入 SolverManager，它在单独的线程池中运行 solvers，并可并行解决多个数据集。

流程

创建 src/main/java/com/example/solver/TimeTableController.java 类：

```
package com.example.solver;

import java.util.UUID;
import java.util.concurrent.ExecutionException;

import com.example.domain.TimeTable;
import org.optaplanner.core.api.solver.SolverJob;
import org.optaplanner.core.api.solver.SolverManager;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/timeTable")
public class TimeTableController {

    @Autowired
    private SolverManager<TimeTable, UUID> solverManager;

    @PostMapping("/solve")
    public TimeTable solve(@RequestBody TimeTable problem) {
        UUID problemId = UUID.randomUUID();
        // Submit the problem to start solving
```

```
        SolverJob<TimeTable, UUID> solverJob = solverManager.solve(problemId, problem);
        TimeTable solution;
        try {
            // Wait until the solving ends
            solution = solverJob.getFinalBestSolution();
        } catch (InterruptedException | ExecutionException e) {
            throw new IllegalStateException("Solving failed.", e);
        }
        return solution;
    }

}
```

在本例中，初始实施会等待 solver 完成，这仍然可以导致 HTTP 超时。完整的实现可以避免 HTTP 超时很多。

17.6. 设置 SOLVER 终止时间

如果您的计划应用程序没有终止设置或终止事件，则理论上会永久运行，实际上最终会导致 HTTP 超时错误。要防止这种情况的发生，请使用 optaplanner.solver.termination.spent-limit 参数指定应用程序终止的时间长度。在大多数应用程序中，将时间设置为至少五分钟(5m)。但是，在 Timetable 示例中，将处理时间限制为 5 秒，这足以避免 HTTP 超时。

流程

使用以下内容创建 src/main/resources/application.properties 文件：

```
quarkus.optaplanner.solver.termination.spent-limit=5s
```

17.7. 使应用程序可执行

完成 Red Hat Build of OptaPlanner Spring Boot timetable 项目后，将所有内容打包成由标准 Java main （） 方法驱动的单个可执行 JAR 文件。

先决条件

- 您有一个已完成的 OptaPlanner Spring Boot timetable 项目。

流程

1. 使用以下内容创建 TimeTableSpringBootApp.java 类：

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TimeTableSpringBootApp {

    public static void main(String[] args) {
        SpringApplication.run(TimeTableSpringBootApp.class, args);
    }

}
```

2. 将由 Spring Initializr 创建的 src/main/java/com/example/DemoApplication.java 类替换为 TimeTableSpringBootApp.java 类。

3. 将 TimeTableSpringBootApp.java 类作为常规 Java 应用程序的主类运行。

17.7.1. 尝试 timetable 应用程序

启动 Red Hat Build of OptaPlanner Spring Boot timetable 应用程序后，您可以使用您想要的任何 REST 客户端测试 REST 服务。本例使用 Linux curl 命令发送 POST 请求。

先决条件

- OptaPlanner Spring Boot timetable 应用程序正在运行。

流程

使用以下命令：

```
$ curl -i -X POST http://localhost:8080/timeTable/solve -H "Content-Type:application/json" -d
'{"timeslotList":[{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"}],"roomList":[{"name":"Room
A"},{"name":"Room B"}],"lessonList":[{"id":1,"subject":"Math","teacher":"A. Turing","studentGroup":"9th
grade"},{"id":2,"subject":"Chemistry","teacher":"M. Curie","studentGroup":"9th grade"},
{"id":3,"subject":"French","teacher":"M. Curie","studentGroup":"10th grade"},
{"id":4,"subject":"History","teacher":"I. Jones","studentGroup":"10th grade"}]}'
```

大约 5 秒后，终止会花费在 application.properties 中定义的时间，服务会返回类似以下示例的输出：

```
HTTP/1.1 200
Content-Type: application/json
...

{"timeslotList":...,"roomList":...,"lessonList":[{"id":1,"subject":"Math","teacher":"A.
Turing","studentGroup":"9th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":{"name":"Room A"}},
{"id":2,"subject":"Chemistry","teacher":"M. Curie","studentGroup":"9th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":{"name":"Room A"}},
{"id":3,"subject":"French","teacher":"M. Curie","studentGroup":"10th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":{"name":"Room B"}},
{"id":4,"subject":"History","teacher":"I. Jones","studentGroup":"10th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":{"name":"Room
B"}}],"score":"0hard/0soft"}
```

请注意，应用程序会将所有四个课程分配给两个时间插槽之一，另一个是两个房间。另请注意，它符合所有硬约束。例如，M. Curie 的两个课程位于不同的时间插槽中。

在服务器端，info 日志显示 OptaPlanner 在 5 秒内执行的操作：

```
... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score calculation
speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score calculation
speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed (28524/sec),
phase total (2), environment mode (REPRODUCIBLE).
```

17.7.2. 测试应用

良好的应用程序包括测试覆盖。这个示例测试 Timetable Red Hat Build of OptaPlanner Spring Boot 应用程序。它使用 JUnit 测试来生成测试数据集并将其发送到 TimeTableController 以解决。

流程

使用以下内容创建 src/test/java/com/example/solver/TimeTableControllerTest.java 类：

```
package com.example.solver;

import java.time.DayOfWeek;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

import com.example.domain.Lesson;
import com.example.domain.Room;
```

```
import com.example.domain.TimeTable;
import com.example.domain.Timeslot;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@SpringBootTest(properties = {
        "optaplanner.solver.termination.spent-limit=1h", // Effectively disable this termination in
favor of the best-score-limit
        "optaplanner.solver.termination.best-score-limit=0hard/*soft"})
public class TimeTableControllerTest {

    @Autowired
    private TimeTableController timeTableController;

    @Test
    @Timeout(600_000)
    public void solve() {
        TimeTable problem = generateProblem();
        TimeTable solution = timeTableController.solve(problem);
        assertFalse(solution.getLessonList().isEmpty());
        for (Lesson lesson : solution.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(solution.getScore().isFeasible());
    }

    private TimeTable generateProblem() {
        List<Timeslot> timeslotList = new ArrayList<>();
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30), LocalTime.of(9,
30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>();
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
        lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
        lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
```

```
        lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
        lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
        return new TimeTable(timeslotList, roomList, lessonList);
    }

}
```

此测试会验证在解决后，所有课程都分配给一个时间插槽和房间。它还会验证是否发现一种可行的解决方案（无硬限制）。

通常，该方案在 200 毫秒内找到可行的解决方案。请注意，@SpringBootTest 注释 的属性 如何覆盖 solver 终止，以便在可行的解决方案(0hard8:0:1::soft)时立即终止。这可避免硬编码代码，因为单元测试可能会在任意硬件上运行。这种方法可确保测试运行时间足够长，以找到可行的解决方案，即使在速度较慢的系统上也是如此。但是，即使在快速系统中，它不会比严格必须运行 milli 秒。

17.7.3. 日志记录

完成 Red Hat Build of OptaPlanner Spring Boot timetable 应用程序后，您可以使用日志信息微调 ConstraintProvider 中的限制。查看 info 日志文件中的分数计算速度，以评估对您约束的更改的影响。以调试模式运行应用程序，以显示应用程序所采用的每个步骤，或使用 trace logging 来记录每个步骤和每个移动。

流程

1. 运行 timetable 应用以固定时间，例如五分钟。

2. 查看日志文件中的分数计算速度，如下例所示：

```
... Solving ended: ..., score calculation speed (29455/sec), ...
```

3. 更改约束，再次运行 planning 应用程序以相同的时间，并检查日志文件中记录的分数计算速度。

4. 以 debug 模式运行应用程序以记录每个步骤：

   - 要从命令行运行调试模式，请使用 -D 系统属性。

- 要更改 application.properties 文件中的日志记录，请在该文件中添加以下行：

```
logging.level.org.optaplanner=debug
```

以下示例显示了 日志文件中的 debug 模式的输出：

```
... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode (REPRODUCIBLE), random (JDK with seed 0).
...     CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15), picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
...     CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15), picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
...
```

5. 使用 trace logging 显示每个步骤和每个步骤。

17.8. 添加数据库和 UI 集成

使用 Spring Boot 创建 Red Hat Build of OptaPlanner 应用程序示例后，添加数据库和 UI 集成。

前提条件

- 您已创建了 OptaPlanner Spring Boot timetable 示例。

流程

1. 为 Timeslot、Room 和 lesson 创建 Java Persistence API (zFCP)存储库。有关创建 JPA 存储库的详情，请参考 Spring 网站 使用 JPA 访问数据。

2. 通过 REST 公开 JPA 存储库。有关公开存储库的详情，请参考 Spring 网站上的 使用 REST 访问 JPA 数据。

3. 构建 TimeTableRepository facade，在单个事务中读取和写入 TimeTable。

4. *按照以下示例所示调整 TimeTableController ：*

```
package com.example.solver;

import com.example.domain.TimeTable;
import com.example.persistence.TimeTableRepository;
import org.optaplanner.core.api.score.ScoreManager;
import org.optaplanner.core.api.solver.SolverManager;
import org.optaplanner.core.api.solver.SolverStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/timeTable")
public class TimeTableController {

    @Autowired
    private TimeTableRepository timeTableRepository;
    @Autowired
    private SolverManager<TimeTable, Long> solverManager;
    @Autowired
    private ScoreManager<TimeTable> scoreManager;

    // To try, GET http://localhost:8080/timeTable
    @GetMapping()
    public TimeTable getTimeTable() {
        // Get the solver status before loading the solution
        // to avoid the race condition that the solver terminates between them
        SolverStatus solverStatus = getSolverStatus();
        TimeTable solution =
timeTableRepository.findById(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
        scoreManager.updateScore(solution); // Sets the score
        solution.setSolverStatus(solverStatus);
        return solution;
    }

    @PostMapping("/solve")
    public void solve() {

solverManager.solveAndListen(TimeTableRepository.SINGLETON_TIME_TABLE_ID,
                timeTableRepository::findById,
                timeTableRepository::save);
    }

    public SolverStatus getSolverStatus() {
        return
solverManager.getSolverStatus(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
    }

    @PostMapping("/stopSolving")
    public void stopSolving() {
```

```
solverManager.terminateEarly(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
    }

}
```

为了简单起见，此代码只处理一个 TimeTable 实例，但可以简单地启用多租户，并并行处理不同高中度的多个 TimeTable 实例。

getTimeTable （） 方法返回数据库的最新时间。它使用 ScoreManager （自动注入）来计算该可时间的分数，以便 UI 可以显示分数。

solve （） 方法启动一个作业，以解决当前的可时间表，并将时间插槽和房间分配存储在数据库中。它使用 SolverManager.solveAndListen （） 方法侦听中间的解决方案，并相应地更新数据库。这可让 UI 在后端仍然解决时显示进度。

5. 现在，solve （） 方法会立即返回，调整 TimeTableControllerTest，如下例所示：

```
package com.example.solver;

import com.example.domain.Lesson;
import com.example.domain.TimeTable;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.optaplanner.core.api.solver.SolverStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@SpringBootTest(properties = {
        "optaplanner.solver.termination.spent-limit=1h", // Effectively disable this
termination in favor of the best-score-limit
        "optaplanner.solver.termination.best-score-limit=0hard/*soft"})
public class TimeTableControllerTest {

    @Autowired
    private TimeTableController timeTableController;

    @Test
    @Timeout(600_000)
    public void solveDemoDataUntilFeasible() throws InterruptedException {
        timeTableController.solve();
        TimeTable timeTable = timeTableController.getTimeTable();
        while (timeTable.getSolverStatus() != SolverStatus.NOT_SOLVING) {
            // Quick polling (not a Test Thread Sleep anti-pattern)
```

```
            // Test is still fast on fast systems and doesn't randomly fail on slow systems.
            Thread.sleep(20L);
            timeTable = timeTableController.getTimeTable();
        }
        assertFalse(timeTable.getLessonList().isEmpty());
        for (Lesson lesson : timeTable.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(timeTable.getScore().isFeasible());
    }

}
```

6. 轮询最新的解决方案，直到解决者完成解决过程。

7. 要视觉化时间表，请在这些 REST 方法之上构建有吸引力的 Web UI。

17.9. 使用 MICROMETER 和 PROMETHEUS 监控您的 CENTRAL TIMETABLE OPTAPLANNER SPRING BOOT 应用程序

OptaPlanner 通过 Micrometer (Java 应用程序的指标检测库)公开指标。您可以将 Micrometer 与 Prometheus 搭配使用，以监控教育时应用程序中的 OptaPlanner solver。

先决条件

- 您已创建了 Spring Boot OptaPlanner school timetable 应用程序。
- 已安装 Prometheus。有关安装 Prometheus 的详情，请查看 Prometheus 网站。

流程

1. 导航到 technology/java-spring-boot 目录。

2. 将 Micrometer Prometheus 依赖项添加到 school timetable pom.xml 文件中：

```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

```
<dependency>
 <groupId>io.micrometer</groupId>
 <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

3. **在 application.properties 文件中添加以下属性：**

```
management.endpoints.web.exposure.include=metrics,prometheus
```

4. 启动 **amp timetable** 应用程序：

```
mvn spring-boot:run
```

5. 在 **Web** 浏览器中打开 **http://localhost:8080/actuator/prometheus**。

第 18 章 红帽构建的 OPTAPLANNER 和 JAVA：中等时间的快速入门指南

本指南指导您完成使用 OptaPlanner 约束创建简单 Java 应用程序的过程，用于解决人工智能(AI)。您将创建一个命令行应用程序，为教育人员和教员优化了院校的时间：

```
...
INFO  Solving ended: time spent (5000), best score (0hard/9soft), ...
INFO
INFO  |            | Room A     | Room B     | Room C     |
INFO  |------------|------------|------------|------------|
INFO  | MON 08:30  | English    | Math       |            |
INFO  |            | I. Jones   | A. Turing  |            |
INFO  |            | 9th grade  | 10th grade |            |
INFO  |------------|------------|------------|------------|
INFO  | MON 09:30  | History    | Physics    |            |
INFO  |            | I. Jones   | M. Curie   |            |
INFO  |            | 9th grade  | 10th grade |            |
INFO  |------------|------------|------------|------------|
INFO  | MON 10:30  | History    | Physics    |            |
INFO  |            | I. Jones   | M. Curie   |            |
INFO  |            | 10th grade | 9th grade  |            |
INFO  |------------|------------|------------|------------|
...
INFO  |------------|------------|------------|------------|
```

您的应用程序将使用 AI 自动将 lesson 实例分配给 Timeslot 和 Room 实例，例如：

- 房间最多可以有一门。
- 教员可以在大多数课程同时进行指导。
- 学员最多可同时参加一门课程。
- 教师更喜欢在同一房间展示所有课程。
- 教师希望检测后续课程和课程之间的差别。
- 同一主题上的学员不类似后续课程。

以数学方式说，中型时间稳定是一个 NP-hard 的问题。这意味着扩展比较困难。简单地简单地强制对所有可能的组合进行迭代，需要数百万年时间用于非公平的数据集，即使在超级计算机上也是如此。satisfy, AI 约束解析器（如 OptaPlanner）具有在合理的时间内提供接近最佳解决方案的高级算法。

先决条件

- 已安装 OpenJDK (JDK) 11。红帽构建的 Open JDK 可通过红帽客户门户网站中的 Software Downloads 页面（需要登录）。
- 已安装 Apache Maven 3.6 或更高版本。Maven 位于 Apache Maven Project 网站。
- IDE，如 IntelliJ IDEA、VSCode 或 Eclipse

18.1. 创建 MAVEN 或 GRADLE 构建文件并添加依赖项

您可以将 Maven 或 Gradle 用于 OptaPlanner school timetable 应用程序。创建构建文件后，添加以下依赖项：

- OptaPlanner-core （编译范围）以解决中等问题
- 对 JUnit 的 OptaPlanner -test （测试范围）测试 CLASS timetabling 约束
- logback-classic （运行时范围）等实现，以查看 OptaPlanner 采取的步骤

流程

1. 创建 Maven 或 Gradle 构建文件。
2. 在您的构建文件中添加 optaplanner-core、Optaplanner-test 和 logback-classic 依赖项：
   - 对于 Maven，将以下依赖项添加到 pom.xml 文件中：

```
<dependency>
  <groupId>org.optaplanner</groupId>
```

```
  <artifactId>optaplanner-core</artifactId>
 </dependency>

 <dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-test</artifactId>
  <scope>test</scope>
 </dependency>

 <dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.3</version>
 </dependency>
```

以下示例显示了完整的 pom.xml 文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>org.acme</groupId>
 <artifactId>optaplanner-hello-world-school-timetabling-quickstart</artifactId>
 <version>1.0-SNAPSHOT</version>

 <properties>
  <maven.compiler.release>11</maven.compiler.release>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

  <version.org.optaplanner>8.38.0.Final-redhat-00004</version.org.optaplanner>
  <version.org.logback>1.2.3</version.org.logback>

  <version.compiler.plugin>3.8.1</version.compiler.plugin>
  <version.surefire.plugin>3.0.0-M5</version.surefire.plugin>
  <version.exec.plugin>3.0.0</version.exec.plugin>
 </properties>

 <dependencyManagement>
  <dependencies>
   <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-bom</artifactId>
    <version>${version.org.optaplanner}</version>
    <type>pom</type>
    <scope>import</scope>
   </dependency>
   <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${version.org.logback}</version>
   </dependency>
```

```
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-core</artifactId>
  </dependency>
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <scope>runtime</scope>
  </dependency>

  <!-- Testing -->
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${version.compiler.plugin}</version>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${version.surefire.plugin}</version>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <version>${version.exec.plugin}</version>
      <configuration>
        <mainClass>org.acme.schooltimetabling.TimeTableApp</mainClass>
      </configuration>
    </plugin>
  </plugins>
</build>

<repositories>
  <repository>
    <id>jboss-public-repository-group</id>
    <url>https://repository.jboss.org/nexus/content/groups/public/</url>
    <releases>
      <!-- Get releases only from Maven Central which is faster. -->
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
```

```
    </repository>
  </repositories>
</project>
```

- *对于 Gradle，请在 gradle.build 文件中添加以下依赖项：*

```
dependencies {
    implementation platform("org.optaplanner:optaplanner-bom:${optaplannerVersion}")
    implementation "org.optaplanner:optaplanner-core"
    testImplementation "org.optaplanner:optaplanner-test"

    runtimeOnly "ch.qos.logback:logback-classic:${logbackVersion}"
}
```

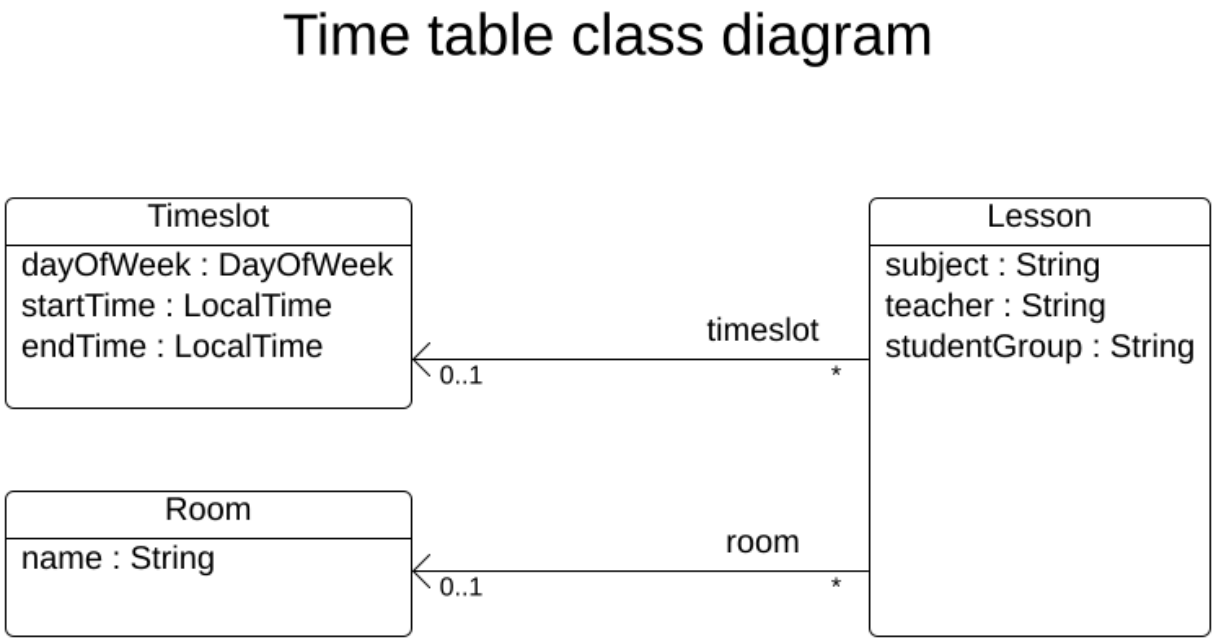
以下示例显示了已完成的 gradle.build 文件。

```
plugins {
    id "java"
    id "application"
}

def optaplannerVersion = "{optaplanner-version}"
def logbackVersion = "1.2.9"

group = "org.acme"
version = "1.0-SNAPSHOT"

repositories {
    mavenCentral()
}

dependencies {
    implementation platform("org.optaplanner:optaplanner-
bom:${optaplannerVersion}")
    implementation "org.optaplanner:optaplanner-core"
    testImplementation "org.optaplanner:optaplanner-test"

    runtimeOnly "ch.qos.logback:logback-classic:${logbackVersion}"
}

java {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}

compileJava {
    options.encoding = "UTF-8"
    options.compilerArgs << "-parameters"
}

compileTestJava {
```

```
    options.encoding = "UTF-8"
}

application {
    mainClass = "org.acme.schooltimetabling.TimeTableApp"
}

test {
    // Log the test execution results.
    testLogging {
        events "passed", "skipped", "failed"
    }
}
```

18.2. 对域对象建模

红帽构建的 OptaPlanner timetable 项目的目标是为每个课程分配时间插槽和房间。要做到这一点，请添加三个类，Timeslot , lesson , 和 Room，如下图所示：



timeslot

Timeslot 类代表了在课程时的间隔，例如 Monday 10:30 - 11:30 或 Tuesday 13:30 - 14:30。在这个示例中，所有时间插槽都有相同的持续时间，在 lunch 或其他间没有时间插槽。

时间段没有日期，因为高校计划每周都会重复。不需要 持续规划。一个 timeslot 被称为 问题，因为在解决过程中没有 Timeslot 实例改变。此类类不需要任何 OptaPlanner 特定注解。

房间

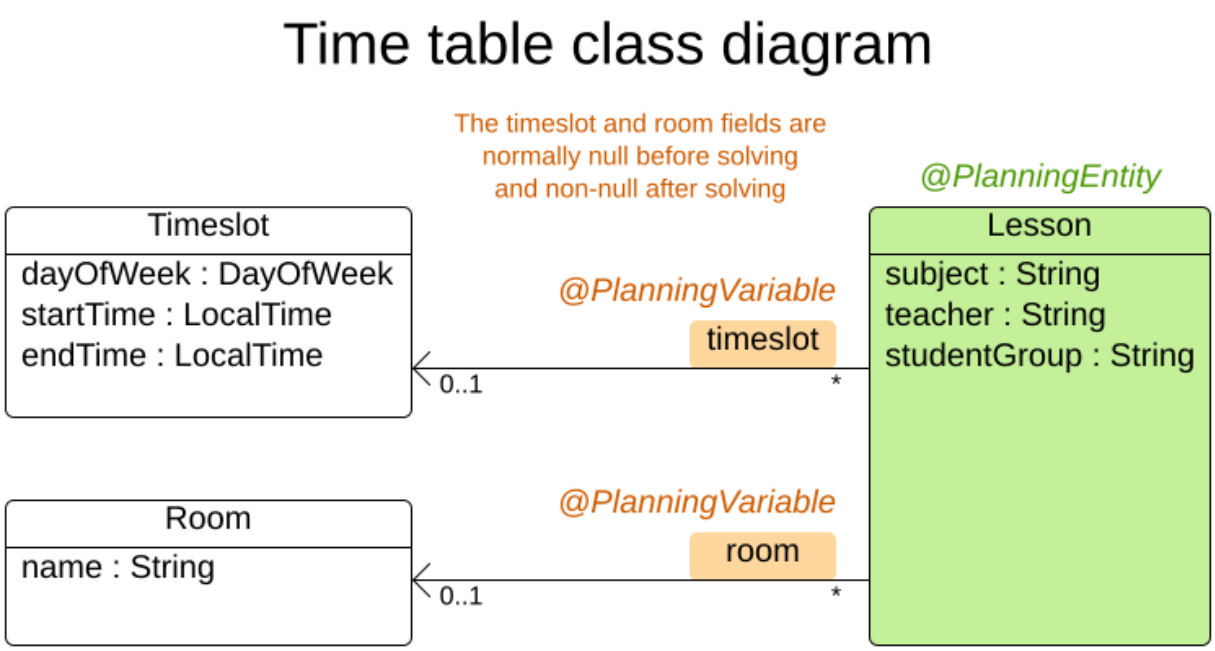
Room 类代表一个跟踪课程的位置，例如 Room A 或 Room B。在这个示例中，所有房间都没有容量限制，它们可以容纳所有课程。

房间 实例在解决时 不会改变，因此也 是一个问题。

lesson

在短时间内，由 Lesson 类表示，教员向一组学员提供主题，例如 Math by A.Turing for 9th grade 或 Chemistry by M.Curie for 10th grade。如果每周由同一生组多次学习一个主题，则有多个仅通过 id 区分的 lesson 实例。例如，第 9 个评级每周有 6 个数个。

在解决期间，OptaPlanner 更改了 Lesson 类的 timeslot 和 room 字段，为每个课程分配时间插槽和房间。因为 OptaPlanner 更改这些字段，所以 lesson 是一个 规划实体 ：



上图中的大部分字段包含输入数据，但 orange 字段除外。一个 lesson 的 timeslot 和 room 字段在输入数据中未分配（空），并在输出数据中分配（非 null）。OptaPlanner 在解决过程中更改这些字段。此类字段称为计划变量。为了让 OptaPlanner 识别它们，timeslot 和 room 字段都需要 @PlanningVariable 注释。它包含类 lesson 需要一个 @PlanningEntity 注释。

流程

1. *创建 src/main/java/com/example/domain/Timeslot.java 类：*

```
package com.example.domain;

import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;

    private Timeslot() {
    }

    public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
        this.dayOfWeek = dayOfWeek;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    @Override
    public String toString() {
        return dayOfWeek + " " + startTime.toString();
    }

    // ********************************
    // Getters and setters
    // ********************************

    public DayOfWeek getDayOfWeek() {
        return dayOfWeek;
    }

    public LocalTime getStartTime() {
        return startTime;
    }

    public LocalTime getEndTime() {
        return endTime;
    }

}
```

注意 toString （） 方法保持输出短，以便更轻松地阅读 OptaPlanner 的 DEBUG 或 TRACE 日志，如稍后所示。

2. *创建 src/main/java/com/example/domain/Room.java 类：*

```
package com.example.domain;

public class Room {
```

```
    private String name;

    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public String getName() {
        return name;
    }

}
```

3. 创建 src/main/java/com/example/domain/Lesson.java 类：

```
package com.example.domain;

import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
```

```
    }

    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // ********************************
    // Getters and setters
    // ********************************

    public Long getId() {
        return id;
    }

    public String getSubject() {
        return subject;
    }

    public String getTeacher() {
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {
        return timeslot;
    }

    public void setTimeslot(Timeslot timeslot) {
        this.timeslot = timeslot;
    }

    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
        this.room = room;
    }

}
```

Lesson 类具有一个 @PlanningEntity 注释，因此 OptaPlanner 知道此类在解决过程中发生了变化，因为它包含一个或多个计划变量。

timeslot 字段具有一个 @PlanningVariable 注释，因此 OptaPlanner 知道它可以更改其值。为了找到潜在的 Timeslot 实例来分配给此字段，OptaPlanner 使用 valueRangeProviderRefs 属性连接到提供 List<Timeslot> 的值范围供应商。有关值范围供应商的信息，请参阅 第 18.4 节 “在规划解决方案中收集域对象”。

room 字段还具有 @PlanningVariable 注释，原因相同。

18.3. 定义限制并计算分数

在解决问题时，分数 代表特定解决方案的质量。分数越高。红帽构建的 OptaPlanner 寻求最佳解决方案，这是在可用时间内获得最高分数的解决方案。这可能是 最佳解决方案。

因为 timetable 示例用例具有硬和软限制，所以请使用 HardSoftScore 类来代表分数：

- *硬约束不能被破坏。例如 ：一个空间最多可以同时有一个。*
- *软限制不应中断。例如 ：教员更倾向于在单一房间进行学习。*

硬约束会根据其他硬限制来加权。软限制是针对其他软限制的权重。硬限制始终超过软约束，无论其对应的权重是什么。

要计算分数，您可以实施 EasyScoreCalculator 类：

```
public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

   @Override
   public HardSoftScore calculateScore(TimeTable timeTable) {
      List<Lesson> lessonList = timeTable.getLessonList();
      int hardScore = 0;
      for (Lesson a : lessonList) {
         for (Lesson b : lessonList) {
            if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
                  && a.getId() < b.getId()) {
               // A room can accommodate at most one lesson at the same time.
               if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                  hardScore--;
               }
               // A teacher can teach at most one lesson at the same time.
               if (a.getTeacher().equals(b.getTeacher())) {
                  hardScore--;
               }
               // A student can attend at most one lesson at the same time.
               if (a.getStudentGroup().equals(b.getStudentGroup())) {
                  hardScore--;
               }
            }
```

```
            }
        }
        int softScore = 0;
        // Soft constraints are only implemented in the "complete" implementation
        return HardSoftScore.of(hardScore, softScore);
    }

}
```

不幸的是，这个解决方案无法很好地扩展，因为它并没有递增：每次将课程分配给不同的时间插槽或房间，所有课程都会被重新评估以计算新的分数。

更好的解决方法是创建一个 src/main/java/com/example/solver/TimeTableConstraintProvider.java 类，以执行增量分数计算。此类使用 OptaPlanner 的 ConstraintStream API，该 API 由 Java 8 Streams 和 SQL 推进。ConstraintProvider 比 EasyScoreCalculator 增加了一个比 EasyScoreCalculator 更强的 magnitude 顺序：O(n)而不是 O(nbusybox)。

流程

创建以下 src/main/java/com/example/solver/TimeTableConstraintProvider.java 类：

```
package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
                // Hard constraints
                roomConflict(constraintFactory),
                teacherConflict(constraintFactory),
                studentGroupConflict(constraintFactory),
                // Soft constraints are only implemented in the "complete" implementation
        };
    }

    private Constraint roomConflict(ConstraintFactory constraintFactory) {
        // A room can accommodate at most one lesson at the same time.

        // Select a lesson ...
        return constraintFactory.forEach(Lesson.class)
                // ... and pair it with another lesson ...
                .join(Lesson.class,
```

```
                // ... in the same timeslot ...
                Joiners.equal(Lesson::getTimeslot),
                // ... in the same room ...
                Joiners.equal(Lesson::getRoom),
                // ... and the pair is unique (different id, no reverse pairs)
                Joiners.lessThan(Lesson::getId))
        // then penalize each pair with a hard weight.
        .penalize(HardSoftScore.ONE_HARD)
        .asConstraint("Room conflict");
}

private Constraint teacherConflict(ConstraintFactory constraintFactory) {
    // A teacher can teach at most one lesson at the same time.
    return constraintFactory.forEach(Lesson.class)
            .join(Lesson.class,
                    Joiners.equal(Lesson::getTimeslot),
                    Joiners.equal(Lesson::getTeacher),
                    Joiners.lessThan(Lesson::getId))
                    .penalize(HardSoftScore.ONE_HARD)
                    .asConstraint("Teacher conflict");
}

private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
    // A student can attend at most one lesson at the same time.
    return constraintFactory.forEach(Lesson.class)
            .join(Lesson.class,
                    Joiners.equal(Lesson::getTimeslot),
                    Joiners.equal(Lesson::getStudentGroup),
                    Joiners.lessThan(Lesson::getId))
                    .penalize(HardSoftScore.ONE_HARD)
                    .asConstraint("Student group conflict");
}

}
```

18.4. 在规划解决方案中收集域对象

TimeTable 实例会打包单个 数据集 的所有 TimeTable、Room 和 lesson 实例。另外，因为它包含所有课程，每个课程都有特定的规划变量状态，所以它是一个 规划解决方案，它分数如下：

- 如果尚未分配课程，则它是一个 未初始化的 解决方案，例如，分数为 -4init/0hard/0soft 的解决方案。
- 如果它中断了硬限制，则它是一个 不可避免的 解决方案，例如分数为 -2hard/-3soft 的解决方案。
- 如果它遵循所有硬限制，则它是一个可行的解决方案，例如分数为 0hard/-7soft 的解决方案。

TimeTable 类具有 @PlanningSolution 注解，因此红帽构建的 OptaPlanner 知道此类包含所有输入和输出数据。

特别是，这个类是问题的输入：

- *包含所有时间插槽的 timeslotList 字段*
  - *这是问题事实列表，因为它们在解决过程中不会改变。*
- *带有所有房间的 roomList 字段*
  - *这是问题事实列表，因为它们在解决过程中不会改变。*
- *具有所有课程的 lessonList 字段*
  - *这是规划实体列表，因为它们在解决过程中发生了变化。*
  - *每个 lesson：*
    - *timeslot 和 room 字段的值通常仍然为 null，因此未分配。它们是规划变量。*
    - *其他字段（如 主题、sonr 和 studentGroup ）已填写。这些字段是问题属性。*

但是，这个类也是解决方案的输出：

- *在解决解决后，每个 Lesson 实例都有非null 时间slot 和 room 字段的 lessonList 字段*
- *代表输出解决方案的质量的 score 字段，例如 0hard/-5soft*

流程

创建 src/main/java/com/example/domain/TimeTable.java 类：

```
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;

@PlanningSolution
public class TimeTable {

    @ValueRangeProvider(id = "timeslotRange")
    @ProblemFactCollectionProperty
    private List<Timeslot> timeslotList;

    @ValueRangeProvider(id = "roomRange")
    @ProblemFactCollectionProperty
    private List<Room> roomList;

    @PlanningEntityCollectionProperty
    private List<Lesson> lessonList;

    @PlanningScore
    private HardSoftScore score;

    private TimeTable() {
    }

    public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
            List<Lesson> lessonList) {
        this.timeslotList = timeslotList;
        this.roomList = roomList;
        this.lessonList = lessonList;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public List<Timeslot> getTimeslotList() {
        return timeslotList;
    }

    public List<Room> getRoomList() {
        return roomList;
    }
```

```
    public List<Lesson> getLessonList() {
        return lessonList;
    }

    public HardSoftScore getScore() {
        return score;
    }

}
```

值范围供应商

timeslotList 字段是一个值范围供应商。它保存 OptaPlanner 可以从中选择的 Timeslot 实例，以分配给 lesson 实例的 timeslot 字段。timeslotList 字段具有一个 @ValueRangeProvider 注释，用于将 id 与 lesson 中 @PlanningVariable 的 @PlanningVariable 的 id 匹配。

遵循相同的逻辑后，roomList 字段也具有 @ValueRangeProvider 注释。

问题事实和规划实体属性

此外，OptaPlanner 需要知道它可以更改哪些更少的实例，以及如何检索用于您的 TimeTableConstraintProvider 分数计算的 Timeslot 和 Room 实例。

timeslotList 和 roomList 字段具有一个 @ProblemFactCollectionProperty 注释，因此您的 TimeTableConstraintProvider 可以从这些实例中选择。

lessonList 有一个 @PlanningEntityCollectionProperty 注释，因此 OptaPlanner 可以在解决期间更改它们，您的 TimeTableConstraintProvider 也可以从它们中选择。

18.5. THE TIMETABLEAPP.JAVA CLASS

在创建了 school timetable 应用程序的所有组件后，您将在 TimeTableApp.java 类中将它们全部放在 TimeTableApp.java 类中。

main （） 方法执行以下任务：

1. *创建 SolverFactory，以 为每个 数据集构建 Solver。*

2. *加载数据集。*

3. *通过 Solver .solve （）解决问题。*

4. *视觉化该数据集的解决方案。*

通常，应用程序有一个 SolverFactory 来为要 解决的 每个问题数据集构建一个新的 Solver 实例。SolverFactory 是 thread-safe，但 Solver 不是。对于 tutorial timetable 应用程序，只有一个数据集，因此只有一个 Solver 实例。

以下是完成的 TimeTableApp.java 类：

```
package org.acme.schooltimetabling;

import java.time.DayOfWeek;
import java.time.Duration;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import org.acme.schooltimetabling.domain.Lesson;
import org.acme.schooltimetabling.domain.Room;
import org.acme.schooltimetabling.domain.TimeTable;
import org.acme.schooltimetabling.domain.Timeslot;
import org.acme.schooltimetabling.solver.TimeTableConstraintProvider;
import org.optaplanner.core.api.solver.Solver;
import org.optaplanner.core.api.solver.SolverFactory;
import org.optaplanner.core.config.solver.SolverConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class TimeTableApp {

    private static final Logger LOGGER = LoggerFactory.getLogger(TimeTableApp.class);

    public static void main(String[] args) {
        SolverFactory<TimeTable> solverFactory = SolverFactory.create(new SolverConfig()
                .withSolutionClass(TimeTable.class)
                .withEntityClasses(Lesson.class)
                .withConstraintProviderClass(TimeTableConstraintProvider.class)
                // The solver runs only for 5 seconds on this small data set.
                // It's recommended to run for at least 5 minutes ("5m") otherwise.
                .withTerminationSpentLimit(Duration.ofSeconds(5)));
```

```
        // Load the problem
        TimeTable problem = generateDemoData();

        // Solve the problem
        Solver<TimeTable> solver = solverFactory.buildSolver();
        TimeTable solution = solver.solve(problem);

        // Visualize the solution
        printTimetable(solution);
    }

    public static TimeTable generateDemoData() {
        List<Timeslot> timeslotList = new ArrayList<>(10);
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30), LocalTime.of(9,
30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(8, 30), LocalTime.of(9,
30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>(3);
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        long id = 0;
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
        lessonList.add(new Lesson(id++, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(id++, "Biology", "C. Darwin", "9th grade"));
        lessonList.add(new Lesson(id++, "History", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
```

```
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Physics", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "French", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "Geography", "C. Darwin", "10th grade"));
        lessonList.add(new Lesson(id++, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(id++, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "10th grade"));

        return new TimeTable(timeslotList, roomList, lessonList);
    }

    private static void printTimetable(TimeTable timeTable) {
        LOGGER.info("");
        List<Room> roomList = timeTable.getRoomList();
        List<Lesson> lessonList = timeTable.getLessonList();
        Map<Timeslot, Map<Room, List<Lesson>>> lessonMap = lessonList.stream()
                .filter(lesson -> lesson.getTimeslot() != null && lesson.getRoom() != null)
                .collect(Collectors.groupingBy(Lesson::getTimeslot,
Collectors.groupingBy(Lesson::getRoom)));
        LOGGER.info("|            | " + roomList.stream()
                .map(room -> String.format("%-10s", room.getName())).collect(Collectors.joining(" |
")) + " |");
        LOGGER.info("|" + "------------|".repeat(roomList.size() + 1));
        for (Timeslot timeslot : timeTable.getTimeslotList()) {
            List<List<Lesson>> cellList = roomList.stream()
                    .map(room -> {
                        Map<Room, List<Lesson>> byRoomMap = lessonMap.get(timeslot);
                        if (byRoomMap == null) {
                            return Collections.<Lesson>emptyList();
                        }
                        List<Lesson> cellLessonList = byRoomMap.get(room);
                        if (cellLessonList == null) {
                            return Collections.<Lesson>emptyList();
                        }
                        return cellLessonList;
                    })
                    .collect(Collectors.toList());

            LOGGER.info("| " + String.format("%-10s",
                    timeslot.getDayOfWeek().toString().substring(0, 3) + " " + timeslot.getStartTime()) +
" | "
                    + cellList.stream().map(cellLessonList -> String.format("%-10s",
                            cellLessonList.stream().map(Lesson::getSubject).collect(Collectors.joining(",
"))))
                            .collect(Collectors.joining(" | "))
                    + " |");
            LOGGER.info("|            | "
                    + cellList.stream().map(cellLessonList -> String.format("%-10s",
                            cellLessonList.stream().map(Lesson::getTeacher).collect(Collectors.joining(",
"))))
                            .collect(Collectors.joining(" | "))
                    + " |");
            LOGGER.info("|            | "
                    + cellList.stream().map(cellLessonList -> String.format("%-10s",
```

```
                cellLessonList.stream().map(Lesson::getStudentGroup).collect(Collectors.joining(", ")))
                        .collect(Collectors.joining(" | "))
                    + " |");
            LOGGER.info("|" + "------------|".repeat(roomList.size() + 1));
        }
        List<Lesson> unassignedLessons = lessonList.stream()
                .filter(lesson -> lesson.getTimeslot() == null || lesson.getRoom() == null)
                .collect(Collectors.toList());
        if (!unassignedLessons.isEmpty()) {
            LOGGER.info("");
            LOGGER.info("Unassigned lessons");
            for (Lesson lesson : unassignedLessons) {
                LOGGER.info("  " + lesson.getSubject() + " - " + lesson.getTeacher() + " - " +
lesson.getStudentGroup());
            }
        }
    }

}
```

main （） 方法首先创建 SolverFactory ：

```
SolverFactory<TimeTable> solverFactory = SolverFactory.create(new SolverConfig()
        .withSolutionClass(TimeTable.class)
        .withEntityClasses(Lesson.class)
        .withConstraintProviderClass(TimeTableConstraintProvider.class)
        // The solver runs only for 5 seconds on this small data set.
        // It's recommended to run for at least 5 minutes ("5m") otherwise.
        .withTerminationSpentLimit(Duration.ofSeconds(5)));
```

SolverFactory 创建注册 @PlanningSolution 类、@PlanningEntity 类和 ConstraintProvider 类，您之前创建的所有类。

如果没有终止设置或 terminationEarly （） 事件，则 solver 会永久运行。为避免这种情况，解决者可将解决时间限制为 5 秒。

在 5 秒后，main （） 方法加载问题，解决问题，并打印解决方案：

```
        // Load the problem
        TimeTable problem = generateDemoData();

        // Solve the problem
        Solver<TimeTable> solver = solverFactory.buildSolver();
        TimeTable solution = solver.solve(problem);

        // Visualize the solution
        printTimetable(solution);
```

solve （） 方法不会立即返回。它在返回最佳解决方案前运行 5 秒。

OptaPlanner 返回可用终止时间里 的最佳解决方案。由于 NP-hard 问题的性质，最好的解决方案可能不是最佳的，特别是对于较大的数据集。增加终止时间，以有可能找到更好的解决方案。

generateDemoData （） 方法会产生一个可以解决的可量问题。

printTimetable （） 方法显示控制台的可时间性，因此可以轻松地确定其是否是好的时间表。

18.6. 创建并运行 PROFILE TIMETABLE 应用程序

现在，您已完成了 kutable Java 应用程序的所有组件，您已准备好将它们全部放在 TimeTableApp.java 类中并运行它。

先决条件

- *您已创建 amp timetable 应用程序所需的所有组件。*

流程

1. *创建 src/main/java/org/acme/acmetimetabling/TimeTableApp.java 类：*

```
package org.acme.schooltimetabling;

import java.time.DayOfWeek;
import java.time.Duration;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import org.acme.schooltimetabling.domain.Lesson;
import org.acme.schooltimetabling.domain.Room;
import org.acme.schooltimetabling.domain.TimeTable;
import org.acme.schooltimetabling.domain.Timeslot;
import org.acme.schooltimetabling.solver.TimeTableConstraintProvider;
import org.optaplanner.core.api.solver.Solver;
import org.optaplanner.core.api.solver.SolverFactory;
import org.optaplanner.core.config.solver.SolverConfig;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class TimeTableApp {

    private static final Logger LOGGER =
LoggerFactory.getLogger(TimeTableApp.class);

    public static void main(String[] args) {
        SolverFactory<TimeTable> solverFactory = SolverFactory.create(new
SolverConfig()
                .withSolutionClass(TimeTable.class)
                .withEntityClasses(Lesson.class)
                .withConstraintProviderClass(TimeTableConstraintProvider.class)
                // The solver runs only for 5 seconds on this small data set.
                // It's recommended to run for at least 5 minutes ("5m") otherwise.
                .withTerminationSpentLimit(Duration.ofSeconds(5)));

        // Load the problem
        TimeTable problem = generateDemoData();

        // Solve the problem
        Solver<TimeTable> solver = solverFactory.buildSolver();
        TimeTable solution = solver.solve(problem);

        // Visualize the solution
        printTimetable(solution);
    }

    public static TimeTable generateDemoData() {
        List<Timeslot> timeslotList = new ArrayList<>(10);
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30),
LocalTime.of(9, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(8, 30),
LocalTime.of(9, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>(3);
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
```

```
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        long id = 0;
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
        lessonList.add(new Lesson(id++, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(id++, "Biology", "C. Darwin", "9th grade"));
        lessonList.add(new Lesson(id++, "History", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Physics", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "French", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "Geography", "C. Darwin", "10th grade"));
        lessonList.add(new Lesson(id++, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(id++, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "10th grade"));

        return new TimeTable(timeslotList, roomList, lessonList);
    }

    private static void printTimetable(TimeTable timeTable) {
        LOGGER.info("");
        List<Room> roomList = timeTable.getRoomList();
        List<Lesson> lessonList = timeTable.getLessonList();
        Map<Timeslot, Map<Room, List<Lesson>>> lessonMap = lessonList.stream()
                .filter(lesson -> lesson.getTimeslot() != null && lesson.getRoom() != null)
                .collect(Collectors.groupingBy(Lesson::getTimeslot,
Collectors.groupingBy(Lesson::getRoom)));
        LOGGER.info("|            | " + roomList.stream()
                .map(room -> String.format("%-10s",
room.getName())).collect(Collectors.joining(" | ")) + " |");
        LOGGER.info("|" + "------------|".repeat(roomList.size() + 1));
        for (Timeslot timeslot : timeTable.getTimeslotList()) {
            List<List<Lesson>> cellList = roomList.stream()
                    .map(room -> {
                        Map<Room, List<Lesson>> byRoomMap = lessonMap.get(timeslot);
                        if (byRoomMap == null) {
                            return Collections.<Lesson>emptyList();
                        }
                        List<Lesson> cellLessonList = byRoomMap.get(room);
                        if (cellLessonList == null) {
                            return Collections.<Lesson>emptyList();
                        }
                        return cellLessonList;
                    })
                    .collect(Collectors.toList());
```

```
            LOGGER.info("| " + String.format("%-10s",
                    timeslot.getDayOfWeek().toString().substring(0, 3) + " " +
timeslot.getStartTime()) + " | "
                    + cellList.stream().map(cellLessonList -> String.format("%-10s",

cellLessonList.stream().map(Lesson::getSubject).collect(Collectors.joining(", "))))
                            .collect(Collectors.joining(" | "))
                    + " |");
            LOGGER.info("|            | "
                    + cellList.stream().map(cellLessonList -> String.format("%-10s",

cellLessonList.stream().map(Lesson::getTeacher).collect(Collectors.joining(", "))))
                            .collect(Collectors.joining(" | "))
                    + " |");
            LOGGER.info("|            | "
                    + cellList.stream().map(cellLessonList -> String.format("%-10s",

cellLessonList.stream().map(Lesson::getStudentGroup).collect(Collectors.joining(",
"))))
                            .collect(Collectors.joining(" | "))
                    + " |");
            LOGGER.info("|" + "------------|".repeat(roomList.size() + 1));
        }
        List<Lesson> unassignedLessons = lessonList.stream()
                .filter(lesson -> lesson.getTimeslot() == null || lesson.getRoom() == null)
                .collect(Collectors.toList());
        if (!unassignedLessons.isEmpty()) {
            LOGGER.info("");
            LOGGER.info("Unassigned lessons");
            for (Lesson lesson : unassignedLessons) {
                LOGGER.info("  " + lesson.getSubject() + " - " + lesson.getTeacher() + " - "
+ lesson.getStudentGroup());
            }
        }
    }

}
```

2. 将 TimeTableApp 类作为普通 Java 应用程序的主类运行。以下输出的结果：

```
...
INFO  |            | Room A     | Room B     | Room C     |
INFO  |------------|------------|------------|------------|
INFO  | MON 08:30  | English    | Math       |            |
INFO  |            | I. Jones   | A. Turing  |            |
INFO  |            | 9th grade  | 10th grade |            |
INFO  |------------|------------|------------|------------|
INFO  | MON 09:30  | History    | Physics    |            |
INFO  |            | I. Jones   | M. Curie   |            |
INFO  |            | 9th grade  | 10th grade |            |
...
```

3.

验证控制台输出。它是否符合所有硬限制？如果您在 TimeTableConstraintProvider 中注释掉 roomConflict 约束，会出现什么情况？

info 日志显示 OptaPlanner 在这 5 秒内执行的操作：

```
... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode (REPRODUCIBLE), random (JDK with seed 0).
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score calculation speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score calculation speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed (28524/sec), phase total (2), environment mode (REPRODUCIBLE).
```

18.7. 测试应用程序

良好的应用程序包括测试覆盖。测试您的 timetable 项目中的限制和 solver。

18.7.1. 测试 education 时间表限制

要以隔离方式测试 timetable 项目的每个约束，请在单元测试中使用 ConstraintVerifier。这会测试每个约束的基写情况与其他测试隔离，这可在添加新的约束时降低维护。

此测试会验证当在同一房间给出三个课程时，这个测试会验证 constraint TimeTableConstraintProvider::roomConflict，两个课程具有相同的时间slot，并用匹配权重 1 来节省。因此，如果约束 weight 为 10hard，它将分数减少 -10hard。

流程

创建 src/test/java/org/acme/optaplanner/solver/TimeTableConstraintProviderTest.java 类：

```
package org.acme.optaplanner.solver;

import java.time.DayOfWeek;
import java.time.LocalTime;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.Room;
import org.acme.optaplanner.domain.TimeTable;
import org.acme.optaplanner.domain.Timeslot;
```

```
import org.junit.jupiter.api.Test;
import org.optaplanner.test.api.score.stream.ConstraintVerifier;

@QuarkusTest
class TimeTableConstraintProviderTest {

    private static final Room ROOM = new Room("Room1");
    private static final Timeslot TIMESLOT1 = new Timeslot(DayOfWeek.MONDAY,
LocalTime.of(9,0), LocalTime.NOON);
    private static final Timeslot TIMESLOT2 = new Timeslot(DayOfWeek.TUESDAY,
LocalTime.of(9,0), LocalTime.NOON);

    @Inject
    ConstraintVerifier<TimeTableConstraintProvider, TimeTable> constraintVerifier;

    @Test
    void roomConflict() {
        Lesson firstLesson = new Lesson(1, "Subject1", "Teacher1", "Group1");
        Lesson conflictingLesson = new Lesson(2, "Subject2", "Teacher2", "Group2");
        Lesson nonConflictingLesson = new Lesson(3, "Subject3", "Teacher3", "Group3");

        firstLesson.setRoom(ROOM);
        firstLesson.setTimeslot(TIMESLOT1);

        conflictingLesson.setRoom(ROOM);
        conflictingLesson.setTimeslot(TIMESLOT1);

        nonConflictingLesson.setRoom(ROOM);
        nonConflictingLesson.setTimeslot(TIMESLOT2);

        constraintVerifier.verifyThat(TimeTableConstraintProvider::roomConflict)
                .given(firstLesson, conflictingLesson, nonConflictingLesson)
                .penalizesBy(1);
    }

}
```

请注意，ConstraintVerifier 在测试过程中如何忽略约束权重，即使这些约束权重在 ConstraintProvider 中被硬编码。这是因为在进入生产前定期更改约束权重。这样，约束 weight tweaking 不会破坏单元测试。

18.7.2. 测试 Central timetable solver

本例在红帽构建的 Quarkus 平台上测试红帽构建的 OptaPlanner education timetable 项目。它使用 JUnit 测试来生成测试数据集并将其发送到 TimeTableController 以解决。

流程

1. 使用以下内容创建 src/test/java/com/example/rest/TimeTableResourceTest.java 类：

-

```
package com.exmaple.optaplanner.rest;

import java.time.DayOfWeek;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import com.exmaple.optaplanner.domain.Room;
import com.exmaple.optaplanner.domain.Timeslot;
import com.exmaple.optaplanner.domain.Lesson;
import com.exmaple.optaplanner.domain.TimeTable;
import com.exmaple.optaplanner.rest.TimeTableResource;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@QuarkusTest
public class TimeTableResourceTest {

    @Inject
    TimeTableResource timeTableResource;

    @Test
    @Timeout(600_000)
    public void solve() {
        TimeTable problem = generateProblem();
        TimeTable solution = timeTableResource.solve(problem);
        assertFalse(solution.getLessonList().isEmpty());
        for (Lesson lesson : solution.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(solution.getScore().isFeasible());
    }

    private TimeTable generateProblem() {
        List<Timeslot> timeslotList = new ArrayList<>();
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30),
LocalTime.of(9, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>();
        roomList.add(new Room("Room A"));
```

```
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
        lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
        lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
        lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
        return new TimeTable(timeslotList, roomList, lessonList);
    }

}
```

此测试会验证在解决后，所有课程都分配给一个时间插槽和房间。它还会验证是否发现一种可行的解决方案（无硬限制）。

2. 在 src/main/resources/application.properties 文件中添加测试属性：

```
# The solver runs only for 5 seconds to avoid a HTTP timeout in this simple implementation.
# It's recommended to run for at least 5 minutes ("5m") otherwise.
quarkus.optaplanner.solver.termination.spent-limit=5s

# Effectively disable this termination in favor of the best-score-limit
%test.quarkus.optaplanner.solver.termination.spent-limit=1h
%test.quarkus.optaplanner.solver.termination.best-score-limit=0hard/*soft
```

通常，该方案在 200 毫秒内找到可行的解决方案。注意 application.properties 文件在测试过程中如何覆盖 solver 终止，以便在找到可行的解决方案 (0hardAttrsoft) 时立即终止。这可避免硬编码代码，因为单元测试可能会在任意硬件上运行。这种方法可确保测试运行时间足够长，以找到可行的解决方案，即使在速度较慢的系统上也是如此。但是，在快速系统中，它不会运行比严格必须长的 millisecond。

18.8. 日志记录

完成 Red Hat Build of OptaPlanner education timetable 项目后，您可以使用日志信息微调 ConstraintProvider 中的限制。查看 info 日志文件中的分数计算速度，以评估对您约束的更改的影响。以调试模式运行应用程序，以显示应用程序所采用的每个步骤，或使用 trace logging 来记录每个步骤和每个移动。

流程

1. 以固定时间（例如 5 分钟）运行 amp time。

2. 查看日志文件中的分数计算速度，如下例所示：

```
... Solving ended: ..., score calculation speed (29455/sec), ...
```

3. 更改约束，再次运行 planning 应用程序以相同的时间，并检查日志文件中记录的分数计算速度。

4. 以 debug 模式运行应用程序以记录应用程序所做的每个步骤：

   - 要从命令行运行调试模式，请使用 -D 系统属性。

   - 要永久启用 debug 模式，请在 application.properties 文件中添加以下行：

```
quarkus.log.category."org.optaplanner".level=debug
```

   以下示例显示了日志文件中的 debug 模式的输出：

```
... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
...     CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15),
picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
...     CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15),
picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
...
```

5. 使用 trace logging 显示每个步骤和每个步骤。

18.9. 使用 MICROMETER 和 PROMETHEUS 来监控您的中级时间 OPTAPLANNER JAVA 应用程序

OptaPlanner 通过 Micrometer (Java 应用程序的指标检测库)公开指标。您可以将 Micrometer 与 Prometheus 搭配使用，以监控教育时应用程序中的 OptaPlanner solver。

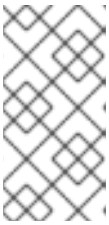
先决条件

- 您已创建了带有 Java 的 OptaPlanner school timetable 应用程序。
- 已安装 Prometheus。有关安装 Prometheus 的详情，请查看 Prometheus 网站。

流程

1. 将 Micrometer Prometheus 依赖项添加到 school timetable pom.xml 文件中，其中 <MICROMETER_VERSION > 是您安装的 Micrometer 的版本：

```
<dependency>
 <groupId>io.micrometer</groupId>
 <artifactId>micrometer-registry-prometheus</artifactId>
 <version><MICROMETER_VERSION></version>
</dependency>
```



注意

还需要 micrometer-core 依赖项。但是，这个依赖项包含在 optaplanner-core 依赖项中，因此您不需要将其添加到 pom.xml 文件中。

2. 将以下导入语句添加到 TimeTableApp.java 类中。

```
import io.micrometer.core.instrument.Metrics;
import io.micrometer.prometheus.PrometheusConfig;
import io.micrometer.prometheus.PrometheusMeterRegistry;
```

3. 将以下行添加到 TimeTableApp.java 类的主要方法的顶部，以便 Prometheus 可以在解决方案启动前从 com.sun.net.httpserver.HttpServer 中 scrap 数据：

```
PrometheusMeterRegistry prometheusRegistry = new
PrometheusMeterRegistry(PrometheusConfig.DEFAULT);

    try {
        HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
        server.createContext("/prometheus", httpExchange -> {
            String response = prometheusRegistry.scrape();
            httpExchange.sendResponseHeaders(200, response.getBytes().length);
            try (OutputStream os = httpExchange.getResponseBody()) {
                os.write(response.getBytes());
            }
        });
```

```
            new Thread(server::start).start();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        Metrics.addRegistry(prometheusRegistry);

        solve();
    }
```

4. 添加下面这行来控制解决时间。通过调整解决时间，您可以根据需要解决的时间来查看指标变化。

```
withTerminationSpentLimit(Duration.ofMinutes(5)));
```

5. 启动 **tutorial** 时间表应用程序。

6. 在 **Web** 浏览器中打开 **http://localhost:8080/prometheus**，在 **Prometheus** 中查看时间表应用程序。

7. 打开您的监控系统，以查看您的 **OptaPlanner** 项目的指标。

   公开以下指标：

   - **optaplanner_solver_errors_total** ：自测量开始以来发生的错误总数。
   - **optaplanner_solver_solve_duration_seconds_active_count** ：当前解决者的数量。
   - **optaplanner_solver_solve_duration_seconds_max:** 最长运行的当前活跃解析器的运行时。
   - **optaplanner_solver_solve_duration_seconds_duration_sum** ：每个活跃解析器的 **solve** 持续时间的总和。例如，如果有两个活跃的解决问题，则运行三分钟，另一个在一分钟内，总解决时间为四分钟。

附录 A. 版本信息

文档最后于 2023 年 7 月 14 日星期五更新。