Red Hat Build of OptaPlanner 8.38

Red Hat build of OptaPlanner を使用したソルバーの開発

Last Updated: 2023-07-25

Red Hat Build of OptaPlanner 8.38 Red Hat build of OptaPlanner を使用したソルバーの開発

法律上の通知

概要

このドキュメントでは、Red Hat build of OptaPlanner を使用してソルバーを開発し、計画の問題に対する最適なソリューションを見つける方法について説明します。

目次

はじめに

Red Hat build of OptaPlanner を使用して、計画問題に対する最適解を決定するソルバーを開発できます。OptaPlanner は、Red Hat build of OptaPlanner の組み込みコンポーネントです。Red Hat build of OptaPlanner のサービスの一部としてソルバーを使用して、特定の制約のある限られたリソースを最適化できます。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みにより、これらの変更は今後の複数のリリースに対して段階的に実施されます。詳細は、**Red Hat CTO である Chris Wright のメッセージ** をご覧ください。

パート I. RED HAT BUILD OF OPTAPLANNER 8.38 のリリースノート

これらのリリースノートでは、Red Hat build of OptaPlanner 8.38 の新機能を表示し、アップグレード手順が記載されています。

第1章 OPTAPLANNER 8.13 から RED HAT BUILD OF OPTAPLANNER 8.38 へのアップグレード

OptaPlanner 8.13 から Red Hat build of OptaPlanner 8.38 にアップグレードするには、OptaPlanner の以前のバージョンを次の順序でマージします。

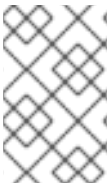
- 8.13.0.Final から 8.14.0.Final へ
- 8.19.0.Final から 8.20.0.Final へ
- 8.22.0.Final から 8.23.0.Final へ
- 8.27.0.Final から 8.28.0.Final へ
- 8.28.0.Final から 8.29.0.Final へ
- 8.31.0.Final から 8.32.0.Final へ
- 8.33.0.Final から 8.34.0.Final へ
- 8.36.0.Final から 8.37.0.Final へ

手順

1. ブラウザーで OptaPlanner Upgrade Recipe 8 ページを開きます。
2. アップグレードする最初のバージョンの手順を完了します (たとえば **8.13.0.Final から 8.14.0.Final**)。
3. 8.37.0.Final にアップグレードされるまで、手順を繰り返します。

第2章 RED HAT BUILD OF OPTAPLANNER 8.38 の新機能

このセクションでは、Red Hat build of OptaPlanner 8.38 の新機能について説明します。



注記

Bavet は、高速スコア計算に使用される機能です。Bavet は現在、OptaPlanner のコミュニティーバージョンでのみ利用できます。Red Hat build of OptaPlanner 8.38 では使用できません。

2.1. ピラーの移動と周辺の選択のパフォーマンスの向上

OptaPlanner は、複数のピラー移動セレクターが事前計算されたピラーキャッシュを共有し、移動セレクターごとにピラーキャッシュを再計算する代わりにそれを再利用できる状況を自動検出できるようになりました。**PillarChangeMove** と **PillarSwapMove など** のピラーの移動を組み合わせると、パフォーマンスが大幅に向上するはずです。

これは、周辺の選択範囲を使用する場合にも当てはまります。OptaPlanner は、事前計算された距離行列が複数の移動セレクター間で共有できる状況を自動検出できるようになり、メモリーと CPU 処理時間を節約できます。

この機能強化の結果、次のインターフェイスの実装はステートレスになることが期待されます。

- **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**

一般に、ソルバー設定がユーザーにインターフェイスの実装を要求する場合、その実装はステートレスであるか、外部状態を含めようとしないことが期待されます。これらのパフォーマンスの向上により、ソルバーが適切と判断した場合にこれらのインスタンスを再利用するようになるため、この要件に従わない場合、微妙なバグやスコアの破損が発生します。

2.2. OPTAPLANNER 設定の改善

EntitySelectorConfig や **ValueSelectorConfig** などのさまざまな設定クラスには、XML ベースのソルバー設定を自然な Java コードに置き換えることを容易にする新しいビルダーメソッドが含まれています。

2.3. K-OPT 移動の PLANNINGLISTVARIABLE サポート

リスト変数の新しい移動セレクター **KOptListMoveSelector** が追加されました。**KOptListMoveSelector** は、単一のエンティティーを選択し、そのルートから **k 個の** エッジを削除し、削除されたエッジのエンドポイントから **k 個の** 新しいエッジを追加します。**KOptListMoveSelector** は、ソルバーが配送経路の問題でローカルオプティマを回避するのに役立ちます。

2.4. SOLUTIONMANAGER によるシャドウ変数の更新のサポート

Explain (solution) や **update (solution)** などの **SolutionManager** (以前の **ScoreManager**) メソッドは、追加の引数 **SolutionUpdatePolicy** を持つ新しいオーバーロードを受け取りました。これは、ソリューションを永続ストレージ (リレーショナルデータベースなど) からロードするユーザーにとって便利です。これらのソリューションには、シャドウ変数やスコアによって運ばれる情報が含まれません。これらの新しいオーバーロードを呼び出して適切なポリシーを選択することにより、OptaPlanner はソリューション内のすべてのシャドウ変数の値を自動的に計算するか、スコアを再計算するか、あるいはその両方を行います。

同様に、**ProblemChangeDirector** は **updateShadowVariables()** と呼ばれる新しいメソッドを受け取りました。これにより、リアルタイム計画でオンデマンドでシャドウ変数を更新できるようになります。

2.5. 値範囲の自動検出

ほとんどの場合、計画変数と値の範囲の間のリンクを自動的に検出できるようになりました。したがって、**@ValueRangeProvider** は ID プロパティーを提供する必要がなくなりました。同様に、プランニング変数は、**valueRangeProviderRefs** プロパティーを介して値の範囲プロバイダーを参照する必要がなくなりました。

コードや設定変更は必要ありません。簡潔さよりも明確さを好むユーザーは、引き続き値範囲プロバイダーを明示的に参照できます。

パート II. RED HAT BUILD OF OPTAPLANNER のスタートガイド

ビジネスルールの開発者は、Red Hat build of OptaPlanner を使用して、限られたリソースや個別の制約の中で計画問題に対する最適解を見つけ出すことができます。

本書を使用して、Red Hat build of OptaPlanner で Solver の開発を開始していきます。

第3章 RED HAT BUILD OF OPTAPLANNER の概要

OptaPlanner は組み込み可能な軽量プランニングエンジンで、プランニングの問題を最適化します。最適化のためのヒューリスティック法およびメタヒューリスティック法を、非常に効率的なスコア計算と組み合わせ、一般的な Java プログラマーが計画問題を効率的に解決できるようにします。

たとえば、OptaPlanner は、さまざまなユースケースの解決に役立ちます。

- **従業員勤務表/患者一覧**: 看護師の勤務シフト作成を容易にし、病床管理を追跡します。
- **教育機関の時間割**: 授業、コース、試験、および会議の計画を容易にします。
- **工場の計画**: 自動車の組み立てライン、機械の操業計画、および作業員のタスク計画を追跡します。
- **在庫の削減**: 紙や金属などの資源の消費を減らし、無駄を最小限に抑えます。

どの組織も、制約のある限定されたリソース (従業員、資産、時間、および資金) を使用して製品やサービスを提供するといった計画問題に直面しています。

OptaPlanner は、Apache Software License 2.0 を使用するオープンソースソフトウェアです。100% Pure Java に認定されており、ほとんどの Java 仮想マシン (JVM) で稼働します。

3.1. BACKWARDS COMPATIBILITY

OptaPlanner は API と実装を分離します。

- **パブリック API**: パッケージ名前空間 **org.optaplanner.core.api**、**org.optaplanner.benchmark.api**、**org.optaplanner.test.api** および **org.optaplanner.persistence.api** 内のすべてのクラスは、今後のマイナーおよびパッチリリースで 100% 下位互換性があります。まれに、メジャーバージョン番号が変更されると、いくつかの特定のクラスに下位互換性のない変更がいくつか含まれることがありますが、そのような変更は アップグレードレシピ に明確に文書化されます。
- **XML 設定**: XML ソルバー設定は、非パブリック API クラスの使用を必要とする要素以外の全要素に対して下位互換性があります。XML Solver 設定は、パッケージ名前空間 **org.optaplanner.core.config** および **org.optaplanner.benchmark.config** のクラスによって定義されます。
- **実装クラス**: その他のクラスはすべて後方互換性が **ありません**。これらは将来のメジャーリリースまたはマイナーリリースで変更される予定です。アップグレードレシピ では、関連する変更点、新しいバージョンへのアップグレードするときにそのような変更に対応する方法を説明します。

3.2. 計画問題

計画問題 では、限られたリソースや個別の制約の中で最適なゴールを見つけ出します。最適なゴールは、次のようなさまざまなものです。

- 最大の利益: 最適なゴールにより、可能な限り高い利益が得られます。
- 経済活動の最小フットプリント: 最適なゴールでは、環境負荷が最小となります。
- スタッフ/顧客の最大満足: 最適なゴールでは、スタッフ/顧客のニーズが優先されます。

これらのゴールに到達できるかどうかは、利用できるリソースの数に依存します。たとえば、以下のようなリソースには制限があります。

- ユーザー数
- 時間
- 予算
- 装置、車両、コンピューター、施設などの物理資産

これらのリソースに関連する個別の制約についても考慮する必要があります。たとえば、要員が働くことのできる時間数、特定の装置を使用することのできる技能、または機器同士の互換性などです。

Red Hat build of OptaPlanner は、Java プログラマーが制約の飽和性の問題を効率的に解決するのに役立ちます。最適化ヒューリスティックとメタヒューリスティックを効率的なスコア計算と組み合わせます。

3.3. 計画問題での NP 完全

例に挙げたユースケースは **通常** NP 完全または NP 困難 であり、以下のことが言えます。

- 問題に対する解を実用的な時間内に検証することが容易です。
- 問題に対する最適解を実用的な時間内に見つけ出す確実な方法がない。

この場合、一般的な 2 つの手法では不十分であるため、問題を解くのが予想より困難だと考えられます。

- 力まかせアルゴリズムでは (より高度な類似アルゴリズムであっても)、時間がかかり過ぎる。
- たとえば ビンパッキング問題の ような迅速なアルゴリズムでは、**容量の大きい順でアイテムを入力する**と、最適とはほど遠い解が返されます。

高度な最適化アルゴリズムを用いる OptaPlanner であれば、このような計画問題に対する適切な解を、妥当な時間内に見つけ出すことができます。

3.4. 計画問題に対する解

計画問題には、多数の解が存在します。

以下に示すように、解は複数のカテゴリーに分類されます。

可能解

可能解とは、制約に違反するかどうかは問わず、あらゆる解を指します。通常、計画問題には膨大な数の可能解が存在します。ただし、このような解の多くは、役に立ちません。

実行可能解

実行可能解とは、いずれの (負の) ハード制約にも違反しない解を指します。実行可能解の数は、可能解の数に比例します。実行可能解が存在しないケースもあります。実行可能解は、可能解の部分集合です。

最適解

最適解とは、最高スコアの解を指します。通常、計画問題には数個の最適解が存在します。実行可能解が存在せず、最適解が現実的ではない場合でも、計画問題には少なくとも 1 つの最適解が必ず存在します。

見つかった最善解

最善解とは、指定された時間内に実施した検索で見つかった最高スコアの解を指します。通常、見つかった最善解は現実的で、十分な時間があれば最適解を見つけることができます。

直観には反していますが、小規模なデータセットの場合であっても、(正しく計算された場合は) 膨大な数の可能解が存在します。

optaplanner-examples/src 配布フォルダーで提供される例では、ほとんどのインスタンスに多数の可能解が存在します。最適解を確実に見つけることができる方法は存在しないため、いかなる実行方法も、これらすべての可能解の部分集合を評価することしかできません。

膨大な数の可能解全体を効率的に網羅するために、OptaPlanner はさまざまな最適化アルゴリズムをサポートしています。

ユースケースによっては、ある最適化アルゴリズムが他のアルゴリズムより勝ることがありますが、それを事前に予測することは不可能です。OptaPlanner では、XML またはコード中の Solver 設定を数行変更するだけで、最適化アルゴリズムを切り替えることができます。

3.5. 計画問題に対する制約

通常、プランニングの問題には、少なくとも 2 つの制約レベルがあります。

- **(負の) ハード制約** は、絶対に違反してはならない。
  例: 1 人の教師は同時に 2 つの講義を受け持つことはできない。

- **(負の) ソフト制約** は、避けることが可能であれば違反してはならない。
  例: 教師 A は金曜日の午後に講義を受け持ちたくない。

正の制約を持つ問題もあります。

- **正のソフト制約 (ボーナス)** は、可能であれば満たす必要がある。
  例: 教師 B は月曜日の午前中に講義を受け持つことを希望している。

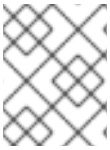
一部の基本的な問題にはハード制約のみがあります。問題によっては、3 つ以上の制約があります (例: ハード制約、中程度の制約、ソフト制約)。

これらの制約により、計画問題における **スコア計算方法** (または **適合度関数**) が定義されます。プランニングの問題の解は、それぞれスコアで等級付けすることができます。OptaPlanner のスコア制約は、Java などのオブジェクト指向言語または Drools ルールで記述されます。

このタイプのコードは柔軟で、スケーラビリティーに優れます。

3.6. RED HAT BUILD OF OPTAPLANNER で提供される例

Red Hat build of OptaPlanner には、OptaPlanner のサンプルが複数同梱されています。たとえばコードなどを確認して、ニーズに合ったものに変更できます。



注記

Red Hat は、Red Hat build of OptaPlanner ディストリビューションに含まれるコードサンプルのサポートはしていません。

OptaPlanner サンプルには、教育関連のコンテストで出題された問題を解決するものもあります。以下の表の **Contest** 列には、このようなコンテストが掲載されています。また、コンテストの目的とし

て、**現実的** か、**非現実的** かの識別をしています。**現実的なコンテスト** とは、以下の基準を満たす、独立した公式コンテストを指します。

- 明確に定義された実際のユースケースであること
- 実際に制約があること
- 実際のデータセットが複数あること
- 特定のハードウェアで特定の時間内に結果を再現できること
- 教育機関および/または企業の運用研究コミュニティーが真剣に参加していること

現実的なコンテストでは、競合のソフトウェアや教育研究と OptaPlanner を客観的に比較できます。

表3.1 サンプルの概要

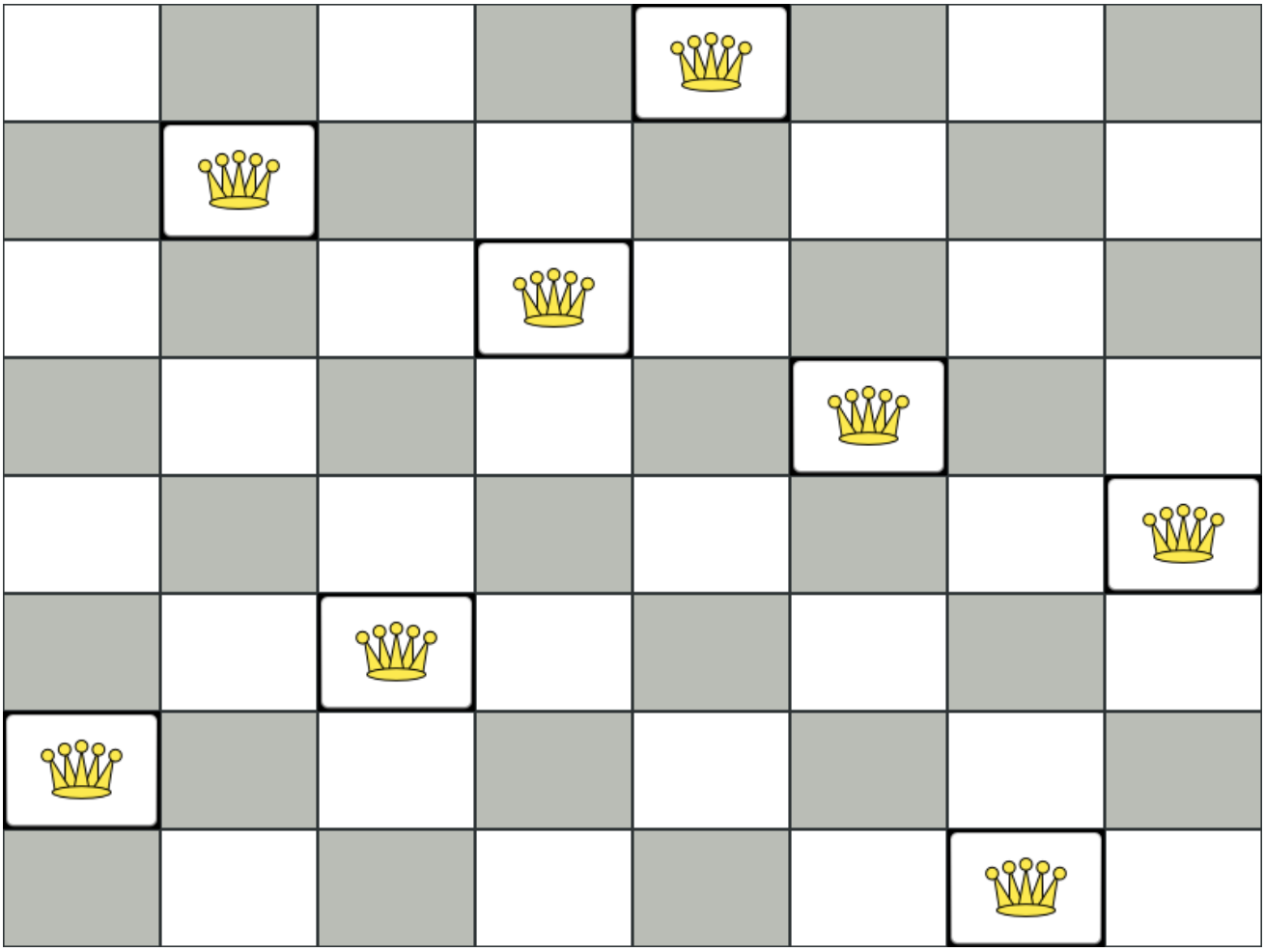
例	ドメイン	サイズ	コンテスト	ディレクトリー名
N クィーン	エンティティークラス 1 つ (変数 1 つ)	エンティティー ⇐ **256** 値 ⇐ **256** 探索空間 ⇐ **10^616**	無意味 (不正が可能)	**nqueens**
クラウドバランシング	エンティティークラス 1 つ (変数 1 つ)	エンティティー ⇐ **2400** 値 ⇐ **800** 探索空間 ⇐ **10^6967**	いいえ (弊社が定義)	**cloudbalancing**
巡回セールスマン	エンティティークラス 1 つ (連鎖変数 1 つ)	エンティティー ⇐ **980** 値 ⇐ **980** 探索空間 ⇐ **10^2504**	現実的でない TSP Web	**tsp**
テニスクラブのスケジュール	エンティティークラス 1 つ (変数 1 つ)	エンティティー ⇐ **72** 値 ⇐ **7** 探索空間 ⇐ **10^60**	いいえ (弊社が定義)	**tennis**
会議のスケジュール	エンティティークラス 1 つ (変数 2 つ)	エンティティー ⇐ **10** 値 ⇐ **320** および ⇐**5** 探索空間 ⇐ **10^320**	いいえ (弊社が定義)	**meetingscheduling**
コースの時間割	エンティティークラス 1 つ (変数 2 つ)	エンティティー ⇐ **434** 値 ⇐ **25** および ⇐**20** 探索空間 ⇐ **10^1171**	現実的 ITC 2007 track 3	**curriculumCourse**

例	ドメイン	サイズ	コンテスト	ディレクトリー名
マシンの再割当て	エンティティークラス 1 つ (変数 1 つ)	エンティティー ⇐ **50000** 値 ⇐ **5000** 探索空間 ⇐ **10^184948**	ほぼ現実的 ROADEF 2012	**machineReassignment**
配送経路	エンティティークラス 1 つ (連鎖変数 1 つ) シャドウエンティティークラス 1 つ (自動シャドウ変数 1 つ)	エンティティー ⇐ **55** 値 ⇐ **2750** 探索空間 ⇐ **10^8380**	現実的でない VRP Web	**vehiclerouting**
時間枠がある中での配送経路	配送経路すべて (シャドウ変数 1 つ)	エンティティー ⇐ **55** 値 ⇐ **2750** 探索空間 ⇐ **10^8380**	現実的でない VRP Web	**vehiclerouting**
プロジェクトジョブのスケジュール	エンティティークラス 1 つ (変数 2 つ) (シャドウ変数 1 つ)	エンティティー ⇐ **640** 値 ⇐ **?** および ⇐ **?** 探索空間 ⇐ **?**	ほぼ現実的 MISTA 2013	**projectjobscheduling**
タスクの割り当て	エンティティークラス 1 つ (リスト変数 1つ) シャドウエンティティークラス 1 つ (自動シャドウ変数 1 つ) (シャドウ変数 1 つ)	エンティティー ⇐ **20** 値 ⇐ **500** 探索空間 ⇐ **10^1168**	いいえ (弊社が定義)	**taskassigning**
試験の時間割	エンティティークラス 2 つ (同じ階層) (変数 2 つ)	エンティティー ⇐ **1096** 値 ⇐ **80** および ⇐ **49** 探索空間 ⇐ **10^3374**	現実的 ITC 2007 track 1	**examination**

例	ドメイン	サイズ	コンテスト	ディレクトリー名
看護師の勤務表	エンティティークラス1つ (変数1つ)	エンティティー ⇐ **752** 値 ⇐ **50** 探索空間 ⇐ **10^1277**	現実的 INRC 2010	**nurserostering**
巡回トーナメント	エンティティークラス1つ (変数1つ)	エンティティー ⇐ **1560** 値 ⇐ **78** 探索空間 ⇐ **10^2301**	現実的でない TTP	**travelingtournament**
会議スケジュール	エンティティークラス1つ (変数2つ)	エンティティー ⇐ **216** 値 ⇐ **18** および ⇐**20** 探索空間 ⇐ **10^552**	いいえ (弊社が定義)	**conferencescheduling**
航空機乗組員のスケジューリング	エンティティークラス1つ (変数1つ) シャドウエンティティークラス1つ (自動シャドウ変数1つ)	エンティティー ⇐ **4375** 値 ⇐ **750** 探索空間 ⇐ **10^12578**	いいえ (弊社が定義)	**flightcrewscheduling**

3.7. N クィーン

n サイズのチェスボードに、他のクィーンに取られないクィーンに **n** 個のクィーンを置きます。最も一般的な **n** クィーンパズルは、**n = 8** の 8 個のクィーンパズルです。



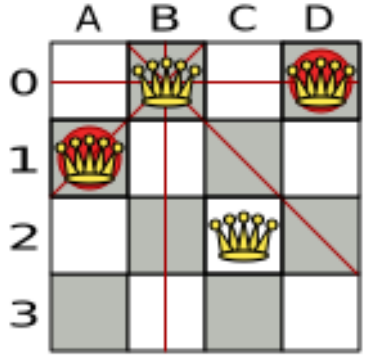
制約:

- n 列および n 行のチェスボードを使用します。
- チェスボードに n 個のクィーンを置きます。
- クィーンが他のクィーンに取られないように配置します。クィーンは、同じ水平線上、垂直線上、対角線上にある他のクィーンを取ることができます。

本書では、４つのクイーンパズルを主な例として多用しています。

以下が提案された解です。

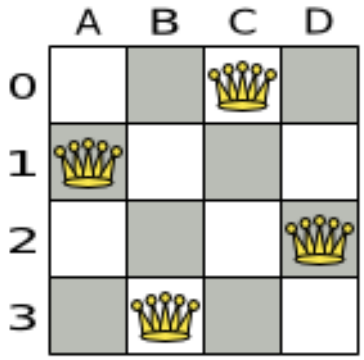
図3.1 4 個のクィーンパズルの誤った解



上記の解は、**A1** と **B0** (および **B0** と **D0**) のクィーンがお互いに駒を取れるので間違っています。**B0** のクィーンをどかせば他のクィーンに取られないようにするという制約は順守できますが、**n** 個のクィーンを置くという制約に違反します。

以下は正しい解です。

図3.2 クィーン 4 個のパズルの正しい解



すべての制約が満たされているので、これが正解です。

n クィーンパズルでは、正解が複数存在する場合が多々あります。特定の **n** に対して考えられる解を見つけるのではなく、特定の **n** に対する正しい解を 1 つ導き出すことにフォーカスします 。

問題の規模

```
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 クィーンは、初心者用のサンプルとして実装されているため、最適化はされていません。それにもかかわらず、クィーンが 64 個になっても簡単に処理できます。何点か変更を加えると、クィーンが 5000 個以上になっても簡単に対応できることが立証されています。

3.7.1. N クィーンのドメインモデル

この例では、4 つのクィーンの問題を解決するドメインモデルを使用します。

- **ドメインモデルの作成**
  適切なドメインモデルを使用すると、プランニングの問題をより簡単に理解し、解決することができます。

  以下は、**n** クィーンの例のドメインモデルです。

```
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
}
```

- **探索空間の計算**
  Queen インスタンスには **Column** (例: 0 は列 A、1 は列 B) および **Row** (例: 0 は行 0、1 は行 1) が含まれます。

  列と行をもとに、昇順の対角線、および降順の対角線を計算することができます。

  列と行のインデックスは、チェスボードの左上隅から数えています。

```
public class NQueens {

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

    private List<Queen> queenList;

    private SimpleScore score;

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

- **解の求め方**
  1 つの **NQueens** インスタンスには **Queen** インスタンスの一覧が含まれています。これが **Solution** 実装として提供され、Solver が解決して読み出します。

たとえば、4 クイーンのサンプルでは、NQueens の **getN()** メソッドが常に 4 を返します。

図3.3 クィーン 4 個の解

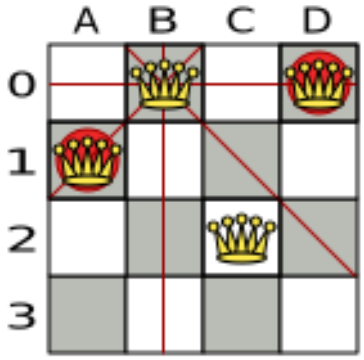


表3.2 ドメインモデルの解の詳細

	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

(*) や (**) のように、クィーン 2 つが同じ行、列、対角線を共有する場合は、2 つの駒が互いを取ることができます。

3.8. クラウドバランシング

この例に関する詳細は、Red Hat build of OptaPlanner クイックスタートガイドを参照してください。

3.9. 巡回セールスマン (TSP - 巡回セールスマン問題)

都市の一覧をもとに、セールスマンが最短距離で、各都市を 1 度だけ訪問するルートを探します。

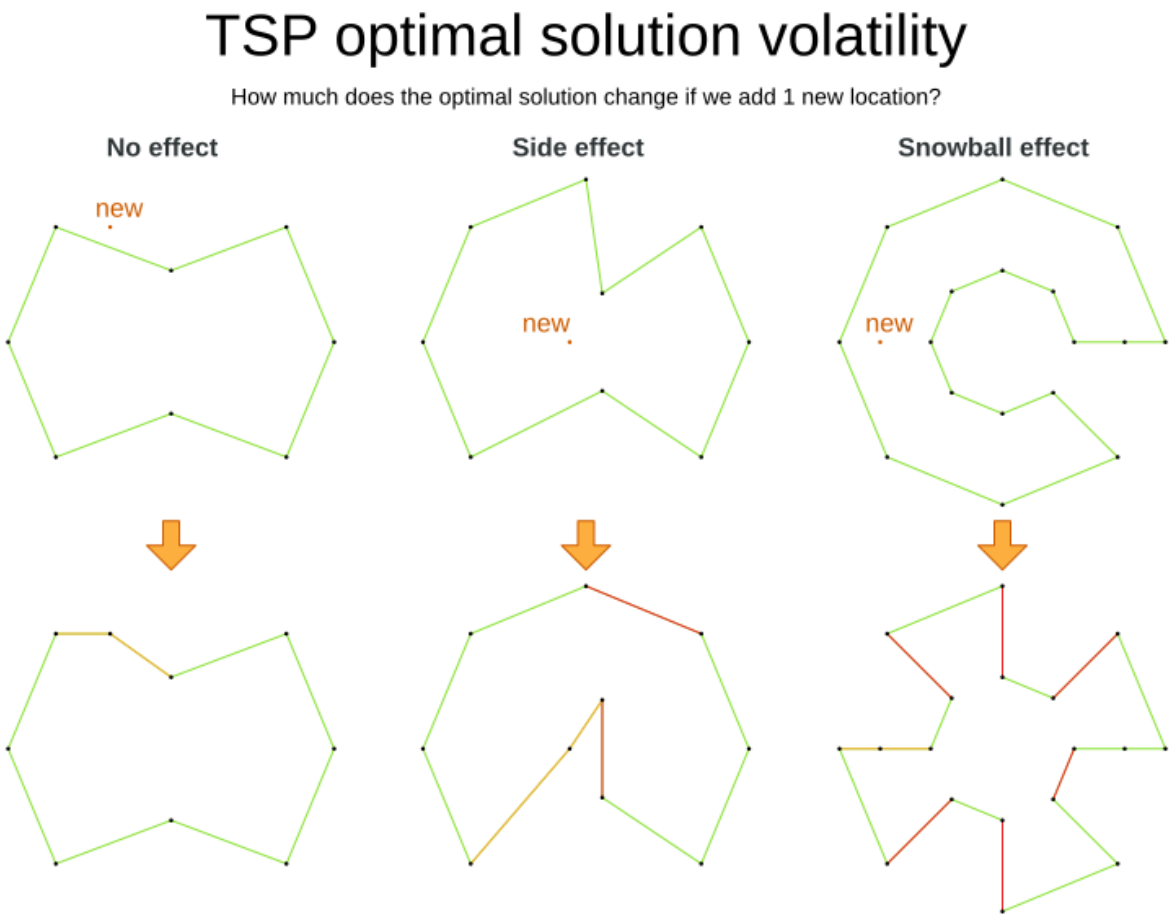
この問題は ウィキペディア に定義されています。これは、計算数学で 最も熱心に研究された問題の 1 つ です。大概は、従業員のシフト勤務など、その他の制約と一緒に計画の問題の一部として使用されます。

問題の規模

```
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.
```

問題の難易度

TSP の定義は単純ですが、問題の解決は驚くほど難しくなります。これは NP 困難問題と呼ばれ、多くの計画の問題と同様、特定の問題のデータセットに対する最適な解は、その問題のデータセットが少しでも変更すると、大幅に変化する可能性があります。



3.10. テニスクラブのスケジュール

テニスクラブでは、毎週 4 チームが総あたりで試合をします。4 つの対戦枠を公平にチームに割り当てます。

ハード制約:

- 競合: チームは 1 日に 1 回だけ試合ができる。
- 参加不可: 日程によって参加できないチームがある。

中程度の制約:

- 公平な割り当て: 各チームが試合をする回数を (ほぼ) 同じにする。

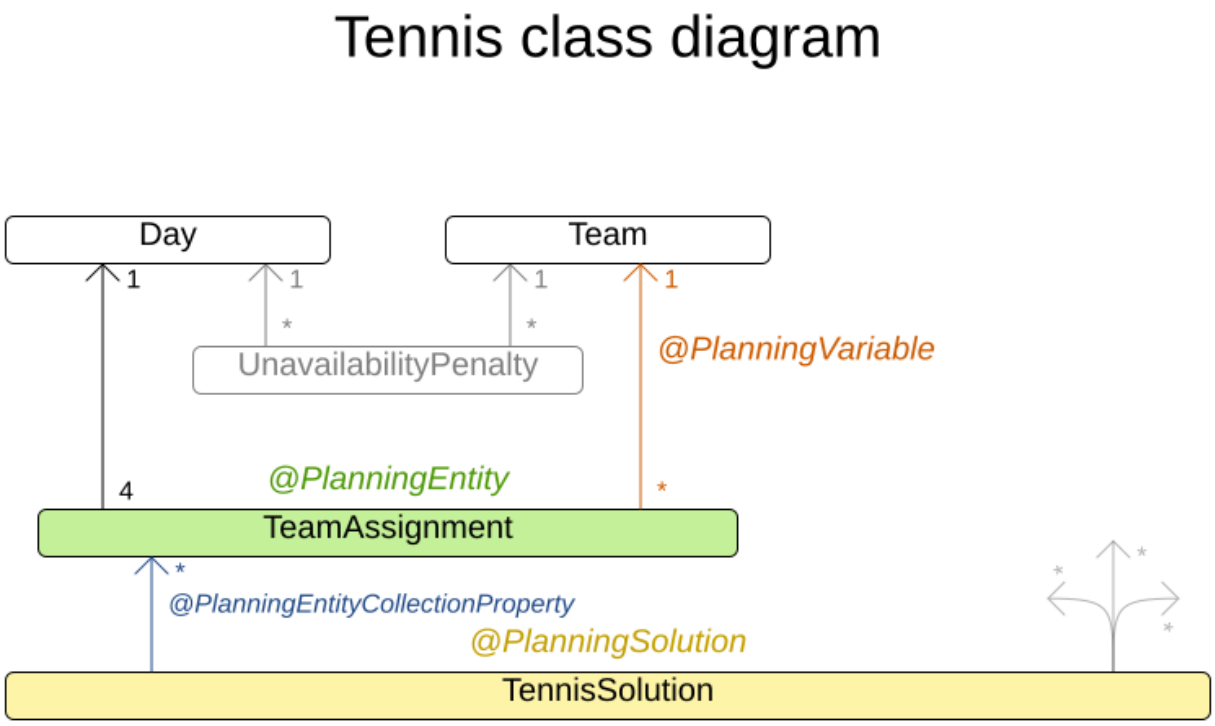
ソフト制約:

- 均等に対戦: 各チームが、各対戦相手と対戦する回数を同じにする。

問題の規模

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

図3.4 ドメインモデル



3.11. 会議のスケジュール

各会議に、開始時間と会議室を割り当てます。会議の長さは異なります。

ハード制約:

- 部屋の制約: 2 つの会議が、同じ時間に同じ会議室を使用することはできない。
- 必須の出席者: 同じ時間に開催される必須の会議を 2 つ割り当てることはできない。
- 必要とされる部屋の収容人数: 会議の出席者全員を収容できない部屋では会議を行ってはいけない。
- 同日中に開始して終了: 会議は複数の日にわたってスケジュールされないようにする。

中程度の制約:

- 任意の出席者: 同じ時間に開催される任意の会議を 2 つ割り当てることはできない。また、任意の会議と必須の会議を同じ時間に割り当てることはできない。

ソフト制約:

- 早い段階でスケジュール: すべての会議をできるだけ早くスケジュールする。
- 会議と会議の間の休憩時間: 会議と会議の間には、最低でも時間枠 1 つ分、休憩を入れる必要がある。

- 会議の重複: 並行して行われる会議の数を最小限に抑えて、どちらかの会議を選択しなければならない状況をなくす。
- 先に大きい部屋から割り当てる: 参加者が登録していない場合でも、できるだけ多数の参加者を収容するために、大きい部屋が空いている場合にはその部屋から割り当てていく必要がある。
- 部屋の不変性: 会議が連続して行われ、休憩の時間枠が 2 つ分より少ない場合には、会議は同じ部屋で行う方が良い。

問題の規模

> 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. コースの時間割 (ITC 2007 TRACK 3 - カリキュラムのスケジュール)

各授業を、時間枠および講義室に割り当ててスケジュールを組みます。

ハード制約:

- 講師の制約: 各講師は、同じ時間に授業を 2 つ受け持つことはできない。
- カリキュラムの制約: カリキュラムには、2 つの授業を同じ時間に設定することはできない。
- 部屋の占有: 同じ時間の同じ講義室に、2 つの授業を割り当てることはできない。
- 利用不可の時間 (データセットごとに指定): 授業には割り当てられない時間がある。

ソフト制約:

- 講義室の収容人数: 講義室の収容人数は、その授業を受ける学生の数よりも多くなければならない。
- 最小限の就業日数: 同じコースの授業の開講期間は、最短になるようにする。
- カリキュラムの緊密さ: 同じカリキュラムに含まれる授業は、時間帯を近く (連続した時間に) 設定する。
- 講義室の不変性: 同じコースの授業は同じ講義室を割り当てる必要がある。

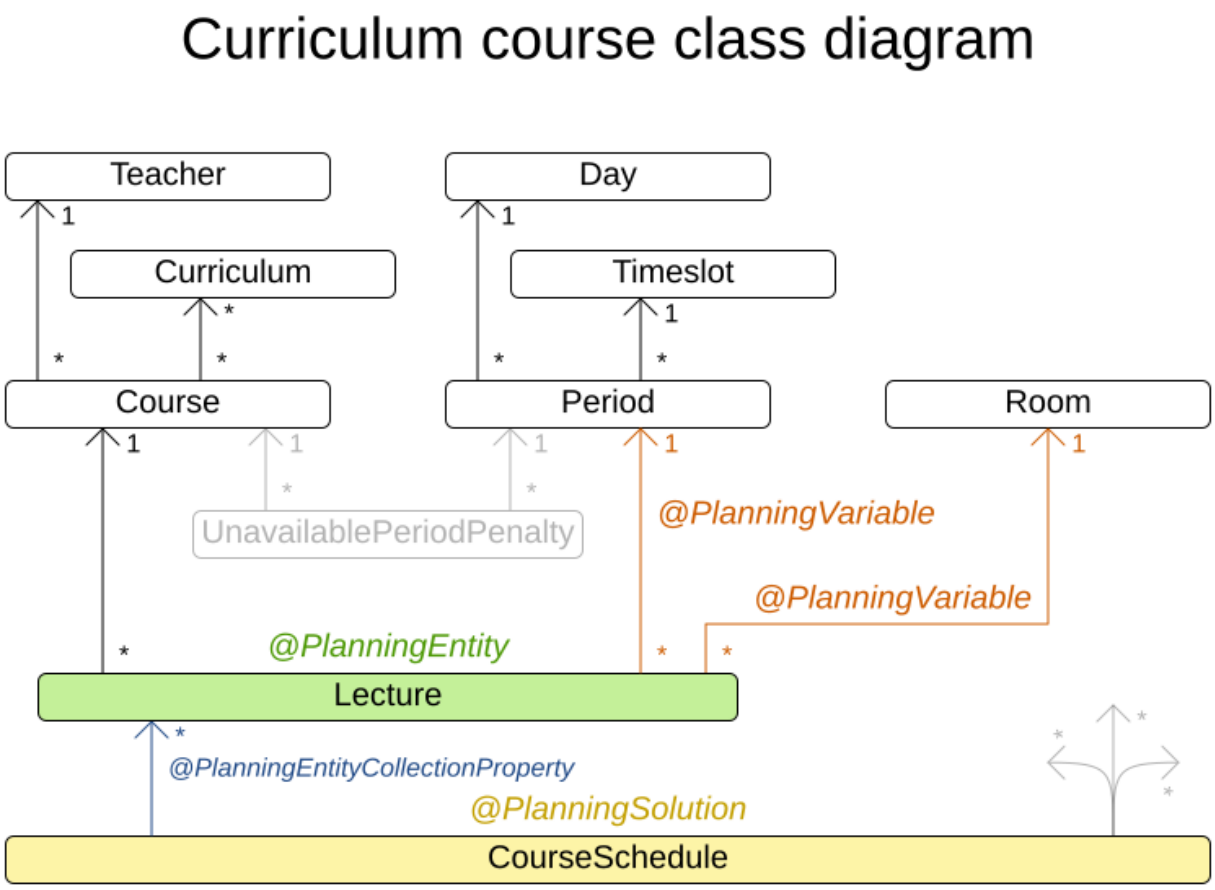
この問題は、International Timetabling Competition 2007 track 3 で定義されています。

問題の規模

> 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.

図3.5 ドメインモデル



3.13. マシンの再割当て (GOOGLE ROADEF 2012)

各プロセスをマシンに割り当てます。全プロセスには、すでに元の (最適化されていない) 割り当てがあります。プロセスにはそれぞれ、各リソース (CPU、メモリーなど) が一定量必要です。これは、クラウドのバランスの例の応用です。

ハード制約:

- 最大容量: マシンに割り当てる各リソースはこの量を超えてはいけない。
- 競合: 同じサービスのプロセスは別のマシンで実行する必要がある。
- 分散: 同じサービスのプロセスは複数の場所に分散させる必要がある。
- 依存関係: 他のサービスに依存するサービスのプロセスは、そのサービスの近くで実行する必要がある。
- 一時的な使用: リソースによっては一時的なものがあり、元のマシンと、新たに割り当てられたマシンの両方の最大容量にカウントされる。

ソフト制約:

- 負荷: 各マシンの各リソースの安全容量を超えてはいけない。
- 負荷分散: 各マシンで利用可能なリソースを分散させて、今後の割り当てに対応できるように容量を空ける。
- プロセスの移動コスト: プロセスには移動コストが発生する。
- サービスの移動コスト: サービスには移動コストが発生する。
- 機械の移動コスト: マシン A からマシン B にプロセスを移動すると、A から B に固有の移動コストが別途発生する。

この問題は the Google ROADEF/EURO Challenge 2012 で定義されています。

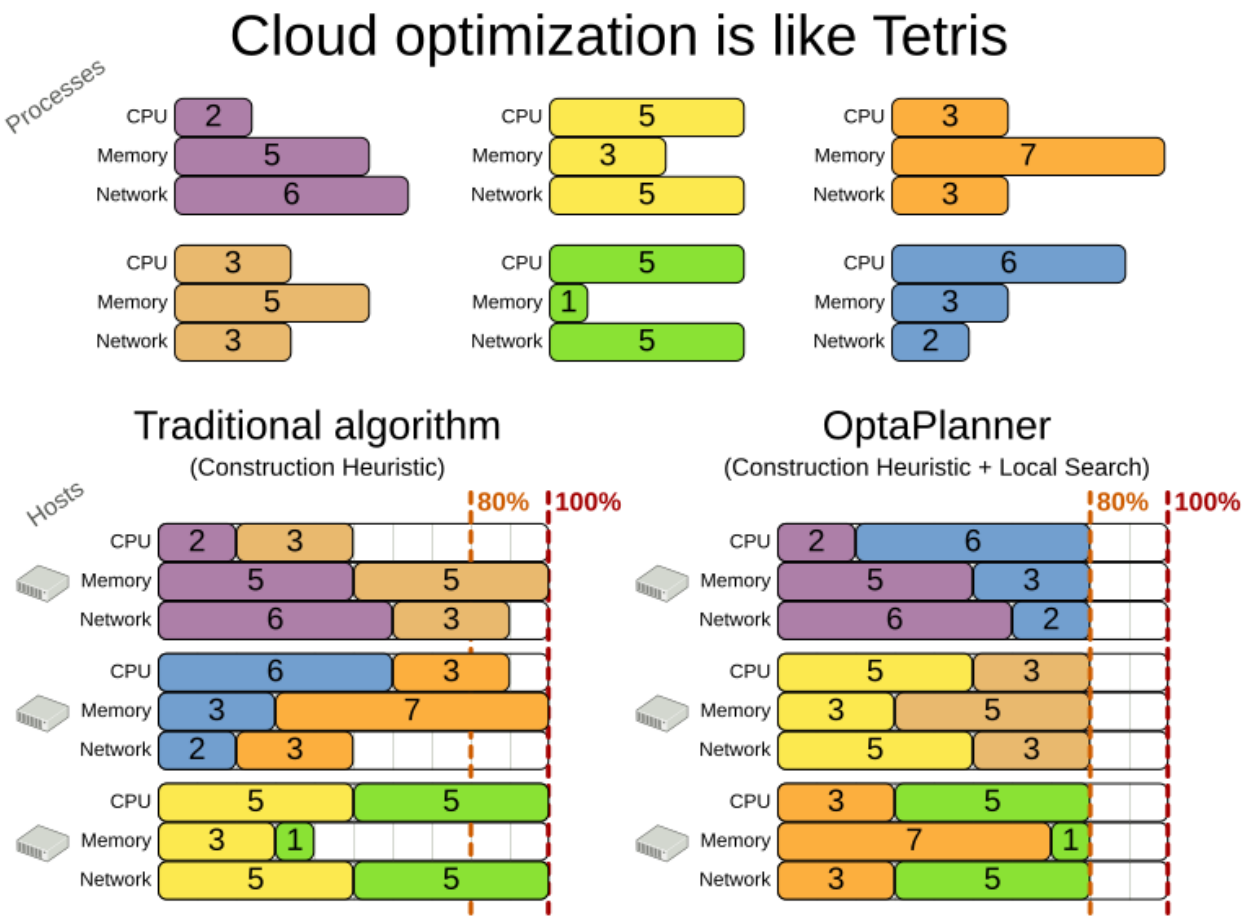
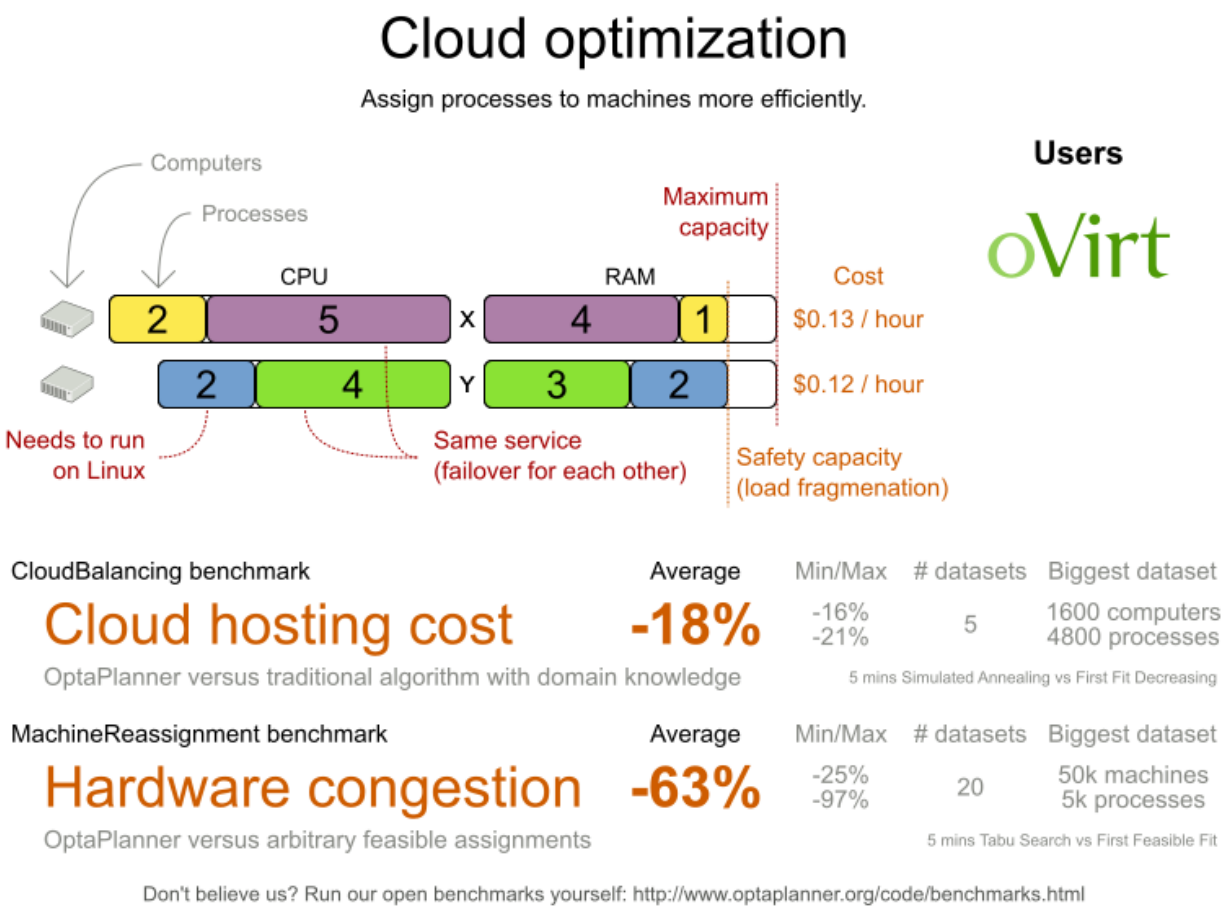

Cloud optimization is like Tetris
Processes
CPU 2
Memory 5
Network 6
CPU 5
Memory 3
Network 5
CPU 3
Memory 7
Network 3
CPU 3
Memory 5
Network 3
CPU 5
Memory 1
Network 5
CPU 6
Memory 3
Network 2
Traditional algorithm
(Construction Heuristic)
Hosts
80%
100%
CPU 2 3
Memory 5 5
Network 6 3
CPU 6 3
Memory 3 7
Network 2 3
CPU 5 5
Memory 3 1
Network 5 5
OptaPlanner
(Construction Heuristic + Local Search)
80%
100%
CPU 2 6
Memory 5 3
Network 6 2
CPU 5 3
Memory 3 5
Network 5 3
CPU 3 5
Memory 7 1
Network 3 5

図3.6 価値提案

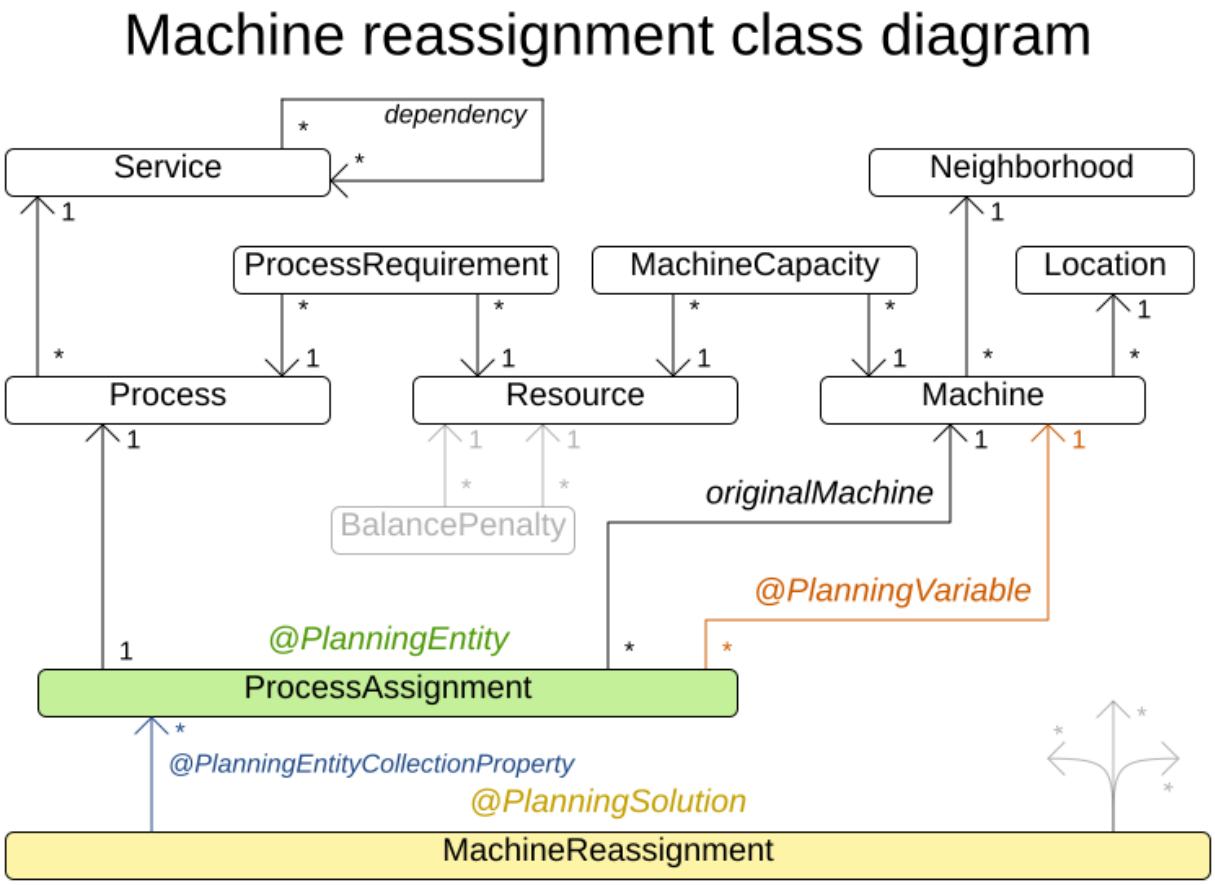


問題の規模

```
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.
```

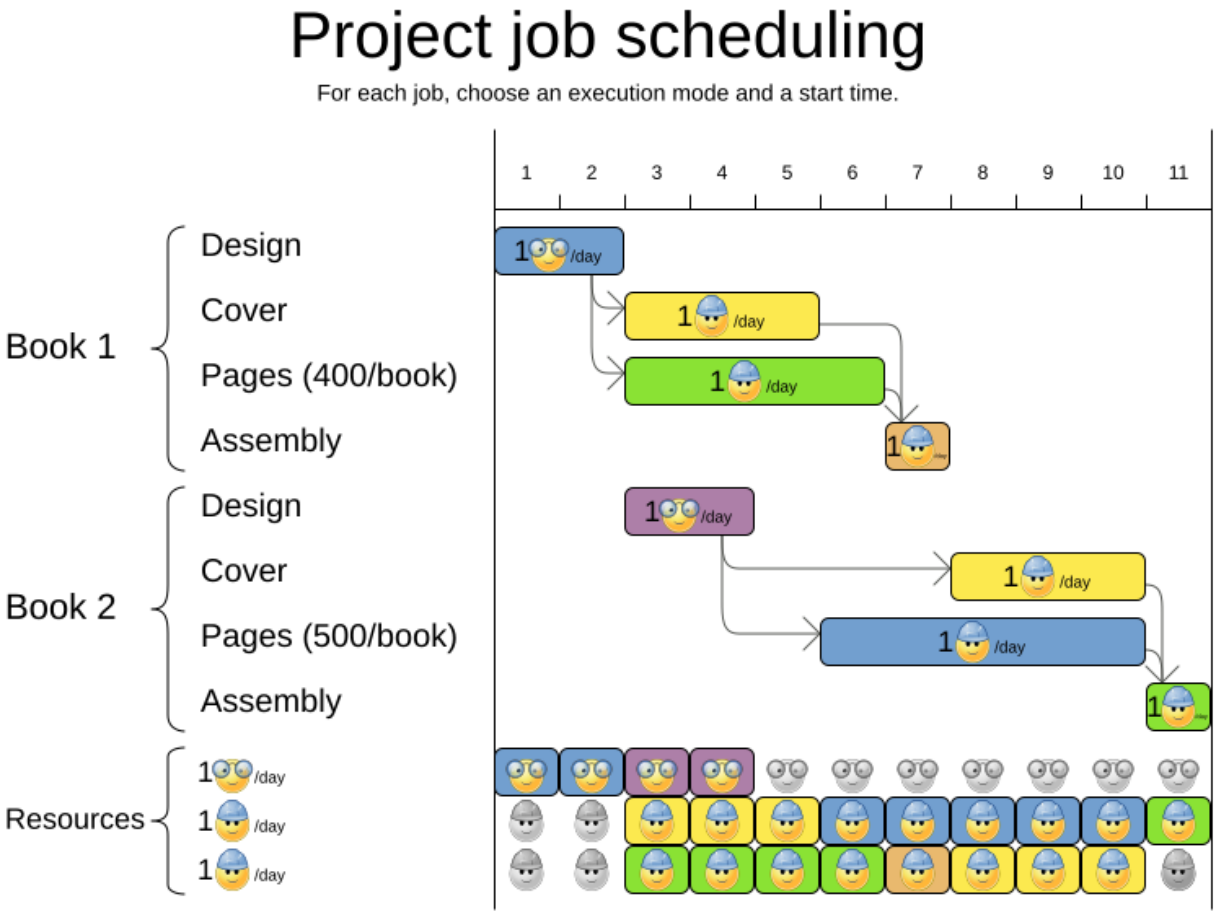
model_b_3 has 6 resources, 5 neighborhoods, 10 locations, 100 machines, 15025 services, 20000 processes and 0 balancePenalties with a search space of 10^40000.
model_b_4 has 6 resources, 5 neighborhoods, 50 locations, 500 machines, 1732 services, 20000 processes and 1 balancePenalties with a search space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

図3.7 ドメインモデル



3.14. プロジェクトジョブのスケジュール

プロジェクトの遅延を最小限に抑えるために、すべてのジョブを時間内に実行できるようにスケジュールを設定します。各ジョブは、プロジェクトに含まれます。ジョブは、異なる方法で実行できます。方法ごとに期間や使用するリソースが異なります。これは、柔軟な **ジョブショップスケジューリング (JSP)** の応用です。



ハード制約:

- ジョブの優先順位: ジョブは、先行のジョブがすべて完了するまで開始しない。
- リソースの容量: 利用可能な量を超えるリソースを使用しない。
  - リソースはローカル (同じプロジェクトのジョブ間で共有)、またはグローバル (全ジョブ間で共有) とする。
  - リソースは更新可能 (1 日に利用可能な容量) または更新不可 (全日で利用可能な容量) とする。

中程度の制約:

- プロジェクトの合計遅延時間: 各プロジェクトの所要時間 (メイクスパン) を最短にする。

ソフト制約:

- メイクスパン合計: 複数のプロジェクトスケジュールの合計所要時間を最短にする。

この問題は、the MISTA 2013 challenge で定義されています。

問題の規模

> 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. タスクの割り当て

従業員のキューのスポットに各タスクを割り当てます。タスクごとに、従業員のアフィニティーレベルから影響を受ける期間と、タスクの顧客が含まれます。

ハード制約:

- スキル: タスクごとに 1 つ以上のスキルが必要である。従業員には、このようなスキルがすべて必要です。

ソフトレベル 0 の制約:

- 極めて重要なタスク: 主要なタスクやマイナーなタスクの前に、極めて重要なタスクを完了する。

ソフトレベル 1 の制約:

- メークスパンの最小化: 全タスクを完了するまでの時間を短縮する。
    - 勤務歴の長い従業員から順番に進めていき、公平性やロードバランシングを作成する

- 勤務歴の長い従業員から順番に進めていき、公平性やロードバランシングを作成する。

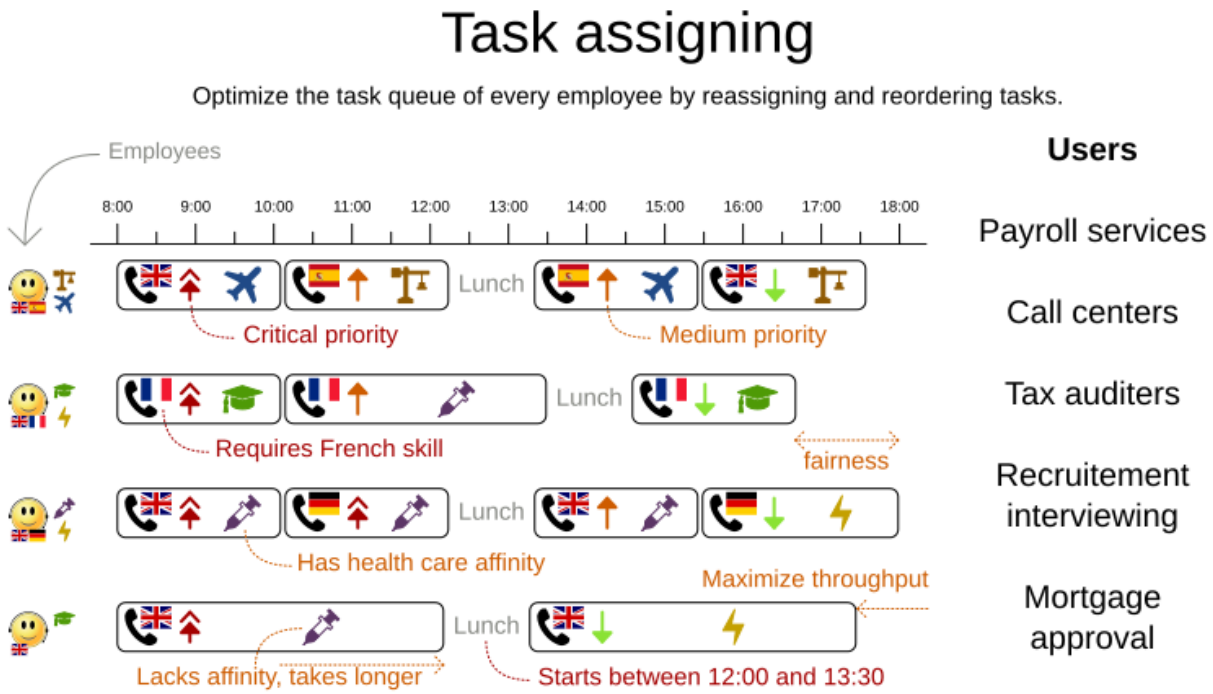
ソフトレベル 2 の制約:

- 主要なタスク: マイナーなタスクの前に、主要なタスクをできるだけ早く完了する。

ソフトレベル 3 の制約:

- マイナーなタスク: できるだけ早くマイナーなタスクを完了する。

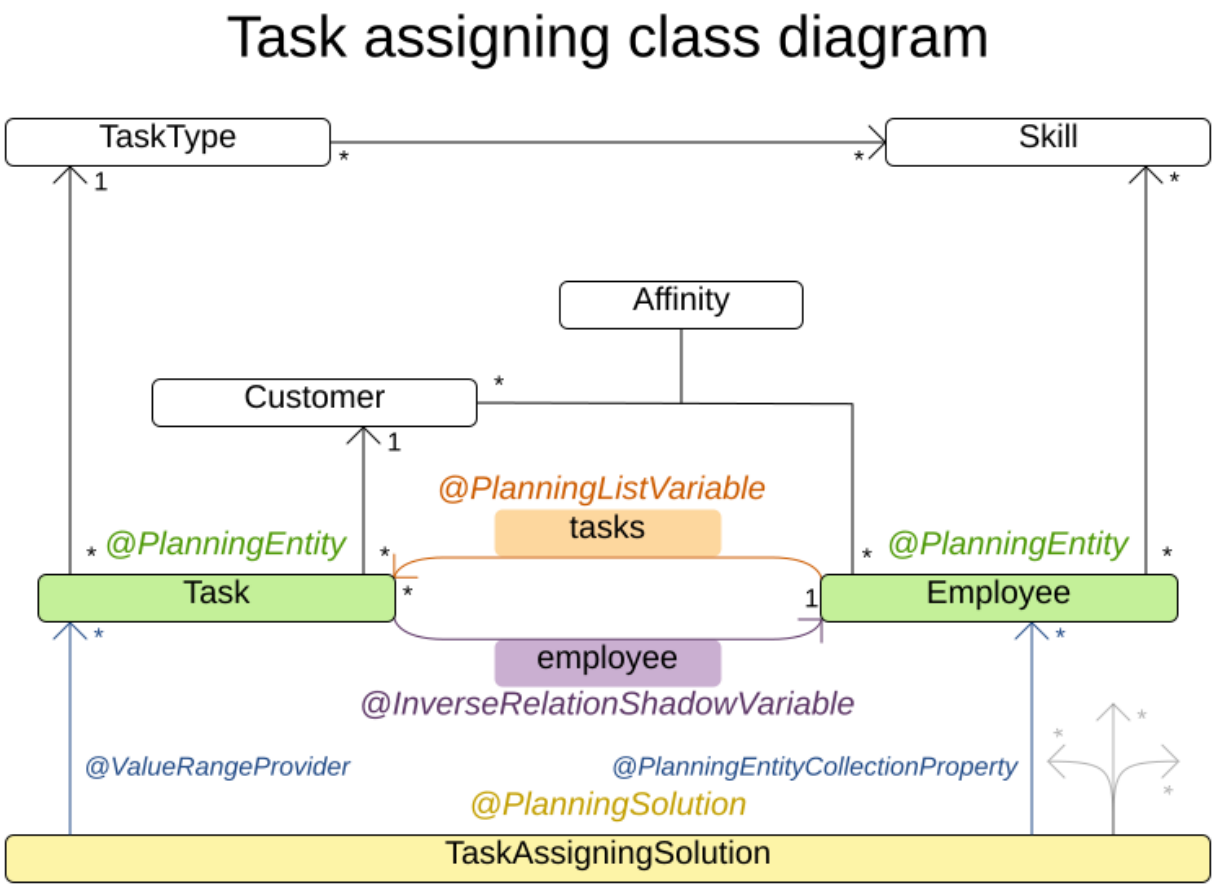
図3.8 価値提案



問題の規模

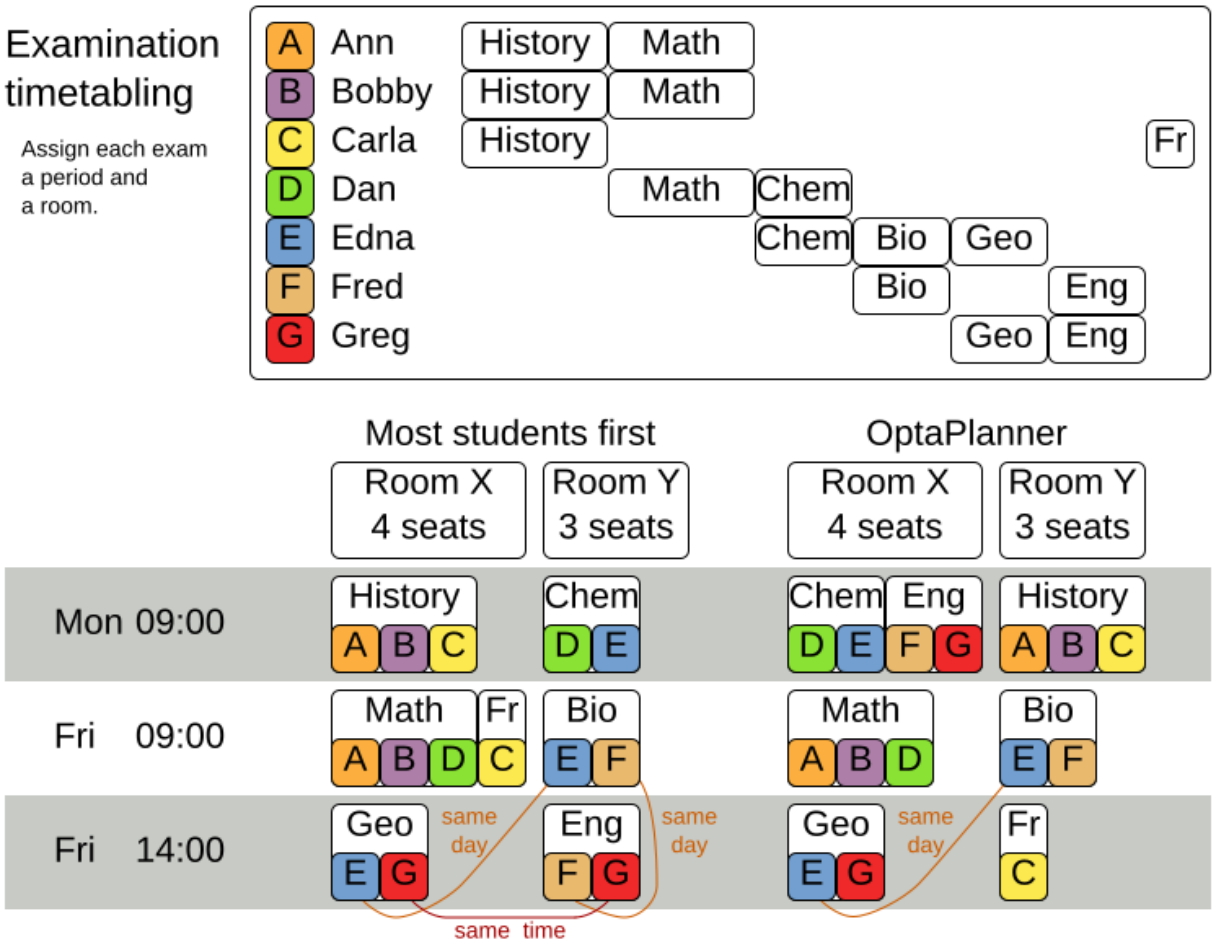
> 24tasks-8employees has 24 tasks, 6 skills, 8 employees, 4 task types and 4 customers with a search space of 10^30.
> 50tasks-5employees has 50 tasks, 5 skills, 5 employees, 10 task types and 10 customers with a search space of 10^69.
> 100tasks-5employees has 100 tasks, 5 skills, 5 employees, 20 task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

図3.9 ドメインモデル



3.16. 試験の時間割 (ITC 2007 TRACK 1 – 試験)

すべての試験に、時間と部屋を割り当てます。同じ時間帯に同じ部屋で、複数の試験を行うことができるものとします。



ハード制約:

- 試験の制約: 同じ学生が受ける 2 つの試験は、同じ時間帯に実施できないものとする。
- 教室の収容人数: 教室の座席数は、常に受験者数よりも多くなければならない。
- 期間: 期間は、すべての試験に対応できる長さでなければならない。
- 期間関連のハード制約 (データセットごとに指定):
  - 一致: 特定の 2 つの試験を同じ時間帯に設定する必要がある (別の教室を使用することも可能)。
  - 除外: 特定の 2 つの試験を同じ時間帯に設定できない。
  - 以降: 特定の試験を、別の特定の試験の後に行う必要がある。
- 教室関連の制約 (データセットごとに指定):
  - 排他的: 特定の試験を、他の試験と同じ教室で行うことはできない。

ソフト制約 (パラメーター化されたペナルティーがそれぞれ設定されている):

- 同じ学生が、続けて試験を 2 つ受けてはいけない。
- 同じ学生が、同じ日に試験を 2 つ受けてはいけない。
- 時間帯の分散: 同じ学生が受ける 2 つの試験は、時間をある程度あける。

- 異なる試験の長さ: 教室を共有する 2 つの試験の長さは、同じにする。
- 前倒し: 規模の大きい試験は、スケジュールを早めに決定する。
- 期間のペナルティー (データセットごとに指定): 期間によっては、使用されるとペナルティーが発生する。
- 部屋のペナルティー (データセットごとに指定): 部屋によっては、使用されるとペナルティーが発生する。

実際に大学から取得した大規模な試験データセットを使用します。

この問題は、International Timetabling Competition 2007 track 1 で定義されています。Geoffrey De Smet は、非常に初期バージョンの OptaPlanner で 4 位を終了しました。このコンペティション以降、多くの改良点が加えられています。

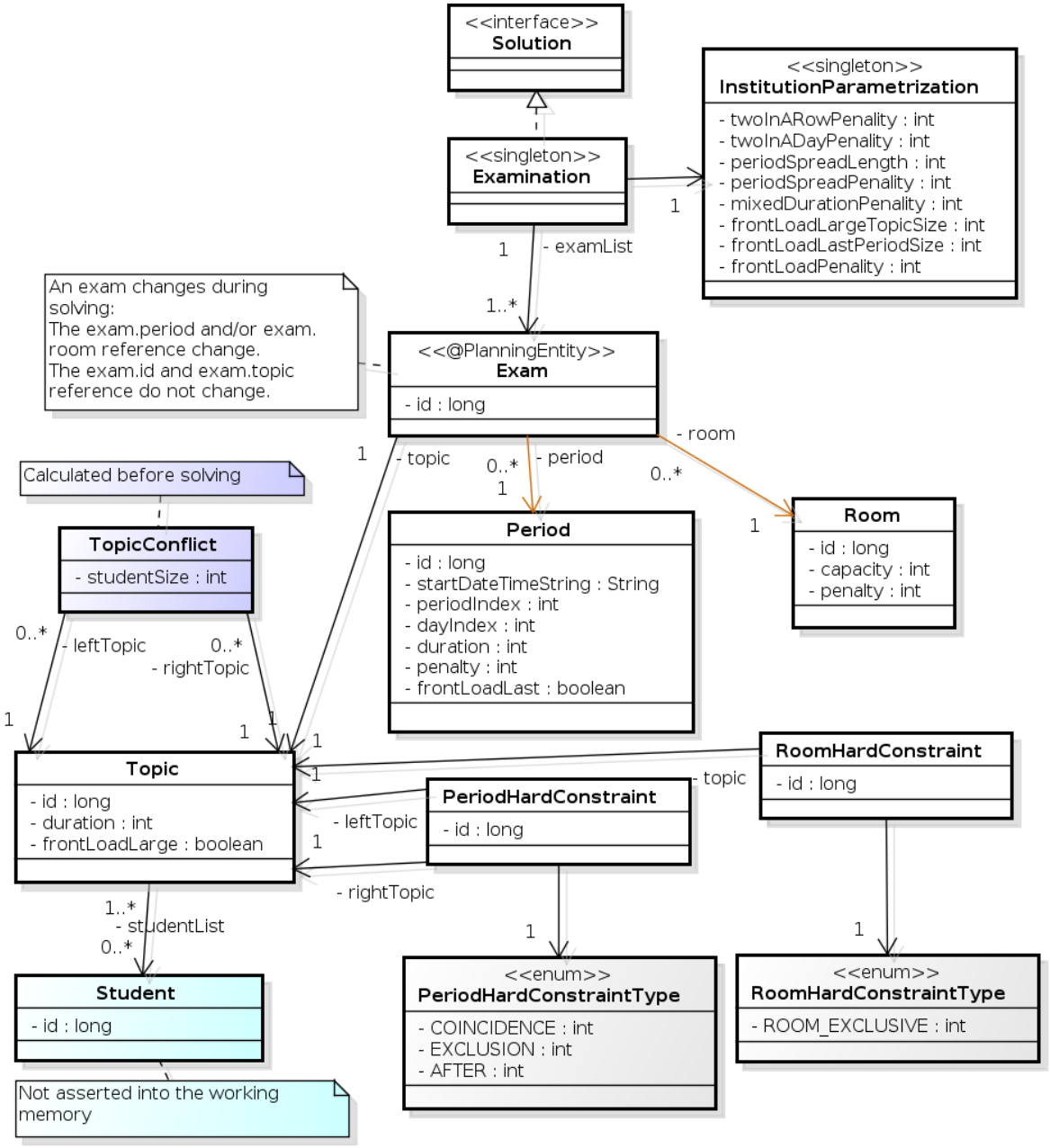
問題の規模

> 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. テストの時間割のドメインモデル

以下の図では、主な試験のドメインクラスを紹介しています。

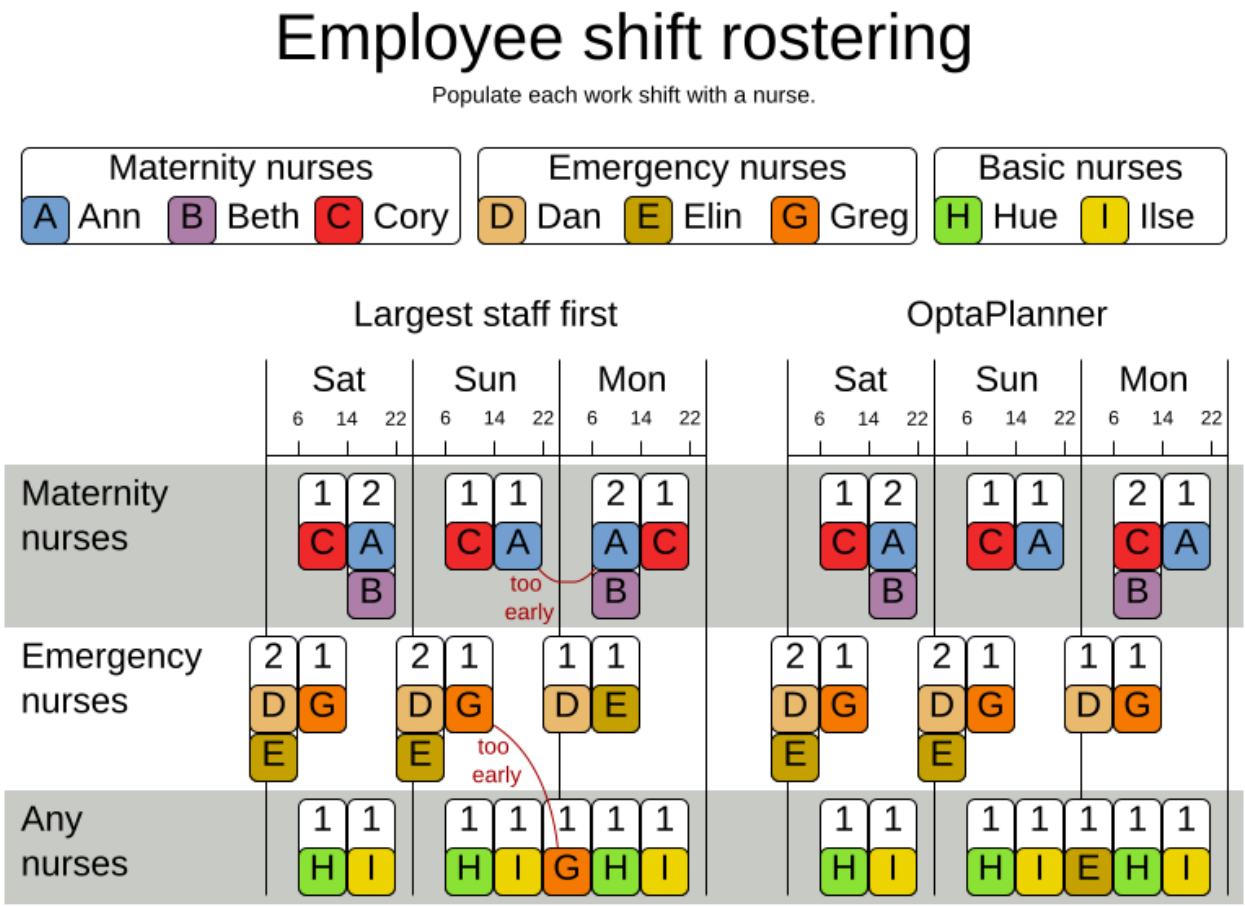
図3.10 試験のドメインクラスの図



試験のコンセプトを、**Exam** クラスと **Topic** クラスに分けた点に注意してください。期間または教室のプロパティーを変更し、解 (プランニングエンティティークラス) を求めると、**Exam** インスタンスが変化します。このとき、**Topic** インスタンス、**Period** インスタンス、および **Room** インスタンスは変化しません (他のクラスと同様、これらも問題ファクトです)。

3.17. 看護師の勤務表 (INRC 2010)

各シフトに看護師を割り当てます。



ハード制約:

- **未割り当てのシフトなし** (組み込み): すべてのシフトを従業員に割り当てる必要がある。
- **シフトの制約**: 従業員には 1 日に 1 シフトだけ割り当てることができる。

ソフト制約:

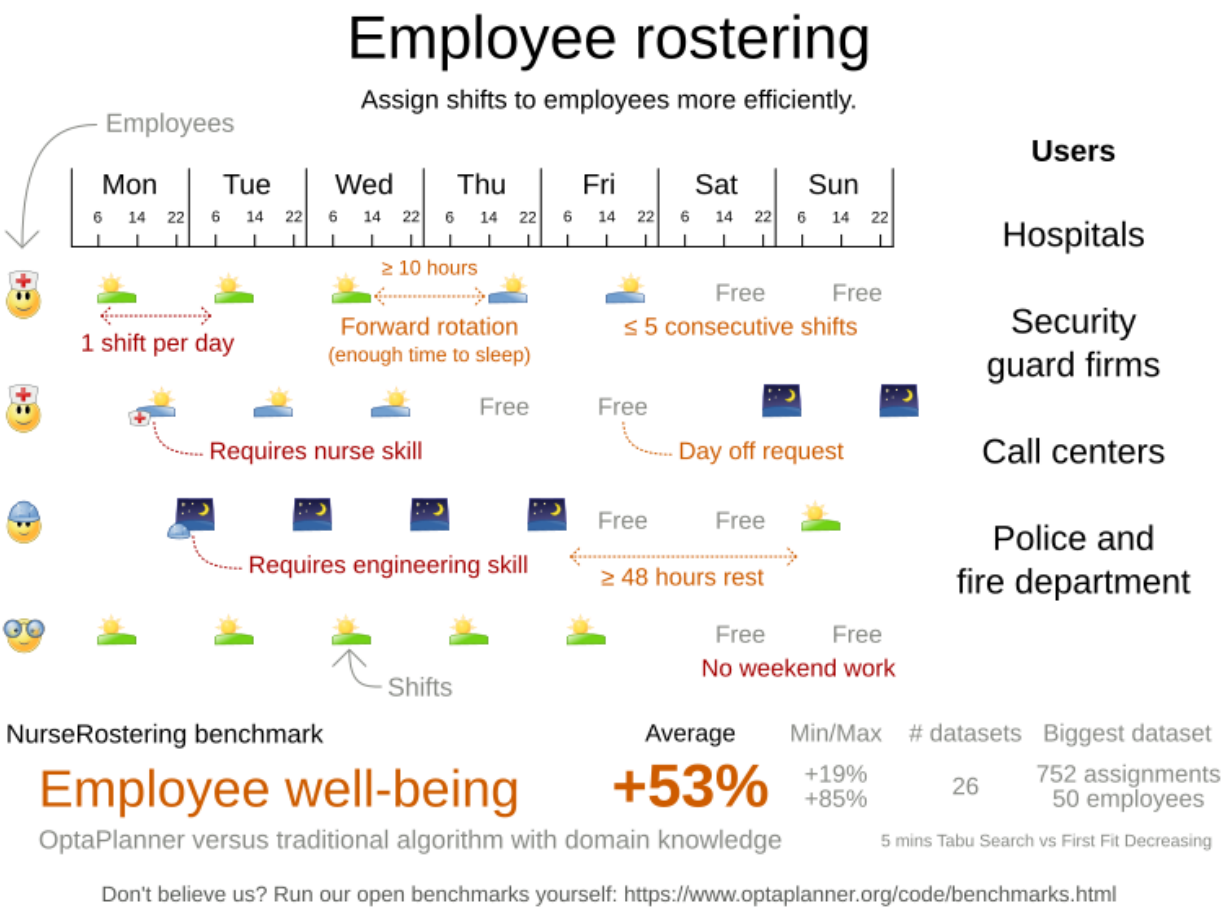
- 契約上の義務。この業界では、頻繁に契約上の義務に違反するため、ハード制約ではなく、ソフト制約として定義することに決定しました。
  - **割り当ての下限および上限**: 各従業員は、(それぞれの契約に合わせて) x より多く、y よりも少ないシフト数を勤務する必要がある。
  - **連続勤務日数の下限および上限**: 各従業員は、(それぞれの契約に合わせて) 連続で x 日から y 日間、勤務する必要がある。
  - **連続公休日数の下限および上限**: 各従業員は、(それぞれの契約に合わせて) 連続で x 日から y 日間、休む必要がある。
  - **週末に連続勤務する回数の下限および上限**: 各従業員は、(それぞれの契約に合わせて) 連続で x 回から y 回、週末勤務する必要がある。
  - **週末の勤務有無を同じにする**: 各従業員は、週末の両日を勤務する、または休む必要がある。
  - **週末のシフトタイプを同じにする**: 各従業員で、同じ週末のシフトタイプは、同じにする必要がある。

  - **好ましくないシフトパターン**: 遅番+早番+遅番など、好ましくないシフトタイプを連続で組み合わせたパターン。

- 従業員の希望:
  - **勤務日のリクエスト**: 従業員は、特定の勤務希望日を申請できる。
  - **公休日のリクエスト**: 従業員は、特定の公休希望日を申請できる。
  - **勤務するシフトのリクエスト**: 従業員は特定のシフトへの割り当てを希望できる。
  - **勤務しないシフトのリクエスト**: 従業員は特定のシフトに割り当てられないように希望できる。
- **他のスキル**: スキルに割り当てられた従業員は、そのシフトで必要な全スキルに堪能である必要がある。

この問題は International Nurse Rostering Competition 2010 で定義されています。

図3.11 価値提案



問題の規模

以下のように、データセットの種類は 3 つあります。

- sprint: 数秒で問題を解決する必要があります。
- medium: 数分で問題を解決する必要があります。
- long: 時間で解決する必要があります。

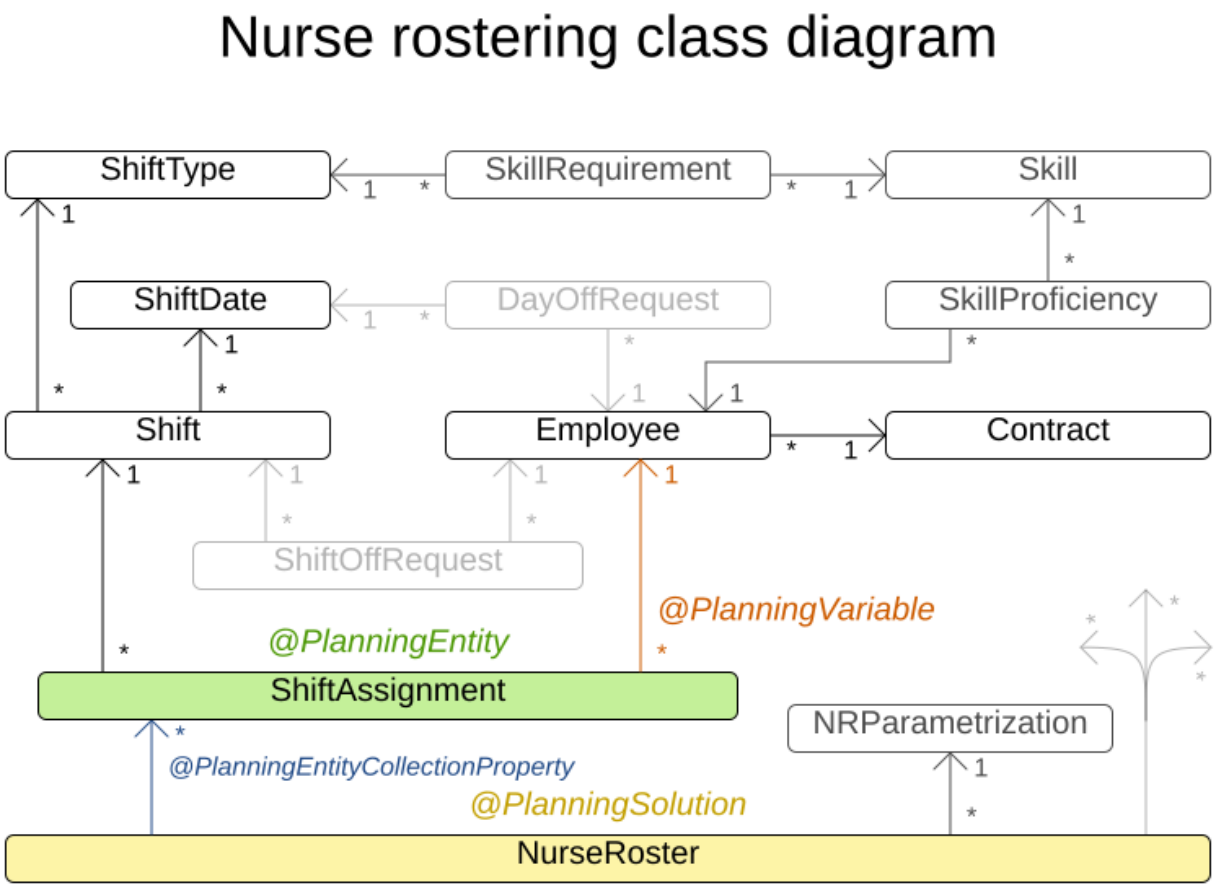
toy1 has 1 skills, 3 shiftTypes, 2 patterns, 1 contracts, 6 employees, 7 shiftDates, 35 shiftAssignments and 0 requests with a search space of 10^27.
toy2 has 1 skills, 3 shiftTypes, 3 patterns, 2 contracts, 20 employees, 28 shiftDates, 180 shiftAssignments and 140 requests with a search space of 10^234.

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.
```

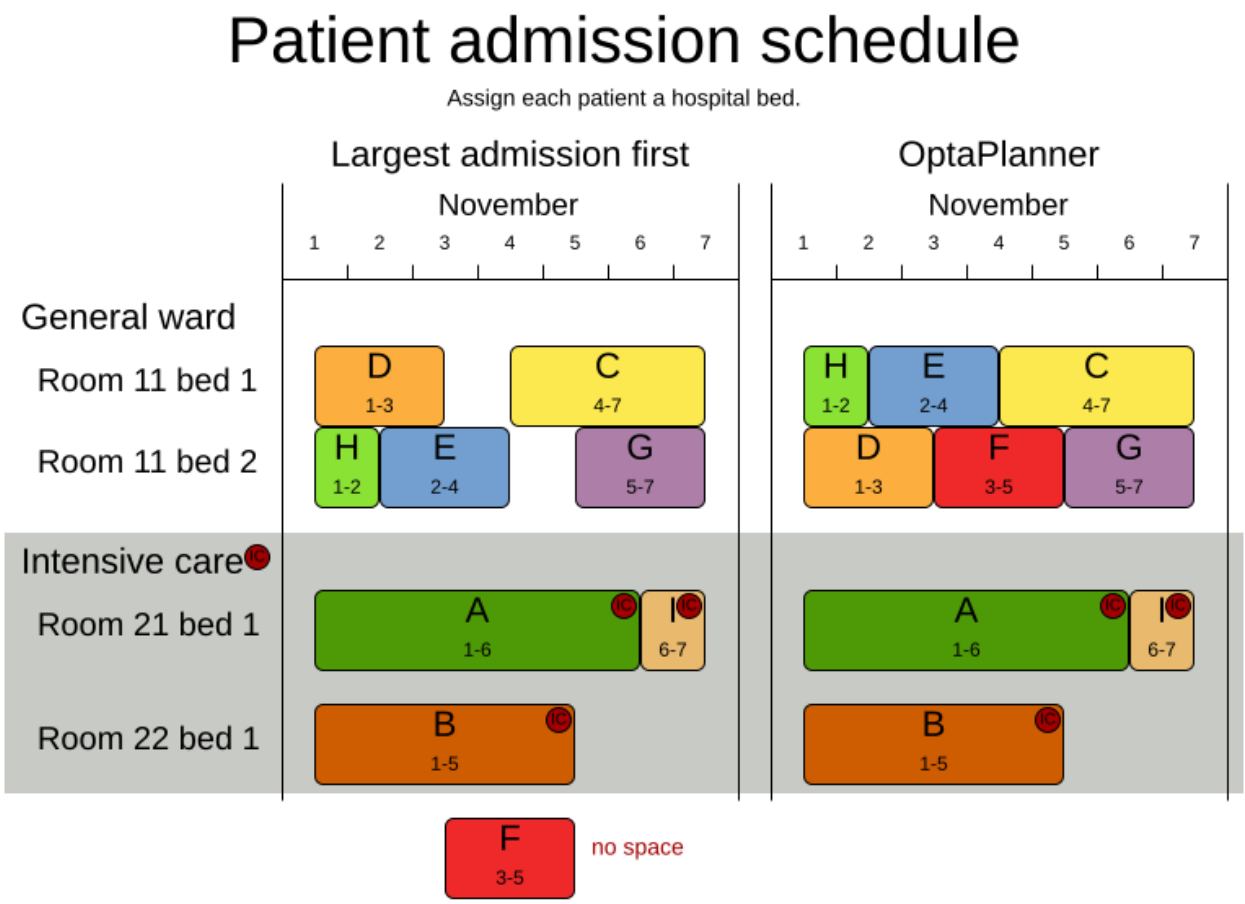
図3.12 ドメインモデル



3.18. 患者の入院スケジュール

患者入院スケジュール (PAS) は、病院のベッド計画とも呼ばれ、病院に入院する各患者にベッドを割り当てます。病床は、患者の予定された滞在期間中、患者に割り当てられます。各病床は病室に属し、各病室は部門に属します。患者の来院日と退院日は決まっています。病床を割り当てるだけで済みます。

この問題は、過度に制約されたデータセットを特徴としています。すべてのプランニングエンティティーを割り当てる必要がない場合は、厳しい制約に違反することなく、必要な数のエンティティーを割り当てることが適切です。これは、過剰制約プランニングと呼ばれます。



ハード制約:

- 同じ夜に 2 人の患者を同じ病床に割り当ててはなりません。重量: **-1000hard * conflictNightCount**。
- 病室には性別制限を設けることができます。女性のみ、男性のみ、同じ夜に同性が宿泊できる、または性別制限がまったくないなどです。重量: **-50hard * nightCount**。
- 部門には最低年齢または最高年齢を設定できます。重量: **-100hard * nightCount**。
- 患者は特定の設備を備えた部屋を要求する場合があります。重量: **-50hard * nightCount**。

中程度の制約:

- データセットが過度に制約されていない限り、すべての患者をベッドに割り当てます。重量: **-1medium * nightCount**

ソフト制約:

- 患者は、たとえば一人部屋を希望する場合など、部屋の最大サイズの好みを指定できます。重量: **-8soft * nightCount**
- 患者は、その患者の病状を専門とする部門に割り当てるのが最善です。重量: **-10soft * nightCount**.
- 患者は、その患者の病状を専門とする病室に割り当てるのが最善です。重量: **-20soft * nightCount**

    - 病室の専門性は、優先度 1 である必要があります。重量: **-10soft *(priority - 1)* nightCount**.

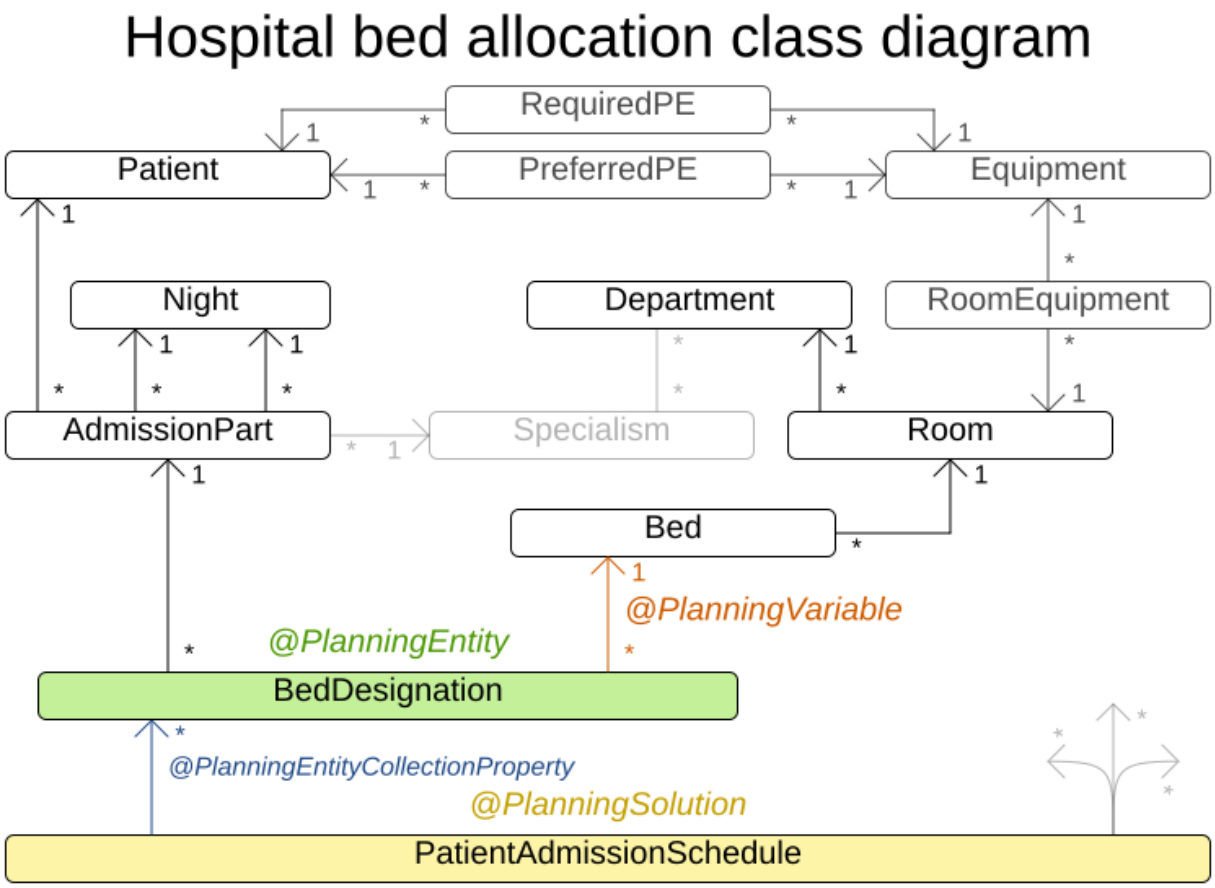
- 患者は、特定の設備を備えた部屋の希望を指定できます。重量: **-20soft * nightCount**

問題は Kaho's Patient Scheduling のバリエーションであり、データセットは実際の病院から取得します。

問題の規模

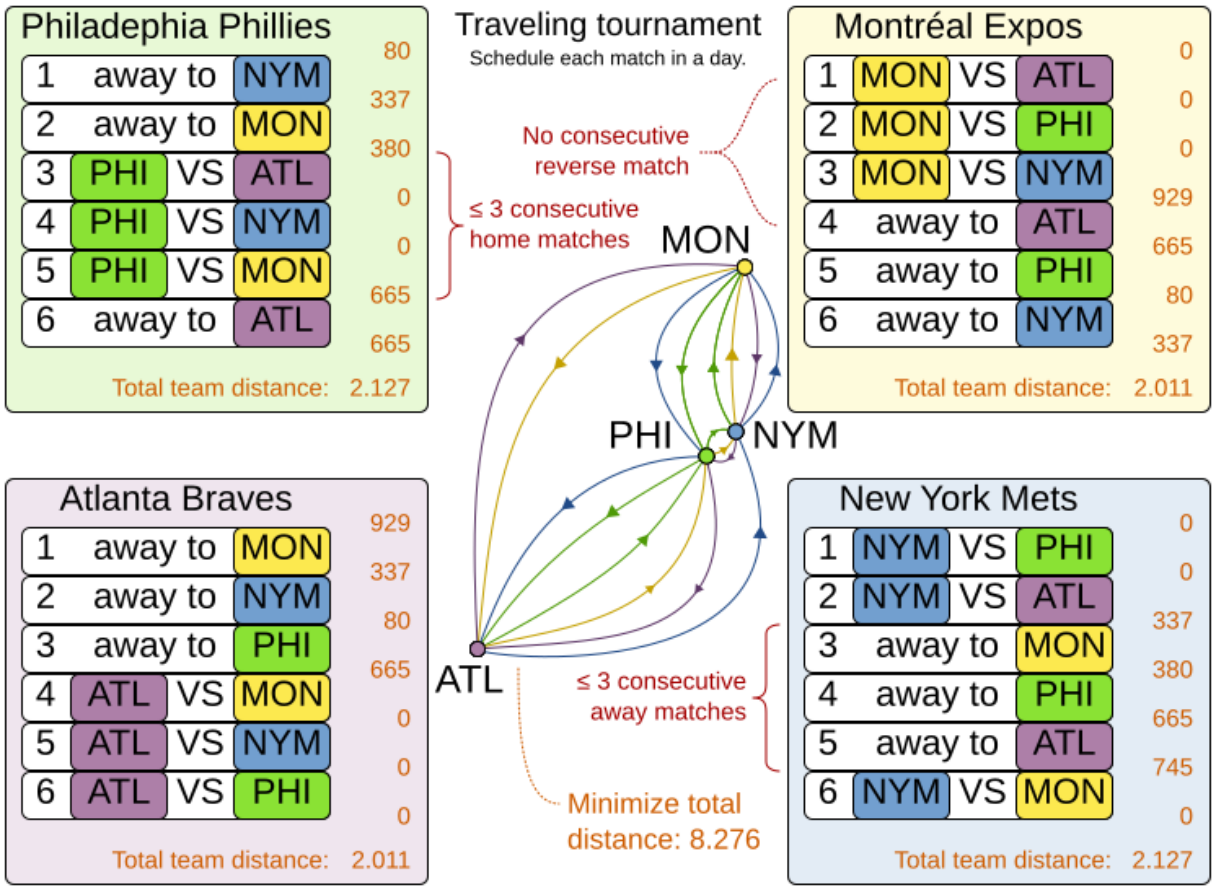
> overconstrained01 has 6 specialisms, 4 equipments, 1 departments, 25 rooms, 69 beds, 14 nights, 519 patients and 519 admissions with a search space of 10^958.
> testdata01 has 4 specialisms, 2 equipments, 4 departments, 98 rooms, 286 beds, 14 nights, 652 patients and 652 admissions with a search space 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.

図3.13 ドメインモデル



3.19. 巡回トーナメント問題 (TTP)

n 人数のチーム間の一致をスケジュールします。



ハード制約:

- 各チームは、他のチームとそれぞれ 2 回 (ホームとアウェイ) 試合をする。
- 各チームは、各時間枠に 1 試合だけ行う。
- 3 回連続で、ホームまたはアウェイでの試合はできない。
- 繰り返しなし: 同じ対戦相手と 2 回連続で対戦できない。

ソフト制約:

- 全チームが移動する合計距離を最小限に抑える。

この問題は Michael Trick の Web サイト (世界記録が含まれます) で定義されています。

問題の規模

```
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. コストを抑えるスケジュール

全タスクを時間内にスケジュールし、機械の電気代を最小限に抑えます。電気代は時間によって異なります。これは、**ジョブショップスケジューリング** の応用です。

ハード制約:

- 開始時間の制限: 各タスクは、最早と最遅の開始時間の制限内に、開始する必要がある。
- 最大容量: マシンに割り当てる各リソースはこの量を超えてはいけない。
- 開始および終了: 各機械は、タスクが割り当てられている間は稼働している必要がある。次のタスクまでの間、起動および終了コストを避けるため、機械をアイドルにすることができる。

中程度の制約:

- 電気代: 全スケジュールの合計電気代を最小限に抑える。
  - 機械の電気代: 稼働中またはアイドル中の機械はそれぞれ、電気を消費し、電気代が発生する (金額は使用時の電気代によって異なる)。
  - タスクの電気代: 各タスクも電気を消費し、電気代が発生する (金額は使用時の電気代によって異なる)。

- 機械の起動および終了コスト: 機械を起動または終了するたびに、追加のコストが発生する。

ソフト制約 (問題に元々設定されている定義に追加):

- 早く開始: なるべく早めにタスクを開始するようにする。

この問題は、ICON challenge で定義されています。

問題の規模

```
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. 投資資産クラスの割り当て (ポートフォリオの最適化)

各資産クラスに投資する相対数を決定します。

ハード制約:

- リスクの最大値: 標準偏差合計は、標準偏差の最大値を超えてはならない。
  - 標準偏差合計の計算は、Markowitz Portfolio Theory を適用した、資産クラスの相対関係を考慮する必要がある。
- 地域の最大値: 地域ごとに数量の最大値がある。
- セクターの最大値: 各セクターに数量の最大値がある。

ソフト制約:

- 期待収益を最大化する。

問題の規模

> 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.

サイズが大きいデータセットは作成/検証されていませんが、問題はないはずです。データに関する適切な情報源として、このアセット相関の Web サイト を参照してください。

3.22. 会議スケジュール

各会議を時間帯と部屋に割り当てていきます。時間帯は重複させることができます。LibreOffice や Excel で編集可能な ***.xlsx** ファイルとの読み書きが可能です。

ハード制約:

- 時間帯の会議タイプ: 会議のタイプは、時間帯の会議タイプと一致する必要がある。
- 部屋が使用中の時間帯: その会議の時間帯に、会議用の部屋が利用できなければならない。

- 部屋の競合: 2 つの会議が、同じ時間に同じ会議室を使用することはできない。
- 講演者が空いていない時間帯: 講演者は必ず、会議の時間帯に空いていなければならない。
- 講演者の競合: 同じ時間帯の 2 つの会議に同じ講演者を割り当てることができない。
- 汎用の時間帯および部屋タグ:
  - 講演者が要求する時間帯タグ: 講演者に、必須時間帯タグが付けられている場合、講演者の会議はそのタグが付いている時間に割り当てる必要がある。
  - 講演者の禁止時間帯タグ: 公演者に、禁止時間帯タグが割り当てられている場合は、そのタグの付いた時間帯に講演者の会議をどれも割り当てることができない。
  - 会議を設定する必要のある時間帯タグ: 会議に必須時間帯タグが付いている場合は、そのタグの付いた時間帯に割り当てる必要がある。
  - 会議の禁止時間帯タグ: 会議に、禁止時間帯タグが割り当てられている場合は、そのタグの付いた時間帯にその会議を割り当てることができない。
  - 講演者が要求する部屋のタグ: 講演者に、必須の部屋タグが付けられている場合、講演者の会議はそのタグが付いている部屋に割り当てる必要がある。
  - 講演者が禁止する部屋のタグ: 講演者に、禁止部屋のタグが付けられている場合、講演者の会議はそのタグが付いている部屋に割り当てことができない。
  - 会議を設定する必要のある部屋タグ: 会議に必須部屋タグが付いている場合は、そのタグの付いた部屋に割り当てる必要がある。
  - 会議の禁止部屋タグ: 会議に、禁止部屋タグが割り当てられている場合は、そのタグの付いた部屋にその会議を割り当てることができない。
- 他の会議と同じ時間帯に設定しないタグ: このタグが付いている会議は、同じ時間帯に重複してスケジュールしてはいけない。
- 受講条件が付いた会議: 受講条件が付いた会議をすべて完了してからでないと対象の会議をスケジュールしてはいけない。

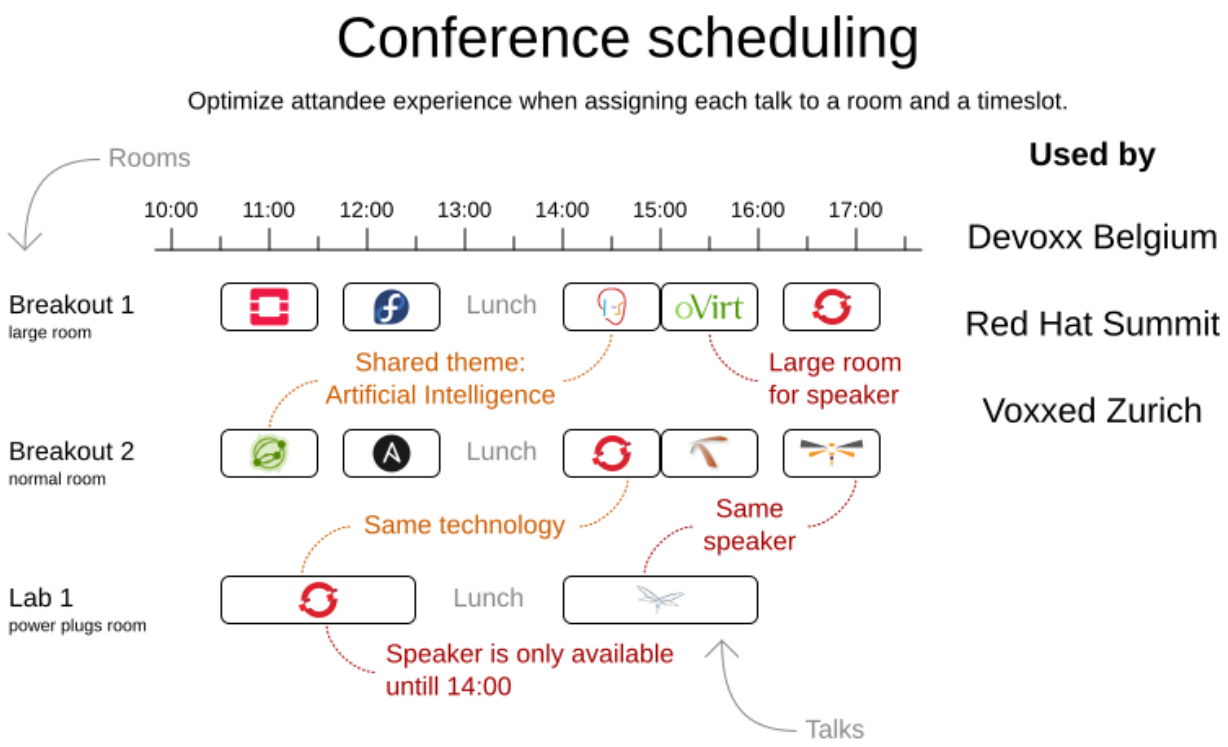
ソフト制約:

- テーマの追跡競合: 同じ時間帯で、テーマのタグが付いた会議の数を最小限に抑える。
- セクターの競合: 同じ時間帯で同じセクタータグの付いた会議の数を最小限に抑える。
- コンテンツの受講者レベルのフロー違反: すべてのコンテンツタグに対して、上級者用の会議の前に入門レベルの会議をスケジュールする。
- 受講者レベルの多様性: すべての時間帯において、異なる受講者レベルの会議数を最大限に増やす。
- 言語の多様性: すべての時間帯において、異なる言語の会議数を最大限を増やす。
- 汎用の時間帯および部屋タグ:
  - 講演者が希望する時間帯タグ: 講演者に、希望の時間帯タグが付けられている場合、講演者の会議はそのタグが付いている時間に割り当てるようにする。

- 講演者が希望しないタイムスロットタグ: 講演者が望ましくないタイムスロットタグを持っている場合、そのタグが付いているタイムスロットには講演を割り当ててはいけません。
- 会議の希望の時間帯タグ: 会議に希望の時間帯タグが付いている場合は、そのタグの付いた時間帯に割り当てるようにする。
- 会議の設定を希望しない時間帯タグ: 会議に、希望しない時間帯タグが付いている場合は、そのタグの付いた時間帯に割り当てないようにする。
- 講演者が希望する部屋のタグ: 講演者に、希望の部屋タグが付けられている場合、講演者の会議はそのタグが付いている部屋に割り当てるようにする。
- 講演者が希望しない部屋のタグ: 講演者に、希望しない部屋タグが付けられている場合、講演者の会議はそのタグが付いている部屋に割り当てるようにする。
- 会議を希望の部屋タグ: 会議に希望の部屋タグが付いている場合は、そのタグの付いた部屋に割り当てるようにする。
- 会議での使用を希望しない部屋タグ: 会議に、希望しない部屋タグが付いている場合、そのタグの付いた部屋に割り当てないようにする。

- 同じ日の会議: テーマタグまたはコンテンツタグを共有する会議は、最低限の日数 (理想的には同じ日) にスケジュールする必要がある。

図3.14 価値提案



問題の規模

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. ロックツアー

次のショーへの移動はロックバンクバスを使用し、空いている日のみショーをスケジュールする。

ハード制約:

- 必要とされるショーをすべてスケジュールする。
- できるだけ多くのショーをスケジュールする。

中程度の制約:

- 収益の機会を最大化する。
- 運転時間を最小限に抑える。
- できるだけ早く到着する。

ソフト制約:

- 長時間の運転は避ける。

問題の規模

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

3.24. 航空機乗組員のスケジューリング

パイロットと客室乗務員にフライトを割り当てます。

ハード制約:

- 必須スキル: フライトの割り当てにはそれぞれ、必要とされるスキルがあります。たとえば、フライト AB0001 ではパイロット 2 名と、客室乗務員 3 名が必要です。
- フライトの競合: 各従業員は同じ時間に出勤できるフライトは 1 つだけにする。
- 2 つのフライト間での移動: 2 つのフライトの間で、従業員は到着先の空港と、出発元の空港に移動できる必要がある。たとえば、アンは 10 時にブリュッセルに到着し、15 時にアムステルダムを出発するなどです。
- 従業員の勤務できない日: 従業員はフライトの当日は空いていなければならない。たとえば、アンは 2 月 1 日に休暇を取っているなど。

ソフト制約:

- 最初の仕事が自宅から出発する。

- 最後の仕事が自宅に到着する。
- 総フライト時間を従業員別に平均的に分散する。

問題の規模

> 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 RED HAT BUILD OF OPTAPLANNER サンプルのダウンロードおよびビルド

Red Hat Build of OptaPlanner の例は、Red Hat カスタマーポータルで入手できる Red Hat build of OptaPlanner ソースパッケージの一部としてダウンロードできます。

注記

Red Hat build of OptaPlanner は GUI に依存しません。デスクトップと同じように、サーバーまたはモバイル JVM 上でも実行できます。

手順

1. Red Hat カスタマーポータルの Software Downloads ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
   - **製品:** Red Hat build of OptaPlanner
   - **バージョン:** 8.38
2. **Red Hat build of OptaPlanner 8.38 Source Distribution**をダウンロードします。
3. **rhbop-8.38.0-optaplanner-sources.zip** ファイルを展開します。
   展開した **org.optaplanner.optaplanner-8.38.0.Final-redhat-00004/optaplanner-examples/src/main/java/org/optaplanner/examples** ディレクトリーには、サンプルソースコードが含まれています。
4. サンプルをビルドするには、**org.optaplanner.optaplanner-8.38.0.Final-redhat-00004** ディレクトリーで次のコマンドを入力します。

   ```
   mvn clean install -Dquickly
   ```

5. サンプルディレクトリーに移動します。

   ```
   optaplanner-examples
   ```

6. 例を実行するには、次のコマンドを入力します。

   ```
   mvn exec java
   ```

第5章 RED HAT BUILD OF QUARKUS プラットフォームでの RED HAT BUILD OF OPTAPLANNER の使用

Red Hat build of OptaPlanner は、Red Hat build of Quarkus プラットフォームと統合されます。OptaPlanner の依存関係を含むプラットフォームアーティファクトの依存関係のバージョンは、Quarkus bill of materials (BOM)ファイル **com.redhat.quarkus.platform:quarkus-bom** で維持されます。どの依存関係バージョンが連携するかを指定する必要はありません。代わりに、Quarkus BOM ファイルを **pom.xml** 設定ファイルにインポートできます。依存関係のバージョンは **<dependencyManagement>** セクションに含まれています。そのため、**pom.xml** ファイルの指定の BOM で管理される個別の Quarkus 依存関係のバージョンを記述する必要はありません。

関連情報

- Maven プラグインを使用して Quarkus プラットフォームに OptaPlanner プロジェクトを作成する方法については、「Maven プラグインを使用した Quarkus プラットフォームでの Red Hat build of OptaPlanner プロジェクトの作成」 を参照してください。
- **code.quarkus.redhat.com** Web サイトを使用して Quarkus プラットフォームで OptaPlanner プロジェクトを生成する方法は、「code.quarkus.redhat.com を使用した Quarkus プラットフォームでの Red Hat build of OptaPlanner プロジェクトの作成」 を参照してください。
- CLI を使用して Quarkus プラットフォームで OptaPlanner プロジェクトを生成する方法は、「Quarkus CLI を使用した Quarkus プラットフォームでの Red Hat build of OptaPlanner プロジェクトの作成」 を参照してください。

5.1. APACHE MAVEN および RED HAT BUILD OF QUARKUS

Apache Maven は分散型構築自動化ツールで、ソフトウェアプロジェクトの作成、ビルド、および管理を行うために Java アプリケーション開発で使用されます。Maven は Project Object Model (POM) ファイルと呼ばれる標準の設定ファイルを使用して、プロジェクトの定義や構築プロセスの管理を行います。POM ファイルは、モジュールおよびコンポーネントの依存関係、ビルドの順番、結果となるプロジェクトパッケージのターゲットを記述し、XML ファイルを使用して出力します。これにより、プロジェクトが適切かつ統一された状態でビルドされるようになります。

Maven リポジトリー

Maven リポジトリーには、Java ライブラリー、プラグイン、およびその他のビルドアーティファクトが格納されます。デフォルトのパブリックリポジトリーは Maven 2 Central Repository ですが、複数の開発チームの間で共通のアーティファクトを共有する目的で、社内のプライベートおよび内部リポジトリーとすることが可能です。また、サードパーティーのリポジトリーも利用できます。

Quarkus プロジェクトでオンライン Maven リポジトリーを使用するか、Red Hat build of Quarkus の Maven リポジトリーをダウンロードできます。

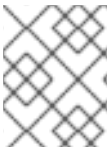
Maven プラグイン

Maven プラグインは、POM ファイルの定義済みの部分で 1 つ以上のゴールを達成します。Quarkus アプリケーションは以下の Maven プラグインを使用します。

- Quarkus Maven プラグイン (**quarkus-maven-plugin**): Maven による Quarkus プロジェクトの作成を実現して、uber-JAR ファイルの生成をサポートし、開発モードを提供します。
- Maven Surefire プラグイン (**maven-surefire-plugin**): ビルドライフサイクルのテストフェーズで使用され、アプリケーションでユニットテストを実行します。プラグインは、テストレポートが含まれるテキストファイルと XML ファイルを生成します。

5.1.1. オンラインリポジトリーの Maven の settings.xml ファイルの設定

ユーザーの **settings.xml** ファイルを設定して、Maven プロジェクトでオンライン Maven リポジトリーを使用できます。これは、推奨の手法です。リポジトリーマネージャーまたは共有サーバー上のリポジトリーと使用する Maven 設定は、プロジェクトの制御および管理性を向上させます。



注記

Maven の **settings.xml** ファイルを変更してリポジトリーを設定する場合、変更はすべての Maven プロジェクトに適用されます。

手順

1. テキストエディターまたは統合開発環境 (IDE) で Maven の **~/.m2/settings.xml** ファイルを開きます。

注記

~/.m2/ ディレクトリーに **settings.xml** ファイルがない場合には、**$MAVEN_HOME/.m2/conf/** ディレクトリーから **~/.m2/** ディレクトリーに **settings.xml** ファイルをコピーします。

2. 以下の行を Maven の **settings.xml** ファイルの **<profiles>** 要素に追加します。

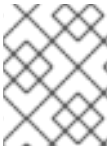
```
<!-- Configure the Maven repository -->
<profile>
 <id>red-hat-enterprise-maven-repository</id>
 <repositories>
  <repository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>https://maven.repository.redhat.com/ga/</url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>
    <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. 以下の行を **settings.xml** ファイルの **<activeProfiles>** 要素に追加し、ファイルを保存します。

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

5.1.2. Quarkus Maven リポジトリーのダウンロードおよび設定

オンライン Maven リポジトリーを使用しない場合は、Quarkus Maven リポジトリーをダウンロードして設定できます。Quarkus Maven リポジトリーには、Java 開発者がアプリケーションの構築に使用する要件の多くが含まれています。この手順では、Maven の **settings.xml** ファイルを編集し、Quarkus Maven リポジトリーを設定する方法を説明します。



注記

Maven の **settings.xml** ファイルを変更してリポジトリーを設定する場合、変更はすべての Maven プロジェクトに適用されます。

手順

1. Red Hat カスタマーポータルの Software Downloads ページから Red Hat build of Quarkus Maven リポジトリーの ZIP ファイルをダウンロードします。
2. ダウンロードしたアーカイブを展開します。
3. ~/**.m2**/ ディレクトリーに移動し、テキストエディターまたは統合開発環境 (IDE) で Maven の **settings.xml** ファイルを開きます。
4. 以下の行を **settings.xml** ファイルの **<profiles>** 要素に追加します。ここで、**QUARKUS_MAVEN_REPOSITORY** はダウンロードした Maven リポジトリーのパスです。**QUARKUS_MAVEN_REPOSITORY** は **file://$PATH** でなければなりません。たとえば **file:///home/userX/rh-quarkus-2.13.8.GA-maven-repository/maven-repository** のようになります。

```
<!-- 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. 以下の行を **settings.xml** ファイルの **<activeProfiles>** 要素に追加し、ファイルを保存します。

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

重要

Maven リポジトリーに古いアーティファクトが含まれる場合は、プロジェクトをビルドまたはデプロイしたときに以下のいずれかの Maven エラーメッセージが表示されることがあります。ここで、**ARTIFACT_NAME** は不明なアーティファクトの名前で、**PROJECT_NAME** は構築を試みているプロジェクトの名前になります。

- **Missing artifact PROJECT_NAME**
- **[ERROR] Failed to execute goal on project ARTIFACT_NAME; Could not resolve dependencies for PROJECT_NAME**

この問題を解決するには、~/**.m2/repository** ディレクトリーにあるローカルリポジトリーのキャッシュバージョンを削除し、最新の Maven アーティファクトを強制的にダウンロードします。

5.2. MAVEN プラグインを使用した QUARKUS プラットフォームでの RED HAT BUILD OF OPTAPLANNER プロジェクトの作成

Apache Maven および Quarkus Maven プラグインを使用して、Red Hat build of OptaPlanner および Quarkus アプリケーションの使用を開始できます。

前提条件

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスタマーポータル (ログインが必要) の ソフトウェアダウンロード ページから入手できます。
- Apache Maven 3.8 以降がインストールされている。Maven は Apache Maven Project の Web サイトから入手できます。

手順

1. コマンドターミナルで以下のコマンドを入力し、Maven が JDK 11 を使用していること、そして Maven のバージョンが 3.8 以上であることを確認します。

```
mvn --version
```

2. 上記のコマンドで JDK 11 が返されない場合は、JDK 11 へのパスを PATH 環境変数に追加し、上記のコマンドを再度入力します。
3. Quarkus OptaPlanner クイックスタートプロジェクトを生成するには、以下のコマンドを入力します。ここで、**redhat-0000x** は Quarkus BOM ファイルの現在のバージョンに置き換えます。

```
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" \
```

このコマンドは、**./optaplanner-quickstart** ディレクトリーで以下の要素を作成します。

- Maven の構造
- **src/main/docker** の **Dockerfile** ファイルの例
- アプリケーションの設定ファイル

表5.1 mvn io.quarkus:quarkus-maven-plugin:2.13.8.SP1-redhat-0000x:create コマンドで使用されるプロパティー

プロパティー	説明
projectGroupId	プロジェクトのグループ ID。
projectArtifactId	プロジェクトのアーティファクト ID。
extensions	このプロジェクトで使用する Quarkus 拡張のコンマ区切りリスト。Quarkus 拡張の全一覧については、特定のコマンドラインで **mvn quarkus:list-extensions** を入力します。
noExamples	テストまたはクラスを使用せずに、プロジェクト構造でプロジェクトを作成します。

projectGroupID および **projectArtifactID** プロパティーの値を使用して、プロジェクトバージョンを生成します。デフォルトのプロジェクトバージョンは **1.0.0-SNAPSHOT** です。

4. OptaPlanner プロジェクトを表示するには、OptaPlanner Quickstarts ディレクトリーに移動します。

```
cd optaplanner-quickstart
```

5. **pom.xml** ファイルを確認します。コンテンツの例を以下に示します。

```
<?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. CODE.QUARKUS.REDHAT.COM を使用した QUARKUS プラットフォームでの RED HAT BUILD OF OPTAPLANNER プロジェクトの作成

https://code.quarkus.redhat.com の Web サイトを使用して Red Hat build of OptaPlanner Quarkus の Maven プロジェクトを生成し、アプリケーションで使用する拡張機能を自動的に追加および設定できます。

本セクションでは、以下のトピックを含む OptaPlanner Maven プロジェクトを生成するプロセスについて説明します。

- アプリケーションの基本情報を指定する
- プロジェクトに追加する機能拡張の選択
- プロジェクトファイルでダウンロード可能なアーカイブの生成
- アプリケーションのコンパイルおよび起動のカスタムコマンドの使用

前提条件

- Web ブラウザーがある。

手順

1. Web ブラウザーで **https://code.quarkus.redhat.com** を開きます。
2. プロジェクトの詳細を指定します。
3. プロジェクトのグループ名を入力します。名前の形式は、Java パッケージ命名規則に従います (例: **com.example**)。
4. プロジェクトから生成された Maven アーティファクトに使用する名前を入力します (例: **code-with-quarkus**)。
5. **Build Tool** > **Maven** を選択して、作成するプロジェクトが Maven プロジェクトであることを指定します。選択するビルドツールにより、以下の項目が決定されます。
   - 生成されたプロジェクトのディレクトリー構造。
   - 生成されたプロジェクトで使用される設定ファイルの形式。
   - アプリケーションのコンパイルおよび起動用のカスタムビルドスクリプトおよびコマンド (プロジェクトの生成後に **code.quarkus.redhat.com** が表示)。

   注記

   Red Hat は、**code.quarkus.redhat.com** を使用した OptaPlanner Maven プロジェクトの作成だけをサポートします。Red Hat では、Gradle プロジェクトの生成はサポートしていません。
6. プロジェクトから生成されたアーティファクトで使用するバージョンを入力します。このフィールドのデフォルト値は **1.0.0-SNAPSHOT** です。semantic versioning の使用が推奨されますが、必要に応じて、別のタイプのバージョンを使用できます。
7. プロジェクトをパッケージ化する時に、ビルドツールが生成するアーティファクトのパッケージ名を入力します。
   パッケージ名は、Java パッケージの命名規則に従い、プロジェクトに使用するグループ名と一致するはずですが、別の名前を指定することもできます。
8. 以下の拡張を選択して、依存関係として組み込みます。
   - RESTEasy JAX-RS (quarkus-resteasy)

- RESTEasy Jackson (quarkus-resteasy-jackson)
- OptaPlanner AI 制約ソルバー (optaplanner-quarkus)
- OptaPlanner Jackson (optaplanner-quarkus-jackson)
  Red Hat は、一覧にある個別の拡張に対してさまざまなレベルのサポートを提供します。レベルは、各拡張名の横にあるラベルで示されています。
  - **SUPPORTED** 拡張: Red Hat は、実稼働環境のエンタープライズアプリケーションでの使用を完全にサポートします。
  - **TECH-PREVIEW** 拡張: Red Hat は、 テクノロジープレビュー機能のサポート範囲 に基づき、限定的に、実稼働環境でのサポートを提供します。
  - **DEV-SUPPORT** 拡張: Red Hat は、実稼働環境での使用をサポートしていません。ただし、新規アプリケーションの開発での使用に対しては、Red Hat 開発者がこれらのコア機能をサポートしています。
  - **DEPRECATED** 拡張: 同じ機能を提供する新しいテクノロジーまたは実装に置き換える予定です。
    Red Hat では、ラベル付けされていない拡張の実稼働環境での使用はサポートしていません。

9. **Generate your application** を選択して選択内容を確認し、生成されたプロジェクトを含むアーカイブのダウンロードリンクのオーバーレイ画面を表示します。オーバーレイ画面には、アプリケーションのコンパイルおよび起動に使用できるカスタムコマンドも表示されます。
10. **Download the ZIP** を選択して、生成されたプロジェクトファイルを含むアーカイブをマシンに保存します。
11. アーカイブの内容を展開します。
12. 展開したプロジェクトファイルが含まれるディレクトリーに移動します。

```
cd <directory_name>
```

13. 開発モードでアプリケーションをコンパイルして起動します。

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

5.4. QUARKUS CLI を使用した QUARKUS プラットフォームでの RED HAT BUILD OF OPTAPLANNER プロジェクトの作成

Quarkus コマンドラインインターフェイス (CLI) を使用して、Quarkus OptaPlanner プロジェクトを作成できます。

前提条件

- Quarkus CLI をインストールしている。詳細は、Building Quarkus Apps with Quarkus Command Line Interface を参照してください。

手順

1. Quarkus アプリケーションを作成します。

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

2. 利用可能な拡張機能を表示するには、以下のコマンドを入力します。

```
quarkus ext -i
```

このコマンドは、以下の拡張機能を返します。

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

3. 以下のコマンドを入力して、エクステンションをプロジェクトの **pom.xml** ファイルに追加します。

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

4. テキストエディターで **pom.xml** ファイルを開きます。ファイルの内容は以下の例のようになります。

```
<?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>
```

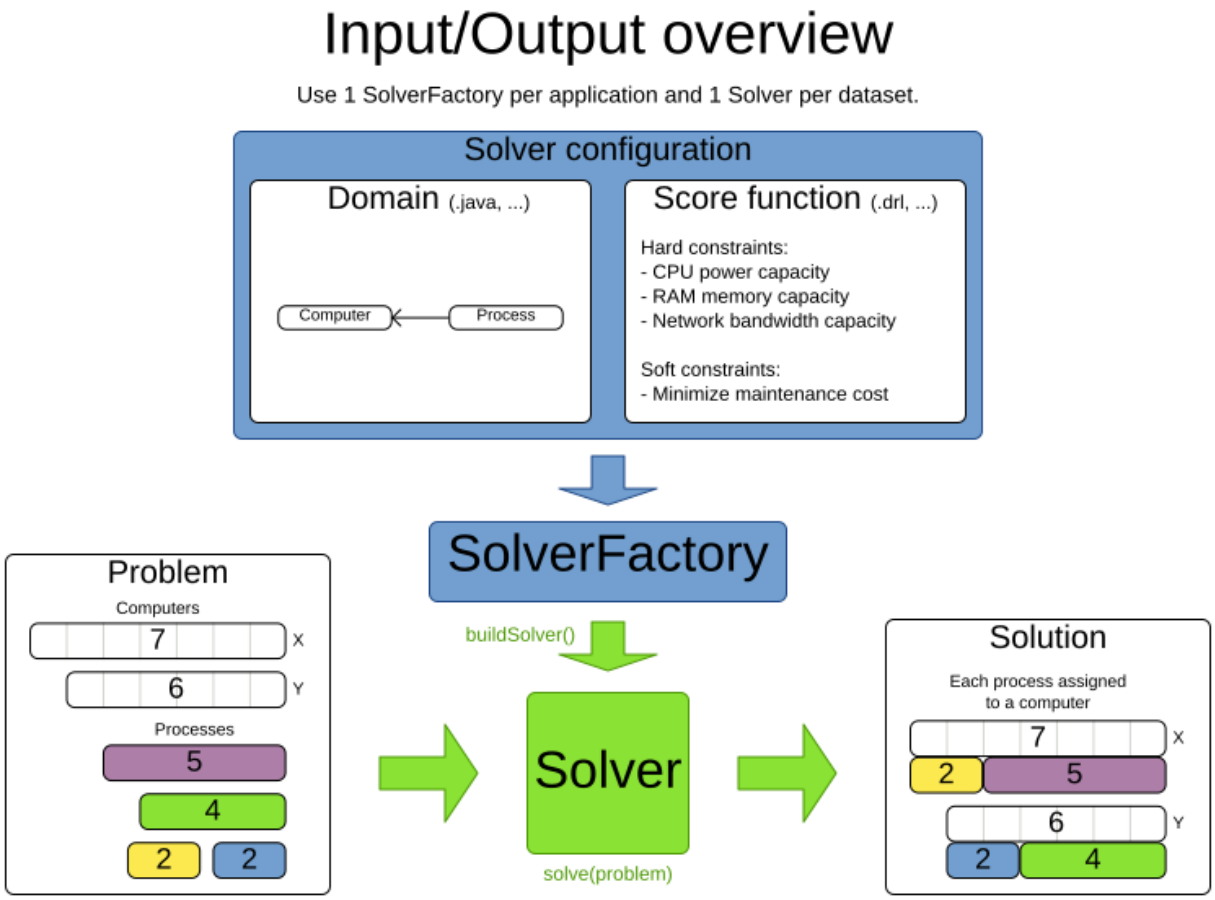
```
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${compiler-plugin.version}</version>
      <configuration>
        <parameters>${maven.compiler.parameters}</parameters>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <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>
```

パート III. RED HAT BUILD OF OPTAPLANNER のソルバー

OptaPlanner を使用して計画の問題を解決するには、次の手順を実行します。

1. **@PlanningSolution** アノテーションでアノテーションが付けられたクラスとして **プランニングの問題をモデル化**します (例: **NQueens** クラス)。
2. **Solver を設定**します (**NQueens** インスタンスの First Fit や Tabu Search Solver など)。
3. データレイヤーから**問題データセットを読み込みます** (例: Queens インスタンス)。これがプラニング問題です。
4. 見つかった最善解を返す **Solver.solve (problem)** で **解決**します。



第6章 RED HAT BUILD OF OPTAPLANNER ソルバーの設定

以下の方法を使用して、OptaPlanner のソルバーを設定できます。

- XML ファイルを使用します。
- **SolverConfig** API を使用します。
- ドメインモデルにクラスアノテーションと JavaBean プロパティーアノテーションを追加します。
- OptaPlanner がドメインにアクセスするために使用するメソッドを制御します。
- カスタムプロパティーを定義します。

6.1. XML ファイルを使用した OPTAPLANNER のソルバーの設定

各サンプルプロジェクトには、編集可能なソルバー設定ファイルがあります。**<EXAMPLE>SolverConfig.xml** ファイルは、**org.optaplanner.optaplanner-8.38.0.Final-redhat-00004/optaplanner-examples/src/main/resources/org/optaplanner/examples/<EXAMPLE>**ディレクトリーにあります。**<EXAMPLE>** は OptaPlanner サンプルプロジェクトの名前です。または、**SolverFactory.createFromXmlFile()** で ファイルから **SolverFactory** を作成することもできます。ただし、移植性の理由から、クラスパスのリソースが推奨されます。

Solver と **SolverFactory** の両方に、 **Solution_** と呼ばれるジェネリック型があります。これは、計画の問題と解決策を表すクラスです。

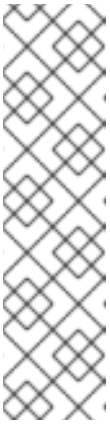
OptaPlanner を使用すると、設定を変更することで、最適化アルゴリズムを比較的簡単に切り替えることができます。

手順

1. **SolverFactory** で **Solver** インスタンスをビルドします。
2. Solver 設定の XML ファイルを設定します。
   a. モデルを定義します。
   b. スコア機能を定義します。
   c. 必要に応じて、最適化アルゴリズムを設定します。
   以下の例は、NQueens 問題に対するソルバー XML ファイルです。

```
<?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</scoreDr
```

```
l>
  </scoreDirectorFactory>

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



注記

一部の環境では、OSGi や JBoss モジュールなどの一部の環境では、JAR ファイルのソルバー設定、スコア DRL、ドメインクラスなどのクラスパスリソースが **optaplanner-core** JAR ファイルのデフォルトの **ClassLoader** で利用できない場合もあります。このような場合は、クラスの **ClassLoader** をパラメーターとして提供します。

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

3. **ClassLoader.getResource()** で定義されているクラスパスリソースとして提供されるソルバー設定 XML ファイルを使用して **SolverFactory** を設定します。

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

6.2. JAVA API を使用した OPTAPLANNER のソルバーの設定

SolverConfig API を使用して Solver を設定できます。これは特に、ランタイム時に値を動的に変更する場合に便利です。以下の例では、NQueens プロジェクトで **Solver** を構築する前のシステムプロパティーに基づいて実行時間を変更します。

```
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();
```

ソルバー設定 XML ファイルのすべての要素は、パッケージ namespace **org.optaplanner.core.config** の **Config** クラスまたは **プロパティー** として使用できます。これらの **Config** クラスは XML 形式の Java 表現です。これらのコンポーネントには、パッケージ namespace **org.optaplanner.core.impl** のランタイムコンポーネントをビルドし、それらを効率的な **Solver** に組み込むことができます。

注記

各ユーザー要求に **SolverFactory** を動的に設定するには、初期化中にテンプレート **SolverConfig** をビルドし、各ユーザー要求のコピーコンストラクターでコピーします。以下の例は、NQueens の問題でこれを実行する方法を示しています。

```
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 アノテーション

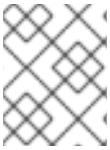
ドメインモデルのクラスは、プランニング変数などのプランニングエンティティーを指定する必要があります。以下の方法のいずれかを使用して、OptaPlanner プロジェクトにアノテーションを追加します。

- ドメインモデルにクラスアノテーションと JavaBean プロパティーアノテーションを追加します。プロパティーアノテーションは setter メソッドではなく getter メソッドに配置する必要があります。アノテーションが付けられた getter メソッドの公開は必要ありません。これは推奨される方法です。
- ドメインモデルにクラスアノテーションとフィールドアノテーションを追加します。アノテーションが付けられたフィールドはパブリックである必要はありません。

6.4. OPTAPLANNER ドメインアクセスの指定

デフォルトでは、OptaPlanner はリフレクションを使用してドメインにアクセスします。リフレクションは信頼性がありますが、直接アクセスに比べると遅くなります。または、Gizmo を使用してドメインにアクセスするように OptaPlanner を設定できます。これにより、リフレクションなしにドメインのフィールドとメソッドに直接アクセスするバイトコードが生成されます。ただし、この手法には以下の制限があります。

- プランニングアノテーションは、パブリックフィールドおよびパブリック getter でのみ指定できます。
- **io.quarkus.gizmo:gizmo** はクラスパス上にある必要があります。



注記

Gizmo がデフォルトのドメインアクセスタイプであるため、Quarkus で OptaPlanner を使用する場合、これらの制限は適用されません。

手順

Quarkus の外部にある Gizmo を使用するには、ソルバー設定で **domainAccessType** を設定します。

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

6.5. カスタムプロパティーの設定

OptaPlanner プロジェクトでは、クラスをインスタンス化し、カスタムプロパティーに明示的に言及するドキュメントを持つソルバー設定要素にカスタムプロパティーを追加できます。

前提条件

- ソルバーがあること。

手順

1. カスタムプロパティーを追加します。
   たとえば、**Easy ScoreCalculator** にキャッシュされる大きな計算があり、1 つのベンチマークでキャッシュサイズを増やす場合は、**myCacheSize** プロパティーを追加します。

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

2. カスタムプロパティーごとにパブリックセッターを追加します。これは、**ソルバーの** ビルド時に呼び出されます。

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

      private int myCacheSize = 500; // Default value

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

    ...
}
```

   boolean、 **int**、 **double**、 **BigDecimal**、 **String**、 **enums** など、ほとんどの値型がサポートされています。

第7章 OPTAPLANNER ソルバーの使用

ソルバーは、計画の問題に対する最適で最適なソリューションを見つけます。ソルバーで解決できる計画問題インスタンスは度に 1 つずつです。ソルバーは、**SolverFactory** メソッドを使用して構築されます。

```
public interface Solver<Solution_> {

    Solution_ solve(Solution_ problem);

    ...
}
```

スレッドセーフであることが **javadoc** に具体的に記載されているメソッドを除いて、ソルバーには単一のスレッドからアクセスする必要があります。**solve()** メソッドは、現在のスレッドを占有します。スレッドを占有すると、REST サービスの HTTP タイムアウトが発生する可能性があり、複数のデータセットを並行して解決するために追加のコードが必要になります。このような問題を回避するには、代わりに **SolverManager** を使用してください。

7.1. 問題の解決

ソルバーを使用して、計画の問題を解決します。

前提条件

- ソルバー設定で構築された **Solver**
- 計画問題インスタンスを表す **@PlanningSolution** アノテーション

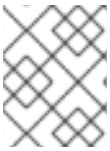
手順

計画問題を **solve()** メソッドの引数として提供します。ソルバーは、見つかった最良の解を返します。

次の例は、NQueens の問題を解決します。

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

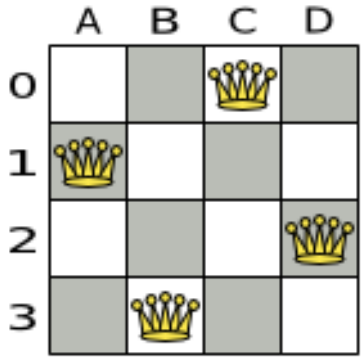
この例では、 **solve()** メソッドは、すべての **Queen** が **Row** に割り当てられた **NQueens** インスタンスを返します。



注記

solve (Solution) メソッドに指定されたソリューションインスタンスは、部分的または完全に初期化できます。これは、繰り返して計画する場合によくあります。

図7.1 8ms のフォークイーンズパズルのベストソリューション (最適なソリューションでもあります)



solve (Solution) メソッドは、問題のサイズとソルバーの設定によっては時間がかかる場合があります。**Solver** は、可能な解決策の検索スペースをインテリジェントに処理し、解決中に遭遇した最良の解決策を記憶します。問題の大きさ、**Solver** が持っているどのくらいの時間、ソルバーの設定、等々の数ある要因により、**最善の** 解決策は、**最適な** 解決策ではない可能性もあります、

注記

メソッド **solve (Solution) に** 与えられたソリューションインスタンスは **Solver** によって変更されますが、それを最良のソリューションと間違えないでください。

solve (Solution) または **getBestSolution()** メソッドによって返されたソリューションのインスタンスは、**solve (Solution)** に渡された計画インスタンスのクローンである可能性が高いです。この場合、つまりは別のインスタンスであることを意味しています。

7.2. ソルバー環境モード

ソルバー環境モードを使用すると、実装の一般的なバグを検出できます。ロギングレベルには影響しません。

ソルバーには単一のランダムインスタンスがあります。一部のソルバー設定は、他の設定よりもランダムインスタンスを多く使用します。たとえば、シミュレーテッドアニーリングアルゴリズムは乱数に大きく依存しますが、**Tabu Search** はスコアの同点を解決するために乱数にのみ依存します。環境モードは、そのランダムインスタンスのシードに影響を与えます。

ソルバー設定 XML ファイルで環境モードを設定できます。次の例では、 **FAST_ASSERT** モードを設定します。

```
<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>
```

次のリストは、ソルバー設定ファイルで使用できる環境モードについて説明しています。

- **FULL_ASSERT** モードは、すべてのアサーションをオンにします。たとえば、増分スコアの計算が移動ごとに破損していないというアサーション、Move 実装のバグ、制約、エンジン自体などでフェイルファストします。このモードは再現可能です。また、非アサートモードよりも

頻繁に **calculateScore()** メソッドを呼び出すため、煩わしいものです。**FULL_ASSERT** モードは、増分スコア計算に依存しないため、非常に低速です。

- **NON_INTRUSIVE_FULL_ASSERT** モードは、Move 実装のバグ、制約、エンジン自体などでフェイルファストするためにいくつかのアサーションをオンにします。このモードは再現可能です。非アサートモードよりも頻繁に **calculateScore()** メソッドを呼び出さないため、邪魔になりません。**NON_INTRUSIVE_FULL_ASSERT** モードは、増分スコア計算に依存しないため、非常に低速です。

- **FAST_ASSERT** モードは、undoMove のスコアが Move の前と同じであるというアサーションなど、ほとんどのアサーションをオンにして、Move 実装のバグ、制約、エンジン自体などをフェイルファストします。このモードは再現可能です。また、非アサートモードよりも頻繁に **calculateScore()** メソッドを呼び出すため、煩わしいものです。**FAST_ASSERT** モードは遅いです。**FAST_ASSERT** モードをオンにして、計画の問題を短時間実行するテストケースを作成します。

- **REPRODUCIBLE** モードは、開発中に推奨されるため、デフォルトのモードです。このモードでは、同じ OptaPlanner バージョンで 2 回実行すると、同じコードが同じ順序で実行されます。次の注意事項が当てはまる場合を除いて、これら 2 つの実行はすべてのステップで同じ結果になります。これにより、バグを一貫して再現できます。また、スコア制約のパフォーマンスの最適化など、特定のリファクタリングを実行全体で公平にベンチマークすることもできます。

注記

REPRODCIBLE モードを使用しているにもかかわらず、次の理由により、アプリケーションが完全に再現できない場合があります。

  - 特にソリューションの実装において、計画エンティティーまたは計画値のコレクションに対して、JVM 実行間で順序が一貫していないが、通常の問題の事実ではない **HashSet** または別の **コレクション** を使用する。**LinkedHashSet** に置き換えます。

  - 時間勾配に依存するアルゴリズム、特にシミュレーテッドアニーリングアルゴリズムを、終了に費やした時間と組み合わせます。割り当てられた CPU 時間大きな違いがあると、時間勾配値に影響を与えます。シミュレーテッドアニーリングアルゴリズムをレイトアクセプタンスアルゴリズムに置き換えるか、終了に費やした時間をステップカウント終了に置き換えます。

- **REPRODUCIBLE** モードは、**NON_REPRODUCIBLE** モードよりもわずかに遅くなる可能性があります。実稼働環境で再現性の恩恵を受けることができる場合は、実稼働でこのモードを使用してください。実際には、 **REPRODUCIBLE** モードでは、シードが指定されていない場合、デフォルトの固定ランダムシードが使用され、ワークスティーリングなどの特定の同時実行の最適化も無効になります。

- **NON_REPRODUCIBLE** モードは、**REPRODUCIBLE** モードよりもわずかに高速です。デバッグやバグ修正が困難になるため、開発中の使用は避けてください。実稼働環境で再現性が重要でない場合は、実稼働で **NON_REPRODUCIBLE** モードを使用してください。実際には、シードが指定されていない場合、このモードは固定ランダムシードを使用しません。

7.3. OPTAPLANNER ソルバーのログレベルの変更

OptaPlanner ソルバーのログレベルを変更して、ソルバーアクティビティーを確認できます。次のリストは、さまざまなログレベルについて説明しています。

- error:**RuntimeException** として呼び出し元のコードに throw されるエラーを除いて、エラーをログに記録します。
  エラーが発生した場合、OptaPlanner は通常は短時間で失敗します。呼び出し元のコードに詳細なメッセージを含む **RuntimeException の** サブクラスを出力します。ログメッセージの重複を避けるために、エラーとしてログに記録されません。**呼び出し元のコードがその RuntimeException を** 明示的にキャッチして排除しない限り、**スレッドのデフォルトの `ExceptionHandler** はとにかくそれをエラーとしてログに記録します。その間、コードはさらに害を及ぼしたり、エラーを難読化したりすることで中断されます。

- **警告**: 疑わしい状況をログに記録します

- **info**: すべてのフェーズとソルバー自体をログに記録します

- **デバッグ**: すべてのフェーズのすべてのステップをログに記録します

- **トレース**: すべてのフェーズのすべてのステップのすべての動きをログに記録します

注記

トレース ログを指定すると、パフォーマンスが大幅に低下します。ただし、**トレース** ロギングは、ボトルネックを発見するための開発中に非常に重要です。

デバッグ ログでさえ、レイトアクセプタンスやシミュレーテッドアニーリングなどの高速ステッピングアルゴリズムではパフォーマンスが大幅に低下する可能性がありますが、タブーサーチなどの低速ステッピングアルゴリズムでは低下しません。

trace`と **デバッグ** ロギングの両方が、ほとんどのアペンダーでのマルチスレッド解決で輻輳を引き起こします。

Eclipse では、コンソールへの **デバッグ** ログにより、スコア計算速度が 1 秒あたり 10000 を超える輻輳が発生する傾向があります。IntelliJ も Maven コマンドラインもこの問題に悩まされていません。

手順

ロギングレベルを **デバッグ** ロギングに設定して、フェーズがいつ終了し、どのくらいの速さでステップが実行されるかを確認します。

次の例は、デバッグログからの出力を示しています。

```
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).
```

費やされた時間の値はすべてミリ秒単位です。

すべてが SLF4J に記録されます。これは、すべてのログメッセージを Logback、Apache Commons Logging、Log4j、または java.util.logging に委任する単純なログファサードです。選択したロギングフレームワークのロギングアダプターに依存関係を追加します。

7.4. LOGBACK を使用して OPTAPLANNER ソルバーアクティビティーをログに記録する

Logback は、OptaPlanner で使用するために推奨されるロギングフレームワークです。Logback を使用して、OptaPlanner ソルバーアクティビティーをログに記録します。

前提条件

- OptaPlanner プロジェクトがあります。

手順

1. 次の Maven 依存関係を OptaPlanner プロジェクトの **pom.xml** ファイルに追加します。

注記

ブリッジの依存関係を追加する必要はありません。

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

2. 次の例に示すように、 **logback.xml** ファイルの **org.optaplanner** パッケージのログレベルを設定します。ここで、**<LEVEL>** はにリストされているログレベルです。「Logback を使用して OptaPlanner ソルバーアクティビティーをログに記録する」 。

```
<configuration>

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

 ...

</configuration>
```

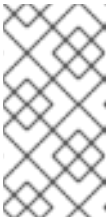
3. オプション: 複数の **ソルバー** インスタンスが同時に実行されている可能性があるマルチテナントアプリケーションがある場合は、各インスタンスのログを別々のファイルに分割します。

   a. **solve()** 呼び出しをマップされた診断コンテキスト (MDC) で囲みます。

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

b. **${tenant.name}** ごとに異なるファイルを使用するようにロガーを設定します。たとえば、**logback.xml** ファイルで **SiftingAppender** を使用します。

```
<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>
```



注記

複数のソルバーまたは 1 つのマルチスレッドソルバーを実行する場合、コンソールを含むほとんどのアペンダーは、**デバッグ** および **トレース** ログで輻輳を引き起こします。この問題を回避するには、非同期アペンダーに切り替えるか、**デバッグ** ログをオフにします。

4. OptaPlanner が新しいレベルを認識しない場合は、一時的にシステムプロパティー **-Dlogback.LEVEL=true** を追加してトラブルシューティングします。

7.5. LOG4J を使用して OPTAPLANNER ソルバーアクティビティーをログに記録する

すでに Log4J を使用していて、より高速な後継である Logback に切り替えたくない場合は、Log4J 用に OptaPlanner プロジェクトを設定できます。

前提条件

- OptaPlanner プロジェクトがあります
- Log4J ロギングフレームワークを使用しています

手順

1. ブリッジの依存関係をプロジェクトの **pom.xml** ファイルに追加します。

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

2. 次の例に示すように、 **log4j.xml** ファイルのパッケージ **org.optaplanner** でログレベルを設定します。ここで、**<LEVEL>** はにリストされているログレベルです。「Logback を使用して OptaPlanner ソルバーアクティビティーをログに記録する」 。

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

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

 ...

</log4j:configuration>
```

3. オプション: 複数の **ソルバー** インスタンスが同時に実行されている可能性があるマルチテナントアプリケーションがある場合は、各インスタンスのログを別々のファイルに分割します。

   a. **solve()** 呼び出しをマップされた診断コンテキスト (MDC) で囲みます。

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

   b. **${tenant.name}** ごとに異なるファイルを使用するようにロガーを設定します。たとえば、**logback.xml** ファイルで **SiftingAppender** を使用します。

```
<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>
```

注記

複数のソルバーまたは 1 つのマルチスレッドソルバーを実行する場合、コンソールを含むほとんどのアペンダーは、**デバッグ** および **トレース** ログで輻輳を引き起こします。この問題を回避するには、非同期アペンダーに切り替えるか、**デバッグ** ログをオフにします。

7.6. ソルバーの監視

OptaPlanner は、Java アプリケーション用のメトリック計測ライブラリーである Micrometer を介してメトリックを公開します。一般的な監視システムで Micrometer を使用して、OptaPlanner ソルバーを監視できます。

7.6.1. Micrometer 用の Quarkus OptaPlanner アプリケーションの設定

OptaPlanner Quarkus アプリケーションを Micrometer および指定された監視システムを使用するように設定するには、Micrometer 依存関係を **pom.xml** ファイルに追加します。

前提条件

- Quarkus OptaPlanner アプリケーションがあります。

手順

1. 次の依存関係をアプリケーションの **pom.xml** ファイルに追加します。ここで **<MONITORING_SYSTEM>** は Micrometer と Quarkus でサポートされている監視システムです。

注記

Prometheus は現在、Quarkus でサポートされている唯一の監視システムです。

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

2. アプリケーションを開発モードで実行するには、次のコマンドを入力します。

```
mvn compile quarkus:dev
```

3. アプリケーションのメトリックを表示するには、ブラウザーに次の URL を入力します。

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

7.6.2. Micrometer 用の Spring Boot OptaPlanner アプリケーションの設定

Micrometer と指定された監視システムを使用するように Spring Boot OptaPlanner アプリケーションを設定するには、**Pom.xml** ファイルに Micrometer 依存関係を追加します。

前提条件

- Spring Boot OptaPlanner アプリケーションがあります。

手順

1. 次の依存関係をアプリケーションの **pom.xml** ファイルに追加します。ここで **<MONITORING_SYSTEM>** は Micrometer と Spring Boot でサポートされている監視システムです。

```
<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. アプリケーションの **application.properties** ファイルに設定情報を追加します。詳細は、Micrometer Web サイトを参照してください。

3. アプリケーションを実行するには、以下のコマンドを入力します。

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

4. アプリケーションのメトリックを表示するには、ブラウザーに次の URL を入力します。
http://localhost:8080/actuator/metrics

注記

次の URL を Prometheus スクレイパーパスとして使用します:
http://localhost:8080/actuator/prometheus

7.6.3. Micrometer 用のプレーンな Java OptaPlanner アプリケーションの設定

Micrometer を使用するようにプレーンな Java OptaPlanner アプリケーションを設定するには、Micrometer の依存関係と、選択した監視システムの設定情報をプロジェクトの **POM.XML** ファイルに追加する必要があります。

前提条件

- プレーンな Java OptaPlanner アプリケーションがあります。

手順

1. 次の依存関係をアプリケーションの **pom.xml** ファイルに追加します。ここで、**<MONITORING_SYSTEM>** は Micrometer で設定された監視システムであり、**<VERSION>** は使用している Micrometer のバージョンです。

```
<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. プロジェクトの **pom.xml** ファイルの先頭に、監視システムの Micrometer 設定情報を追加します。詳細は、Micrometer Web サイトを参照してください。

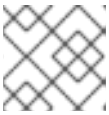
3. 設定情報の下に次の行を追加します。ここで、**<MONITORING_SYSTEM>** は追加した監視システムです。

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

次の例は、Prometheus 監視システムを追加する方法を示しています。

```
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. 監視システムを開いて、OptaPlanner プロジェクトのメトリックを表示します。次のメトリックが公開されます。



注記

メトリックの名前と形式は、レジストリーによって異なります。

- **optaplanner.solver.errors.total**: 測定開始以降に解決中に発生したエラーの総数。
- **optaplanner.solver.solve-length.active-count**: 現在解いているソルバーの数。
- **optaplanner.solver.solve-length.seconds-max**: 現在アクティブなソルバーの実行時間が最も長い実行時間。
- **optaplanner.solver.solve-length.seconds-duration-sum**: アクティブな各ソルバーの解決時間の合計。たとえば、アクティブなソルバーが 2 つあり、一方が 3 分間実行され、もう一方が 1 分間実行されている場合、合計計算時間は 4 分です。

7.6.4. 追加メトリクス

より詳細な監視を行うには、ソルバー設定で OptaPlanner を設定して、パフォーマンスコストで追加のメトリックを監視できます。次の例では、**BEST_SCORE** および **SCORE_CALCULATION_COUNT** メトリクスを使用しています。

```
<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>
```

この設定では、次のメトリックを有効にできます。

- **SOLVE_DURATION** (デフォルトで有効、マイクロメータメーター ID: **optaplanner.solver.solve.duration**): 最長のアクティブソルバーの解決時間、アクティブソルバーの数、アクティブなすべてのソルバーの累積時間を測定します。

- **ERROR_COUNT** (デフォルトで有効、マイクロメーターメーター ID: **optaplanner.solver.errors**): 解決中に発生したエラーの数を測定します。

- **SCORE_CALCULATION_COUNT** (デフォルトで有効、マイクロメーターメーター ID: **optaplanner.solver.score.calculation.count**): OptaPlanner が実行したスコア計算の数を測定します。

- **BEST_SCORE** (マイクロメーターメーター ID: **optaplanner.solver.best.score.***): OptaPlanner がこれまでに見つけた最適解のスコアを測定します。スコアのレベルごとに個別のメーターがあります。たとえば、**HardSoftScore** の場合、**optaplanner.solver.best.score.hard.score** および **optaplanner.solver.best.score.soft.score** メーターがあります。

- **STEP_SCORE** (マイクロメーターメーター ID: **optaplanner.solver.step.score.***): OptaPlanner が実行する各ステップのスコアを測定します。スコアのレベルごとに個別のメーターがあります。たとえば、**HardSoftScore** の場合、**optaplanner.solver.step.score.hard.score** および **optaplanner.solver.step.score.soft.score** メーターがあります。

- **BEST_SOLUTION_MUTATION** (マイクロメーターメーター ID: **optaplanner.solver.best.solution.mutation**): 連続する最適解の間で変更された計画変数の数を測定します。

- **MOVE_COUNT_PER_STEP** (マイクロメータメーター ID: **optaplanner.solver.step.move.count**): ステップで評価された移動の数を測定します。

- **MEMORY_USE** (マイクロメーターメーター ID: **jvm.memory.used**): JVM 全体で使用されるメモリーの量を測定します。このメトリクスは、ソルバーが使用するメモリーの量を測定するものではありません。同じ JVM 上の 2 つのソルバーは、このメトリックに対して同じ値を報告します。

- **CONSTRAINT_MATCH_TOTAL_BEST_SCORE** (マイクロメーターメーター ID: **optaplanner.solver.constraint.match.best.score.***): OptaPlanner がこれまでに見つけた最適解に対する各制約のスコアの影響を測定します。スコアのレベルごとに個別のメーターがあり、各制約のタグが付いています。たとえば、パッケージ com.example の制約 Minimize Cost の **HardSoftScore** には、 **optaplanner.solver.constraint.match.best.score.hard.score** と **optaplanner.solver.constraint.match.best.score.soft.score** があります。これらは、タグ "constraint.package=com.example" と "constraint.name=Minimize Cost" を持ちます。

- **CONSTRAINT_MATCH_TOTAL_STEP_SCORE** (マイクロメーターメーター ID: **optaplanner.solver.constraint.match.step.score.***): 現在のステップに対する各制約のスコアの影響を測定します。スコアのレベルごとに個別のメーターがあり、各制約のタグが付いています。たとえば、パッケージ com.example の制約 Minimize Cost の **HardSoftScore** には、**optaplanner.solver.constraint.match.step.score.hard.score** と **optaplanner.solver.constraint.match.step.score.soft.score** があります。これらには、タグ "constraint.package=com.example" と "constraint.name=Minimize Cost" があります。

- **PICKED_MOVE_TYPE_BEST_SCORE_DIFF** (マイクロメーターメーター ID: **optaplanner.solver.move.type.best.score.diff.***): 特定の移動タイプが最適解をどの程度改善するかを測定します。スコアのレベルごとに個別のメーターがあり、移動タイプのタグが付いています。たとえば、プロセスのコンピューターの **HardSoftScore** と **ChangeMove** に

は、**optaplanner.solver.move.type.best.score.diff.hard.score** と **optaplanner.solver.move.type.best.score.diff.soft.score** メーターがあります。これには、**move.type=ChangeMove(Process.computer)** タグがあります。

- **PICKED_MOVE_TYPE_STEP_SCORE_DIFF** (マイクロメーターメーター ID: **optaplanner.solver.move.type.step.score.diff.***): 特定の移動タイプが最適解をどの程度改善するかを測定します。スコアのレベルごとに個別のメーターがあり、移動タイプのタグが付いています。たとえば、プロセスのコンピューターの **HardSoftScore** と **ChangeMove** には、**optaplanner.solver.move.type.step.score.diff.hard.score** と **optaplanner.solver.move.type.step.score.diff.soft.score**メーターがあります。これには、タグ **move.type=ChangeMove(Process.computer)** が含まれます。

7.7. 乱数ジェネレーターの設定

多くのヒューリスティックおよびメタヒューリスティックは、移動の選択、スコアの結びつきの解決、確率ベースの移動の受け入れなどを疑似乱数ジェネレーターに依存しています。解決中に、同じランダムインスタンスが再利用され、再現性、パフォーマンス、およびランダム値の均一な分布が向上します。

ランダムシードは、疑似乱数ジェネレータを初期化するために使用される番号です。

手順

1. オプション: ランダムインスタンスのランダムシードを変更するには、 **randomSeed** を指定します。

```
<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. オプション: 疑似乱数ジェネレーターの実装を変更するには、以下のソルバー設定ファイルにリストされている **randomType** プロパティーの値を指定します。ここで、**<RANDOM_NUMBER_GENERATOR>** は疑似乱数ジェネレーターです。

```
<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>
```

次の疑似乱数ジェネレータがサポートされています。

- **JDK** (デフォルト): 標準の乱数ジェネレーターの実装 (**java.util.Random**)
- **MERSENNE_TWISTER:**コモンズ数学 によって乱数ジェネレータの実装
- **WELL512A**, **WELL1024A**, **WELL19937A**, **WELL19937C**, **WELL44497A** と **WELL44497B**: Commons Mathによる乱数ジェネレータの実装

ほとんどのユースケースでは、 **randomType** プロパティーの値は、複数のデータセットでの最良のソリューションの平均品質に大きな影響を与えません。

第8章 OPTAPLANNER SOLVERMANAGER

SolverManager は、REST およびその他のエンタープライズサービスの計画問題の解決を簡素化するための 1 つ以上の **ソルバー** インスタンスのファサードです。

Solver.solve (...) メソッドとは異なり、**SolverManager** は、次の特性があります。

- **SolverManager.solve (…)** はすぐに戻ります。呼び出し元のスレッドをブロックせずに、非同期解決の問題をスケジュールします。これにより、HTTP およびその他のテクノロジーのタイムアウトの問題が回避されます。
- **SolverManager.solve (…)** は、同じドメインの複数の計画問題を並行して解決します。

内部的には、 **SolverManager** は、**Solver.solve (…)** を呼び出すソルバースレッドのスレッドプールと、最適なソリューション変更イベントを処理するコンシューマースレッドのスレッドプールを管理します。

Quarkus と SpringBoot では、 **SolverManager** インスタンスがコードに自動的に挿入されます。Quarkus または SpringBoot 以外のプラットフォームを使用している場合は、 **create (…)** メソッドを使用して **SolverManager** インスタンスをビルドします。

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

SolverManager.solve (…) メソッドに送信される各問題には、一意の問題 ID が必要です。後で **getSolverStatus (problemId)** または **terminateEarly (problemId)** を呼び出すと、その問題 ID を使用して計画の問題を区別します。問題 ID は、**Long** 、 **String** 、 **java.util.UUID** などのイミュータブルなクラスである必要があります。

SolverManagerConfig クラスには、並行して実行されるソルバーの数を制御する **parallelSolverCount** プロパティーがあります。たとえば、 **parallelSolverCount** プロパティー `が **4** に設定されていて、5 つの問題を送信すると、4 つの問題がすぐに解決を開始し、最初の問題の 1 つが終了すると 5 番目の問題が開始します。これらの問題がそれぞれ 5 分間解決した場合、5 番目の問題は 10 分で終了します。デフォルトでは、 **parallelSolverCount** は **AUTO** に設定されており、ソルバーの **moveThreadCount** に関係なく、CPU コアの半分に解決されます。

最適なソリューションを取得するには、終了を解決した後、通常は **SolverJob.getFinalBestSolution()** を使用します。

```
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 ...;
}
```

ただし、ユーザーがソリューションを必要とする前にバッチの問題を解決する方法と、ユーザーがソリューションを積極的に待っている間にライブで解決する方法の両方について、より良いアプローチがあります。

現在の **SolverManager** 実装は単一のコンピューターノードで実行されますが、将来の作業は、クラウド全体にソルバーの負荷を分散することを目的としています。

8.1. 問題のバッチ解決

バッチ解決とは、複数のデータセットを並行して解決することです。バッチ解決は夜間処理で特に役立ちます。

- 通常、深夜には問題の変化はほとんどまたはまったくありません。一部の組織は期限を強制します。たとえば、**深夜零時までに休日のリクエストを送信する**といったものです。
- ソルバーは、結果を待つ人が誰もいないため、CPU リソースが安価であることが多いため、はるかに長く、多くの場合は数時間実行できます。
- 翌営業日に従業員が職場に到着したときにソリューションが利用できます。

手順

parallelSolverCount によって制限した上で並列バッチでの問題解決するには、各データセットのためには、以下のクラスによって作成した以下の各データについて **solve (...)** を呼び出します。

```
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. 解決して進捗状況を確認する

ユーザーがソリューションを待っている間にソルバーが実行されている場合、ユーザーは結果を受け取るまでに数分または数時間待つ必要がある場合があります。すべてが順調に進んでいることをユーザーに保証するために、これまでに達成された最良の解決策と最良のスコアを表示して進捗状況を示します。

手順

1. 中間の最良のソリューションを処理するには、**solveAndListen(...) を** 使用します。

```
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);
    }

}
```

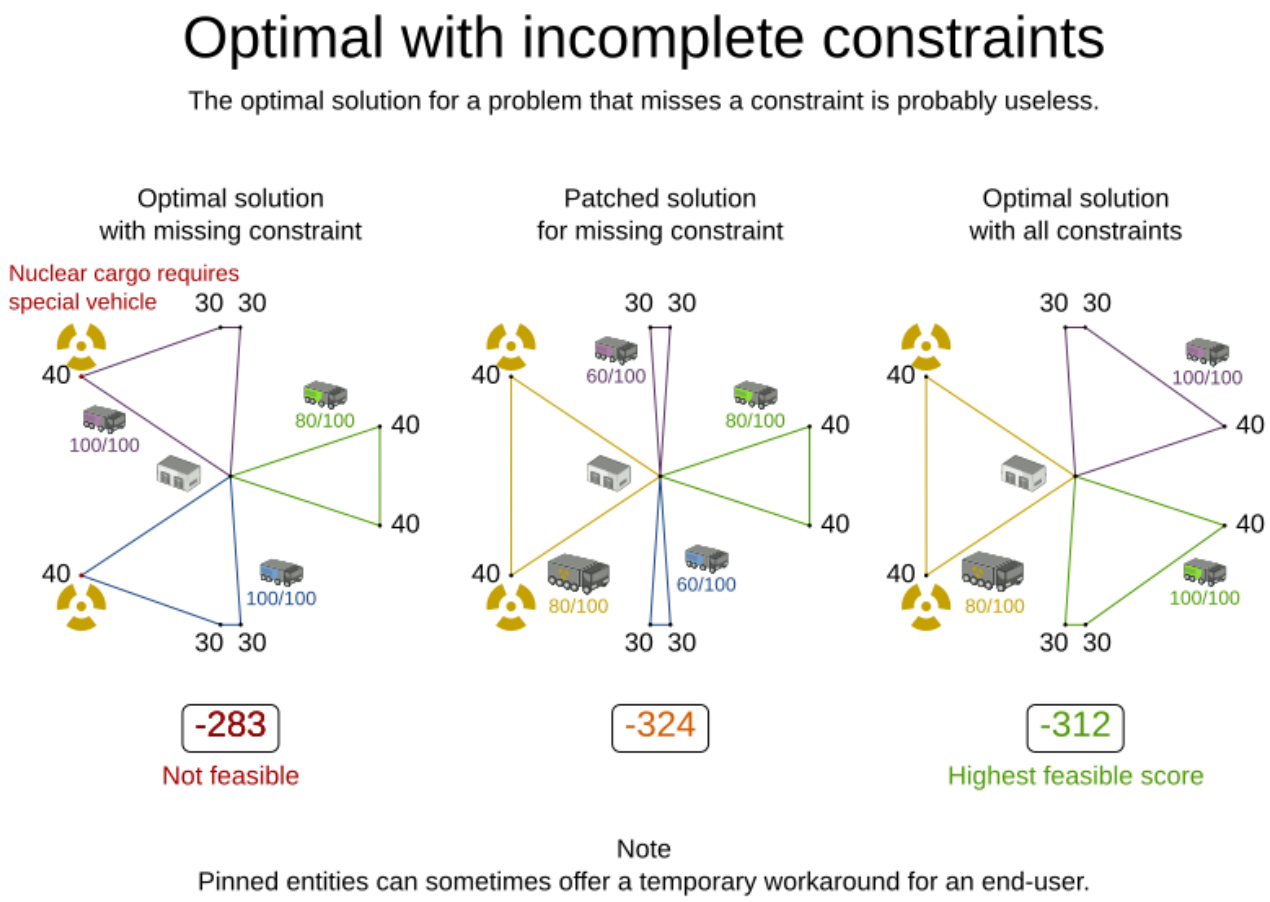
この実装は、データベースを使用して、データベースをポーリングする UI と通信します。より高度な実装は、最適なソリューションを UI またはメッセージングキューに直接プッシュします。

2. ユーザーが中間の最良の解決策に満足し、より良い解決策をこれ以上待ちたくない場合は、**SolverManager.terminateEarly (problemId)** を呼び出します。

パート IV. OPTAPLANNER スコアの計算

すべての **@PlanningSolution** クラスにはスコアがあります。スコアは、2 つのソリューションを比較する客観的な方法です。スコアが高いソリューションほど優れています。ソルバーは、考えられるすべての解の中から最高スコアのソリューションを見つけることを目的としています。**最良の解決策** は、ソルバーが解決中に遭遇した最高スコアのソリューションであり、これが **最適解** である可能性があります。

OptaPlanner は、どのソリューションがビジネスに最適であるかを自動的に認識できないため、ビジネス要件に従って、指定された **@PlanningSolution** インスタンスのスコアを計算する方法を OptaPlanner に指示する必要があります。次の図に示すように、重要なビジネス制約を忘れたり、実装できない場合、このソリューションはおそらく役に立ちません。



第9章 OPTAPLANNER のビジネス制約

ビジネス制約は、シナリオ内の条件を制限するために使用されます。条件は、既存のビジネス契約、リソースの可用性、従業員の好み、またはビジネスルールに基づく場合があります。OptaPlanner でビジネス制約を実装するには、ビジネス制約をスコア制約として形式化する必要があります。OptaPlanner で使用できる次のスコアプロパティーは、柔軟なソリューションを提供します。

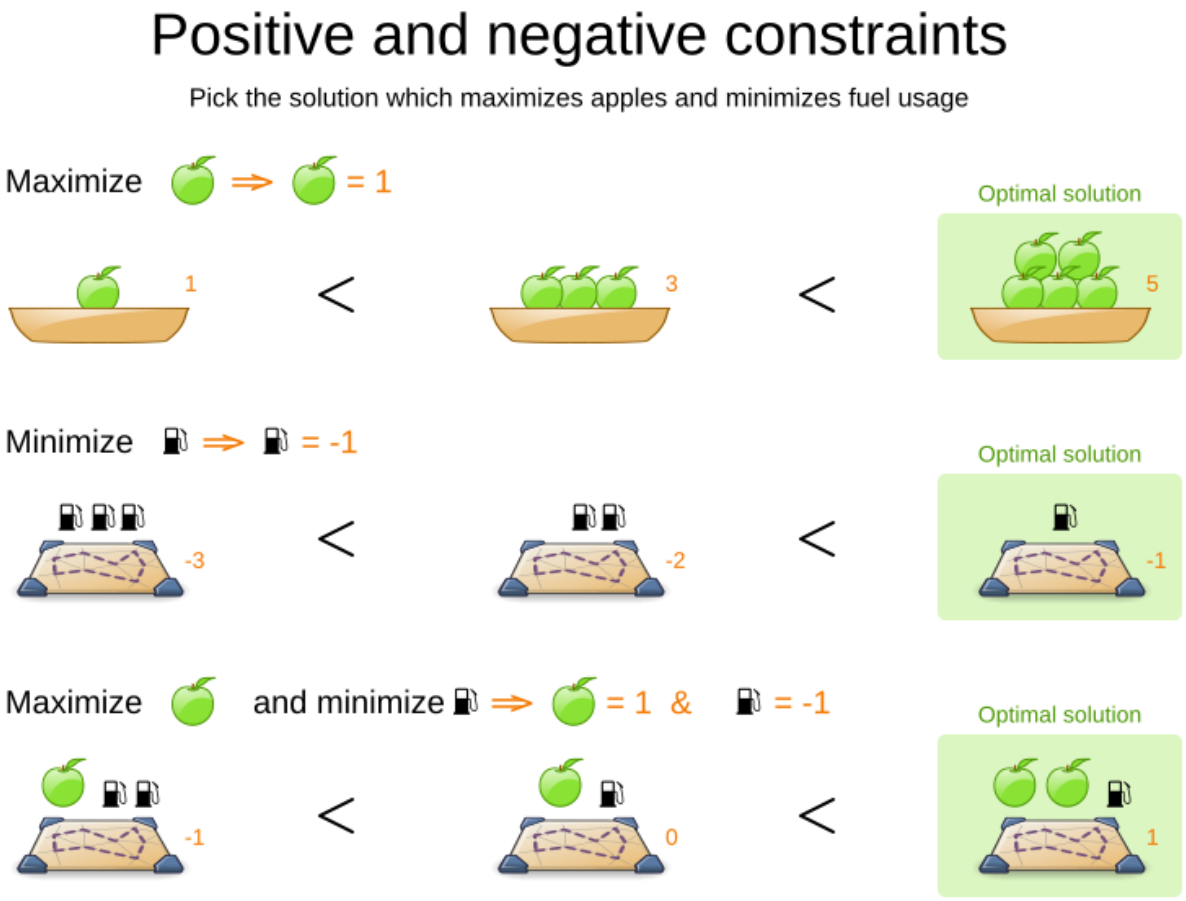
- **Score signum**: 制約タイプを正または負にします
- **スコアの重み**: 制約タイプにコストまたは利益を設定します。
- **スコアレベル (ハード、ソフトなど)**: 制約タイプのグループに優先順位を付けます。

注記

ビジネスがすべてのスコア制約を事前に知っているとは考えないでください。最初のリリース後にスコア制約が追加、変更、または削除されることが予想されます。

9.1. マイナスおよびプラスのスコア制約

すべてのスコア手法は制約に基づいています。制約には、**ソリューション内のリンゴの収穫量を最大化する** などの単純なパターン、またはより複雑なパターンを使用できます。制約は負または正のいずれかです。正の制約は、最大化したい制約です。負の制約は、最小化したい制約です。



このイメージは、制約が正か負かに関係なく、最適解が常に最高のスコアを持つことを示しています。

ほとんどの計画問題には負の制約しかないため、負のスコアになります。この場合、スコアは破られた

負の制約の重みの合計であり、完全スコアは 0 です。たとえば、N クイーン問題では、スコアは、互いに攻撃できるクイーンペアの数のマイナスになります。同じスコアレベルであっても、負の制約と正の制約を組み合わせることができます。

ネガティブな制約が破られるか、ポジティブな制約が満たされるために、特定のプランニングエンティティーセットに対して制約がアクティブ化されることを、**制約の一致** と呼びます。

9.2. スコア制約の重み

すべてのスコア制約が同じように重要であるわけではありません。1 つの制約を 1 回破るのが、別の制約を複数回破るのと同じくらい悪い場合、これら 2 つの制約は、同じスコアレベルであっても異なる重みを持ちます。

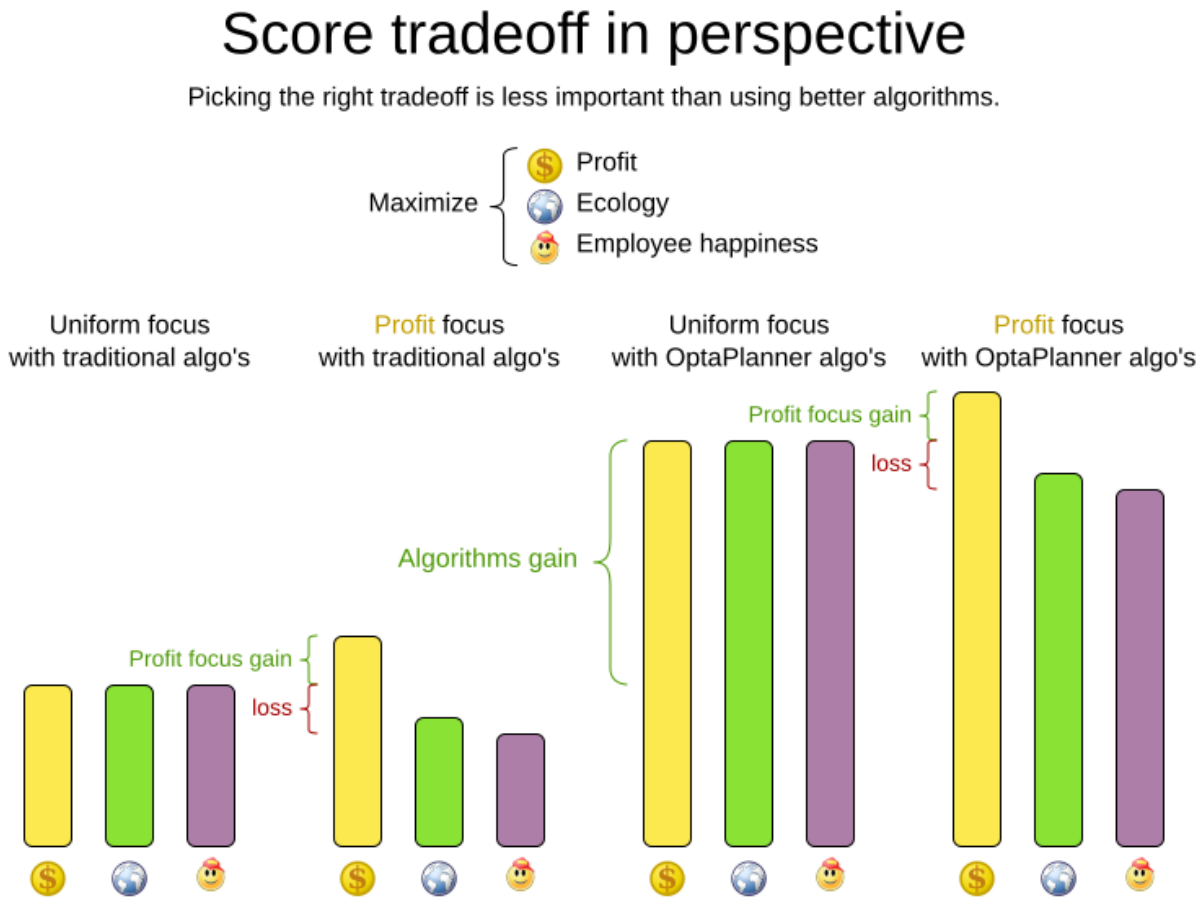
すべてにコストを割り当てることができるユースケースでは、スコアの重み付けが簡単です。この場合、正の制約により収益が最大化され、負の制約により経費が最小化され、合わせて利益が最大化されます。

あるいは、社会的公平性を確保するためにスコアの重み付けもよく使用されます。たとえば、休日を希望した看護師は、大晦日には通常の日よりも高い料金を支払います。

制約一致の重みは、関係する計画エンティティーによって異なります。たとえば、クラウドバランシング問題では、アクティブなコンピューターのソフト制約一致の重みはそのコンピューターのメンテナンスコストであり、これはコンピューターごとに異なります。

制約に適切な重みを付けることは、他の制約に対して選択とトレードオフを行うことになるため、多くの場合、分析上の決定が困難になります。利害関係者が異なれば、優先順位も異なります。

実装の開始時に制約の重みを議論して時間を無駄にしないでください。代わりに、**@constraintConfiguration** アノテーションを追加し、ユーザーが UI を通じて変更できるようにします。次の図に示すように、不正確な重み付けは、平凡なアルゴリズムよりも被害が少なくなります。

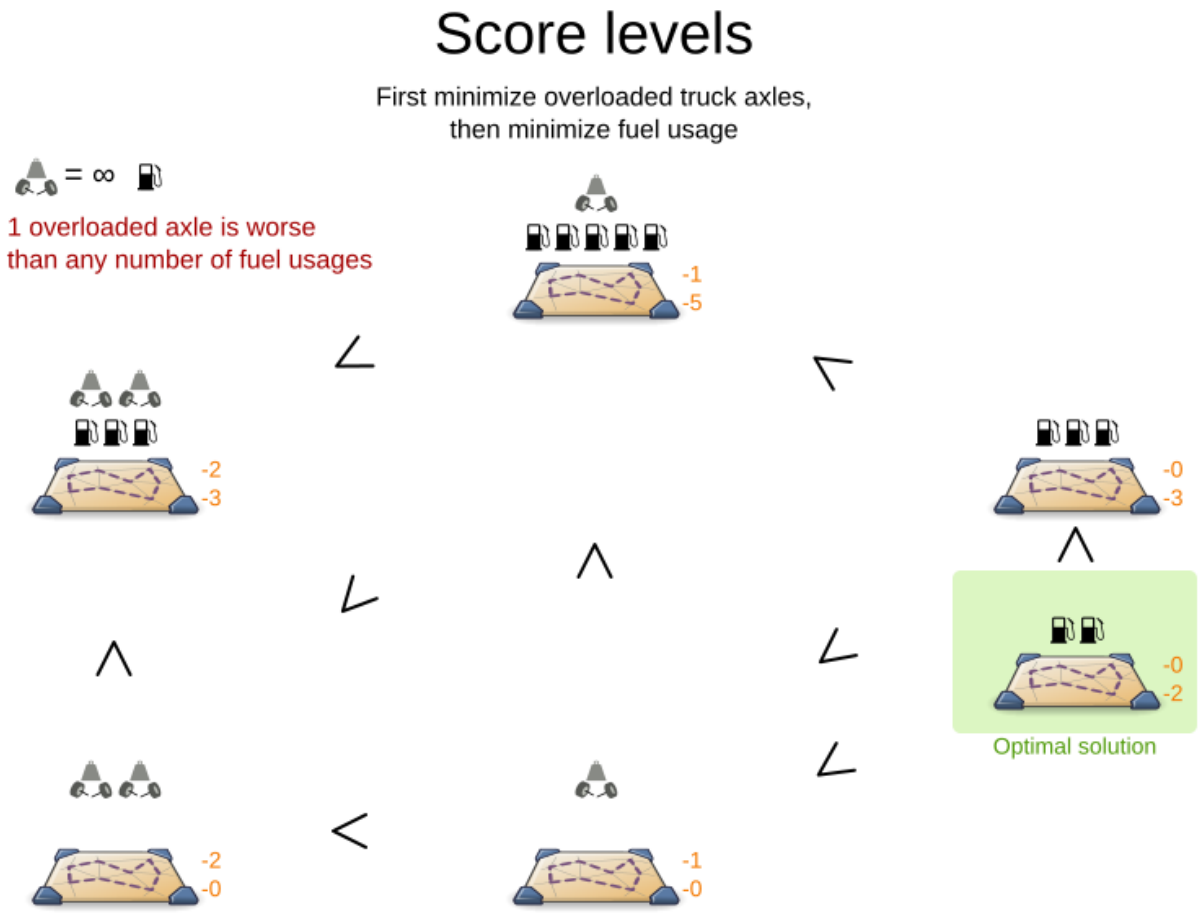


ほとんどのユースケースでは、**HardSoftScore** などの **int** 重みを持つスコアを使用します。

9.3. スコア制約レベル

場合によっては、スコア制約が何度破られたとしても、スコア制約が別のスコア制約よりも優先されることがあります。この場合、それらのスコア制約は異なるレベルにあります。たとえば、看護師は物理的現実の制約のため、同時に 2 つのシフトを勤務することができないため、この制約は看護師の幸福に関するすべての制約よりも優先されます。

ほとんどのユースケースには、ハードとソフトの 2 つのスコアレベルしかありません。2 つのスコアのレベルが順番に比較されます。最初に最初のスコアレベルが比較されます。2 つのスコアが異なる場合、残りのスコアレベルは無視されます。たとえば、**0** 個のハード制約と **1000000** 個のソフト制約を破るスコアは、**1** つのハード制約と **0** 個のソフト制約を破るスコアよりも優れています。

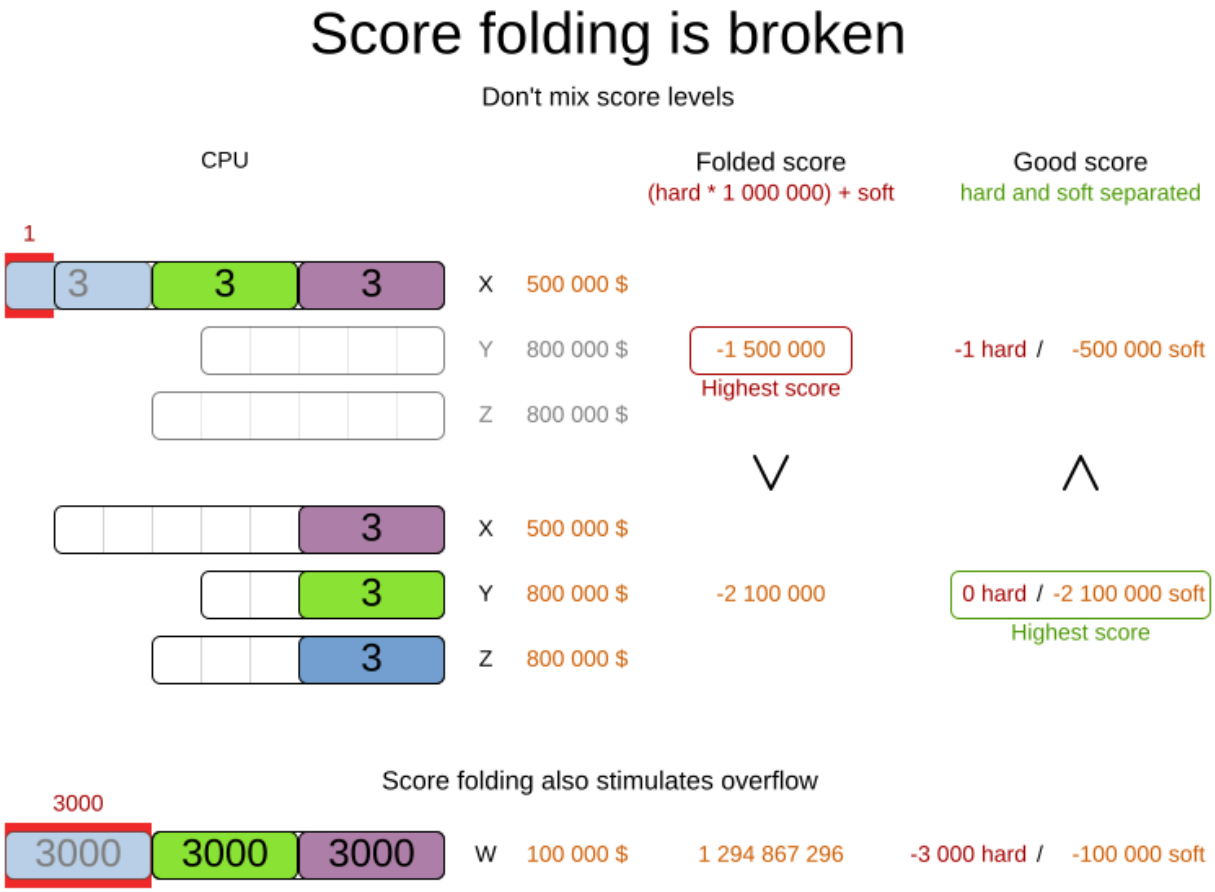


2 つ以上のスコアレベルがある場合、厳しい制約が破られなければ、スコアは **実現可能** です。

注記

デフォルトでは、OptaPlanner はすべての計画変数に計画値を割り当てます。実行可能な解決策がない場合、最適な解決策は実行不可能であることを意味します。一部の計画エンティティーを未割り当てのままにするには、過剰制約計画を適用します。

制約ごとに、スコアレベル、スコアの重み、およびスコアの符号を選択する必要があります。たとえば、**-1soft** は、スコアレベルが **Soft**、重みが **1**、および負の符号を持ちます。ビジネスで実際に異なるスコアレベルが必要な場合は、大きな制約の重みを使用しないでください。**スコアの折りたたみ** として知られるこの回避策は壊れています。



注記

ビジネスでは、ハード制約は破ることができないため、重みはすべて同じであり、したがって重みは問題ではない、と言われるかもしれません。これは事実ではありません。特定のデータセットに対して実行可能なソリューションが存在しない場合、企業は実行不可能性が最も低いソリューションを使用して、不足しているビジネスリソースの数を推定できます。たとえば、クラウドのバランスの問題では、実行不可能性が最も低い解決策によって、新しいコンピューターが何台必要であるかを明らかにできます。

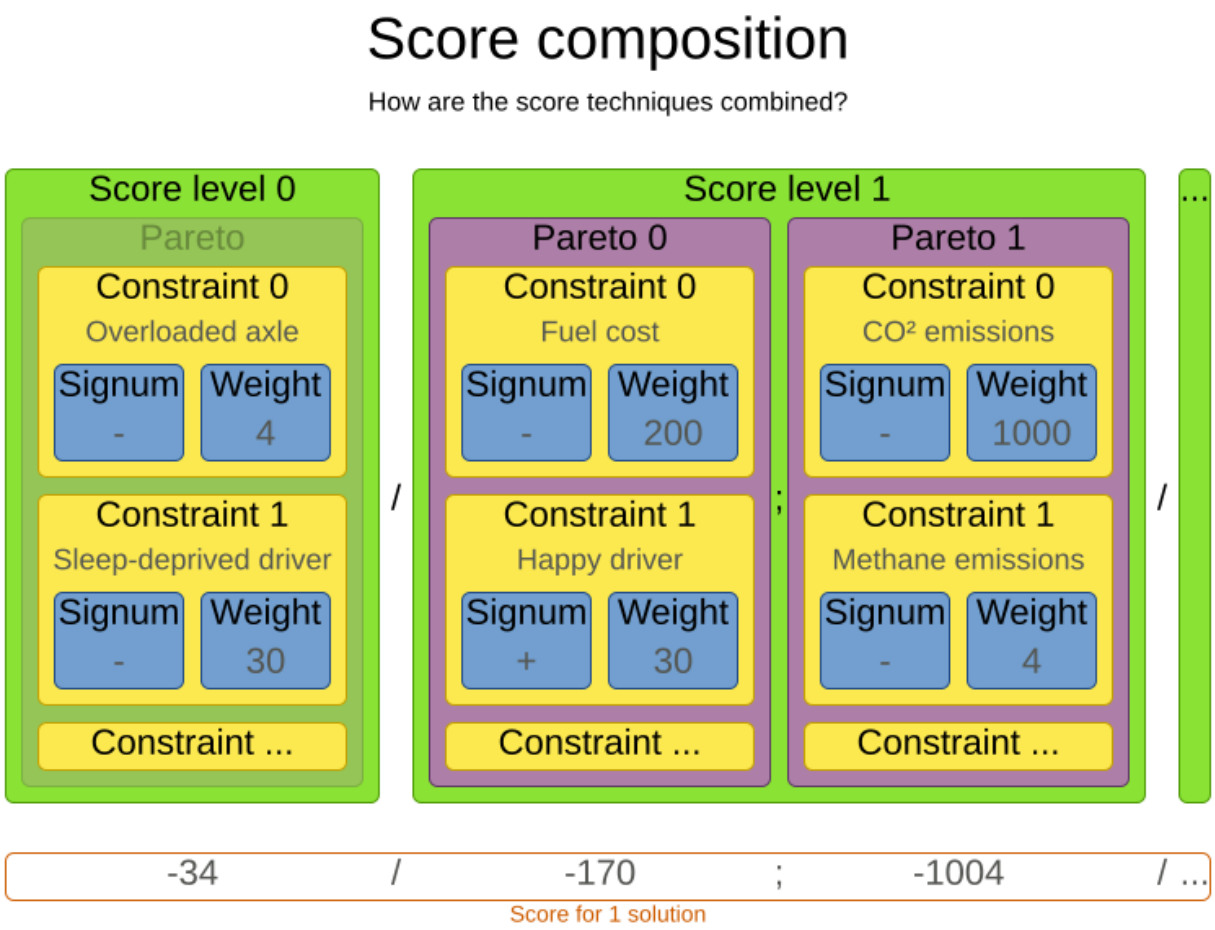
さらに、すべてのハード制約の重みが同じである場合、スコアトラップが作成される可能性があります。たとえば、クラウドバランシングの問題では、コンピューターのプロセスに搭載されている CPU が 7 つ少なすぎる場合、CPU が 1 つしか搭載されていない場合の 7 倍の重み付けが必要になります。

OptaPlanner は 3 つ以上のスコアレベルもサポートしています。たとえば、企業は、両方の制約が物理的現実の制約よりも優先されているにもかかわらず、利益が従業員の満足度よりも優先される、またはその逆であると決定する場合があります。

注記

OptaPlanner は多くのスコアレベルを処理できますが、公平性や負荷分散をモデル化するために、多くのスコアレベルを使用する必要はありません。

ほとんどのユースケースでは、**HardSoftScore** や **HardMediumSoftScore** など、2 つまたは 3 つの重みを持つスコアを使用します。これらのテクニックはすべてシームレスに組み合わせることができます。

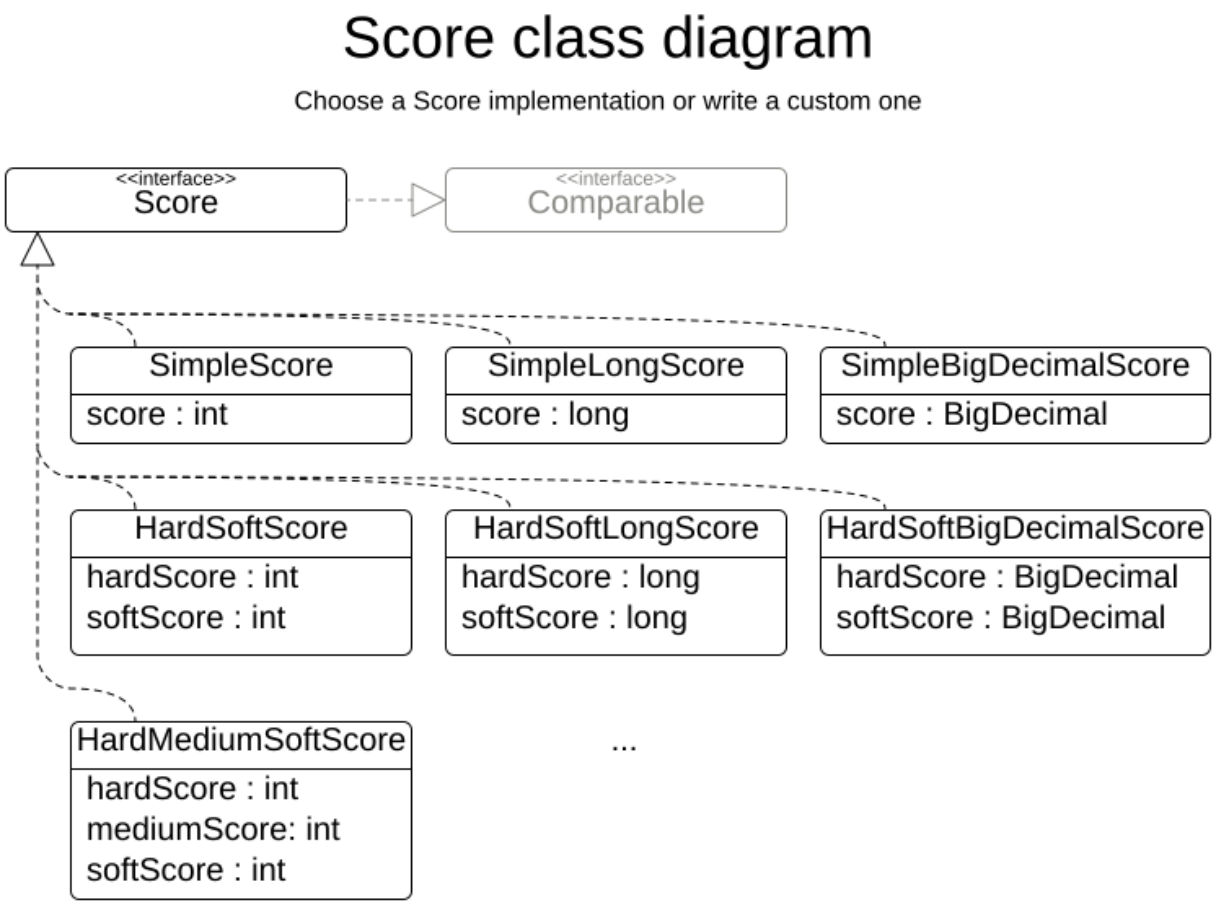

Score composition
How are the score techniques combined?
Score level 0
Pareto
Constraint 0
Overloaded axle
Signum
-
Weight
4
Constraint 1
Sleep-deprived driver
Signum
-
Weight
30
Constraint ...
/
Score level 1
Pareto 0
Constraint 0
Fuel cost
Signum
-
Weight
200
Constraint 1
Happy driver
Signum
+
Weight
30
Constraint ...
;
Pareto 1
Constraint 0
CO² emissions
Signum
-
Weight
1000
Constraint 1
Methane emissions
Signum
-
Weight
4
Constraint ...
/
...
-34
/
-170
;
-1004
/ ...
Score for 1 solution

第10章 OPTAPLANNER SCORE インターフェイス

スコアは、**Comparable** インターフェイスを拡張した **Score** インターフェイスによって表されます。

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

使用するスコア実装は、ユースケースによって異なります。スコアが単一の **long** 値に効率的に収まらない可能性があります。OptaPlanner にはいくつかの組み込みスコア実装がありますが、カスタムスコアを実装することもできます。ほとんどの使用例では、組み込みの **HardSoftScore** スコアが使用されます。



すべての Score 実装には **initScore** (**int**) もあります。これは主に OptaPlanner での内部使用を目的としており、初期化されていない計画変数の負の数です。ユーザーの観点から見ると、すべての計画変数を初期化する前に構築ヒューリスティックが終了しない限り、これは **0** です。この場合、**Score.isSolutionInitialized()** は **false** を返します。

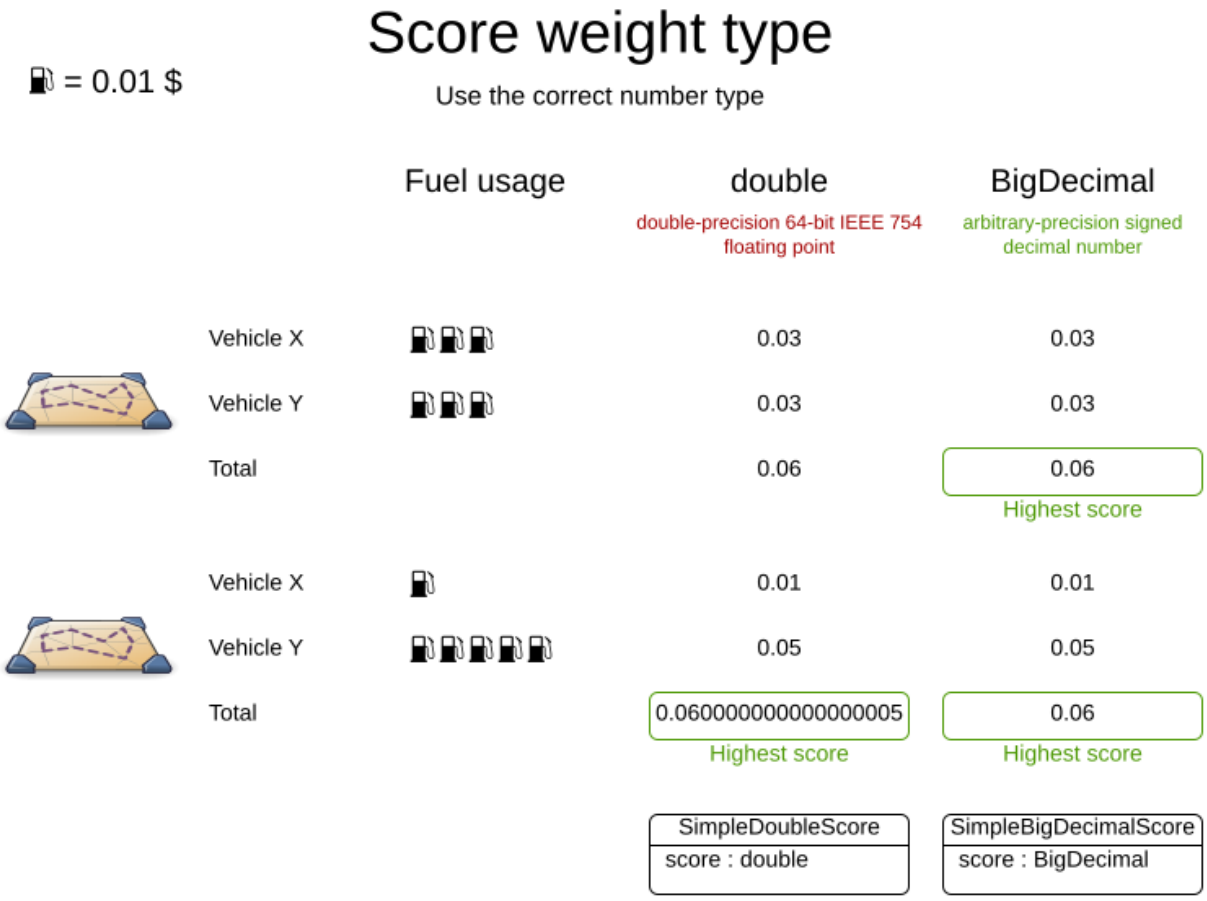
スコアの実装 (例: **HardSoftScore**) は、ソルバーランタイム全体で同じである必要があります。スコアの実装は、ソリューションドメインクラスで設定されます。

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

    @PlanningScore
```

```
    private HardSoftScore score;

}
```

10.1. スコア計算における浮動小数点数

スコア計算では浮動小数点数型 **float** または **double** を使用しないでください。代わりに **BigDecimal** またはスケーリングされた **Long** を使用してください。浮動小数点数は 10 進数を正しく表すことができません。たとえば、**double** には値 **0.05** を正しく含めることはできません。代わりに、最も近い表現可能な値が含まれます。浮動小数点数を使用する加算や減算を含む算術演算は、特に計画問題においては、次の図に示すように誤った決定につながります。



さらに、浮動小数点数の加算は結合的ではありません。

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

これは **スコアの破損** につながります。

10 進数 (**BigDecimal**) にはこれらの問題はありません。

注記

BigDecimal の算術演算は、**int**、**long**、または **double** の算術演算よりもかなり遅くなります。一部の実験では、スコアの計算に 5 倍の時間がかかります。

したがって、多くの場合、1 つのスコアの重みの **すべての** 数値を 10 の倍数で乗算し、スコアの重みがスケーリングされた **int** または **long** に収まるようにすることは価値があります。たとえば、すべての重みに **1000** を乗算すると、fuelCost **0.07** は、**fuelCostMillis 70** になり、10 進数のスコア重みは使用されなくなります。

10.2. スコア計算の種類

ソリューションのスコアを計算するには、いくつかの種類の方法があります。

- **Easy Java のスコア計算**: Java または別の JVM 言語の単一メソッドですべての制約をまとめて実装します。この方法は拡張性がありません。
- **制約ストリームのスコア計算**: 各制約を Java または別の JVM 言語で個別の制約ストリームとして実装します。この方法は高速で拡張可能です。
- **Java インクリメント演算子によるスコア計算** (非推奨): Java または別の JVM 言語で複数の低レベルメソッドを実装します。この方法は高速で拡張可能ですが、実装と保守が非常に困難です。
- **Drools スコア計算 (非推奨)**: 各制約を DRL の個別のスコアルールとして実装します。この方法は拡張可能です。

各スコア計算タイプは、**HardSoftScore** や **HardMediumSoftScore など** の任意のスコア定義で機能します。すべてのスコア計算タイプはオブジェクト指向であり、既存の Java コードを再利用できます。

重要

スコア計算は読み取り専用である必要があります。計画主体や問題の事実をいかなる形でも変更してはなりません。たとえば、スコア計算では、プランニングエンティティーのセッターメソッドを呼び出してはなりません。

OptaPlanner は、**environmentMode** アサーションが有効でない限り、ソリューションを予測できる場合には、ソリューションのスコアを再計算しません。たとえば、勝利ステップが完了した後、その動きは以前に実行され取り消されているため、スコアを計算する必要はありません。そのため、スコア計算中に適用された変更が実際に行われるという保証はありません。

計画変数が変更されたときに計画エンティティーを更新するには、代わりにシャドウ変数を使用します。

10.2.1. Easy Java のスコア計算タイプの実装

Easy Java のスコア計算タイプは、Java でスコア計算を実装する簡単な方法を提供します。Java または別の JVM 言語の単一メソッドですべての制約をまとめて実装できます。

- 利点:
  - プレーンな古い Java を使用するため、学習曲線が不要です
  - スコア計算を既存のコードベースまたはレガシーシステムに委譲する機会を提供します

- デメリット:
  - 最も遅い計算タイプ
  - インクリメント演算子によるスコア計算がないため拡張できません

手順

1. **EasyScoreCalculator** インターフェイスを実装します。

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

    Score_ calculateScore(Solution_ solution);

}
```

次の例では、N Queens 問題でこのインターフェイスを実装しています。

```
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. ソルバー設定で **EasyScoreCalculator** クラスを設定します。次の例は、N クイーン問題でこのインターフェイスを実装する方法を示しています。

```
  <scoreDirectorFactory>
```

```
<easyScoreCalculatorClass>org.optaplanner.examples.nqueens.optional.score.NQueensEasy
ScoreCalculator</easyScoreCalculatorClass>
 </scoreDirectorFactory>
```

3. **EasyScoreCalculator** メソッドの値をソルバー設定で動的に設定し、ベンチマーカーがこれらのパラメーターを調整できるようにするには、**easyScoreCalculatorCustomProperties** 要素を追加し、カスタムプロパティーを使用します。

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

10.2.2. Java インクリメント演算子によるスコア計算によるスコア計算タイプの実装

Java インクリメント演算子によるスコア計算タイプは、Java でスコア計算を増分的に実装する方法を提供します。

注記

このタイプは推奨されません。

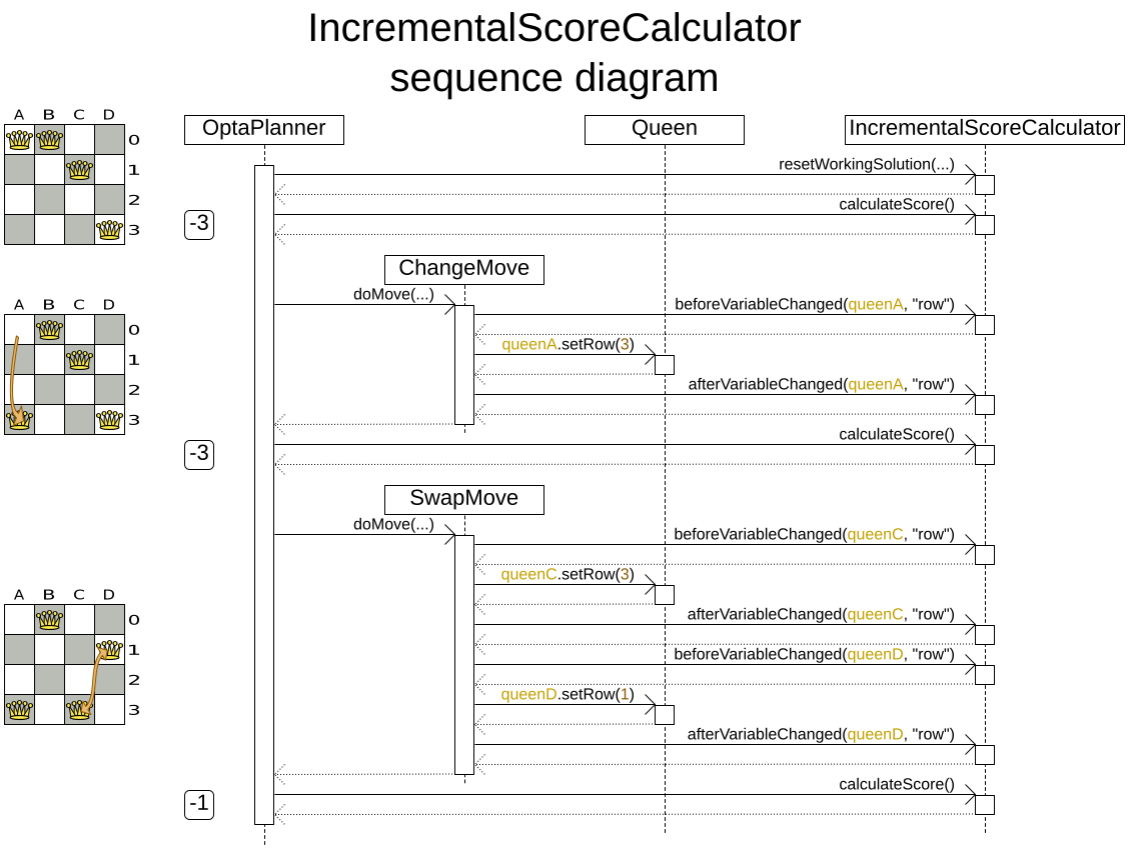
- 利点:
  - 非常に高速で拡張可能です。正しく実装されていれば、これが現時点で最も高速なタイプです。
- デメリット:
  - 記述しにくいです。
    - マップやインデックスなどを多用するスケーラブルな実装です。
    - これらのパフォーマンスの最適化をすべて自分で学習、設計、作成、改善する必要があります。
  - 読みにくいです。スコア制約を定期的に変更すると、メンテナンスコストが高くなる可能性があります。

手順

1. **IncrementalScoreCalculator** インターフェイスのすべてのメソッドを実装します。

```
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();

}
```



次の例では、N Queens 問題でこのインターフェイスを実装しています。

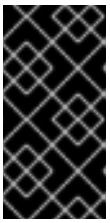
```
public class NQueensAdvancedIncrementalScoreCalculator
    implements IncrementalScoreCalculator<NQueens, SimpleScore> {

    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. ソルバー設定で、**incrementalScoreCalculatorClass** クラスを設定します。次の例は、N クイーン問題でこのインターフェイスを実装する方法を示しています。

```
  <scoreDirectorFactory>

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



重要

インクリメント演算子によるスコア計算コードの一部を作成したりレビューしたりするのは難しい場合があります。**EasyScoreCalculator** を使用して、**environmentMode** によってトリガーされたアサーションを実行することによって、その正確性をアサートします。

3. ソルバー設定で **IncrementalScoreCalculator** の値を動的に設定し、ベンチマーカーがそれらのパラメーターを調整できるようにするには、**incrementalScoreCalculatorCustomProperties** 要素を追加し、カスタムプロパティーを使用します。

```
  <scoreDirectorFactory>

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

4. オプション: **ConstraintMatchAwareIncrementalScoreCalculator** インターフェイスを実装して、次の目標を達成します。

- **ScoreExplanation.getConstraintMatchTotalMap()** を使用して、スコア制約ごとにスコア

を分割してスコアを説明します。

- **ScoreExplanation.getIndictmentMap()** を使用して、それぞれが破る制約の数によって計画エンティティーを視覚化または並べ替えます。
- **IncrementalScoreCalculator** が **FAST_ASSERT** または **FULL_ASSERT環境モード** で破損している場合は、詳細な分析を受け取ります。

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

    void resetWorkingSolution(Solution_ workingSolution, boolean
constraintMatchEnabled);

    Collection<ConstraintMatchTotal<Score_>> getConstraintMatchTotals();

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

たとえば、マシンの再割り当てでは、制約タイプごとに1つの **ConstraintMatchTotal** を作成し、制約一致ごとに **addConstraintMatch()** を呼び出します。

```
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() コードは、多くの場合、通常の **IncrementalScoreCalculator** メソッドのロジックの一部を複製します。制約ストリームと Drools スコア計算には、追加のドメイン固有のコードを必要とせずに、必要に応じて制約一致が自動的に認識されるため、この欠点はありません。

第11章 INITIALIZINGSCORETREND クラス

InitializingScoreTrend クラスを最適化アルゴリズムに追加して、追加の変数が初期化され、すでに初期化された変数が変化しないときにスコアがどのように変化するかを指定できます。構築ヒューリスティックや徹底的な検索などの一部の最適化アルゴリズムは、この情報が利用可能な場合に高速に実行されます。

スコアまたは各スコアレベルの次の傾向のいずれかを個別に指定できます。

- **ANY** (デフォルト): 追加の変数を初期化すると、スコアがプラスまたはマイナスに変化する可能性があります。この傾向ではパフォーマンスは向上しません。
- **ONLY_UP** (まれ): 追加の変数を初期化すると、スコアはプラスにのみ変更されます。**ONLY_UP** トレンドには次の条件が必要です。
  - あるのは正の制約だけです。
  - 次の変数を初期化する場合、以前に初期化された変数と一致した正の制約と一致しないことはできません。
- **ONLY_DOWN**: 追加の変数を初期化すると、スコアはマイナスにのみ変更されます。**ONLY_DOWN** には次の条件が必要です。
  - 負の制約しかありません。
  - 次の変数を初期化する場合、以前に初期化された変数によって一致した負の制約との一致を解除することはできません。

ほとんどのユースケースには負の制約のみがあります。これらの使用例の多くには、次の例に示すように、スコアを下げるだけの **InitializingScoreTrend** クラスがあります。

```
  <scoreDirectorFactory>

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

あるいは、次の例に示すように、各スコアレベルの傾向を個別に指定することもできます。

```
  <scoreDirectorFactory>

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

第12章 無効なスコアの検出

environmentMode クラスを使用し、値を **FULL_ASSERT** または **FAST_ASSERT** として指定すると、環境モードはインクリメント演算子によるスコア計算でスコアの破損を検出します。

ただし、これを行うと、スコア計算ツールがビジネスが望む方法でスコア制約を実装しているかどうかは検証されません。たとえば、1 つの制約が常に間違ったパターンに一致する可能性があります。独立した実装に対して制約を検証するには、**assertionScoreDirectorFactory** クラスを設定します。

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

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

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

この例では、**NQueensConstraintProvider** 実装が **EasyScoreCalculator** によって検証されます。

注記

この手法はスコアの破損を分離するのに効果的ですが、制約が実際のビジネスニーズを実装していることを検証するには、通常、**ConstraintVerifier** を使用した単体テストの方が優れています。

第13章 スコア計算パフォーマンスのコツ

ソルバーの実行時間のほとんどは、ソルバーの最も深いループで呼び出されるスコア計算の実行に関係します。スコア計算が高速化すると、同じアルゴリズムでより短い時間で同じソリューションが返されます。通常、これにより、同じ時間内でより良い解決策が提供されます。スコア計算のパフォーマンスを向上させるには、次のテクニックを使用します。

13.1. スコア計算速度

スコア計算を改善するときは、最高のスコアを最大化するのではなく、スコア計算速度を最大化することに重点を置きます。スコア計算が大幅に改善されても、たとえばアルゴリズムがローカルオプティマまたはグローバルオプティマに陥っている場合など、最高スコアの改善がほとんど、またはまったく起こらない場合があります。代わりに計算速度に注目すると、スコア計算の改善がより顕著に現れます。

1秒あたりのスコア計算速度は、スコア計算以外の実行時間の影響を受けますが、スコア計算パフォーマンスの信頼できる測定値となります。結果は、問題データセットの問題の規模によって異なります。通常、**EasyScoreCalculator** クラスを使用しない限り、大規模な問題であっても、1秒あたりのスコア計算速度は **1000** を超えます。

計算速度を監視することで、スコア制約を削除または追加し、最新の計算速度と元の計算速度を比較できます。

注記

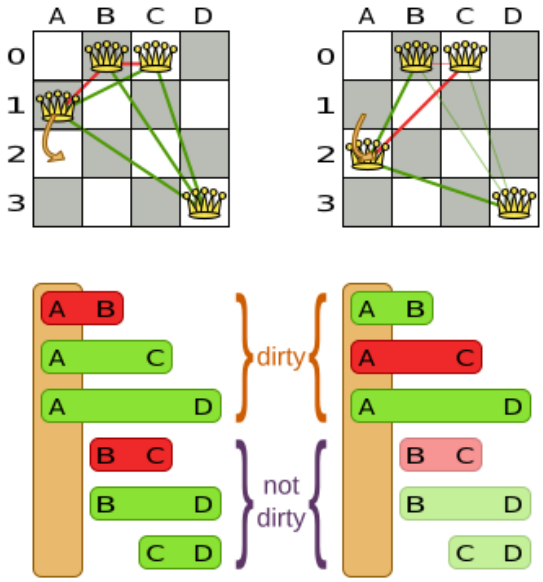
最高のスコアを元の最高のスコアと比較することは無意味です。リンゴとオレンジを比べるようなものです。

13.2. インクリメント演算子によるスコア計算

インクリメント演算子によるスコア計算は、デルタベースのスコア計算とも呼ばれます。ソリューションが変更されると、インクリメント演算子によるスコア計算では、ソリューション評価ごとにスコア全体を再計算するのではなく、現在の状態と前の状態の間の変化を評価することによって新しいスコアを見つけます。たとえば、N クイーン問題では、次の図に示すように、クイーン A が行 **1** から **2** に移動するとき、**incrementalScoreCalculation** クラスはクイーン B と C が互いに攻撃できるかどうかをチェックしません。これは、どちらも位置を変更していないためです。

Incremental score calculation

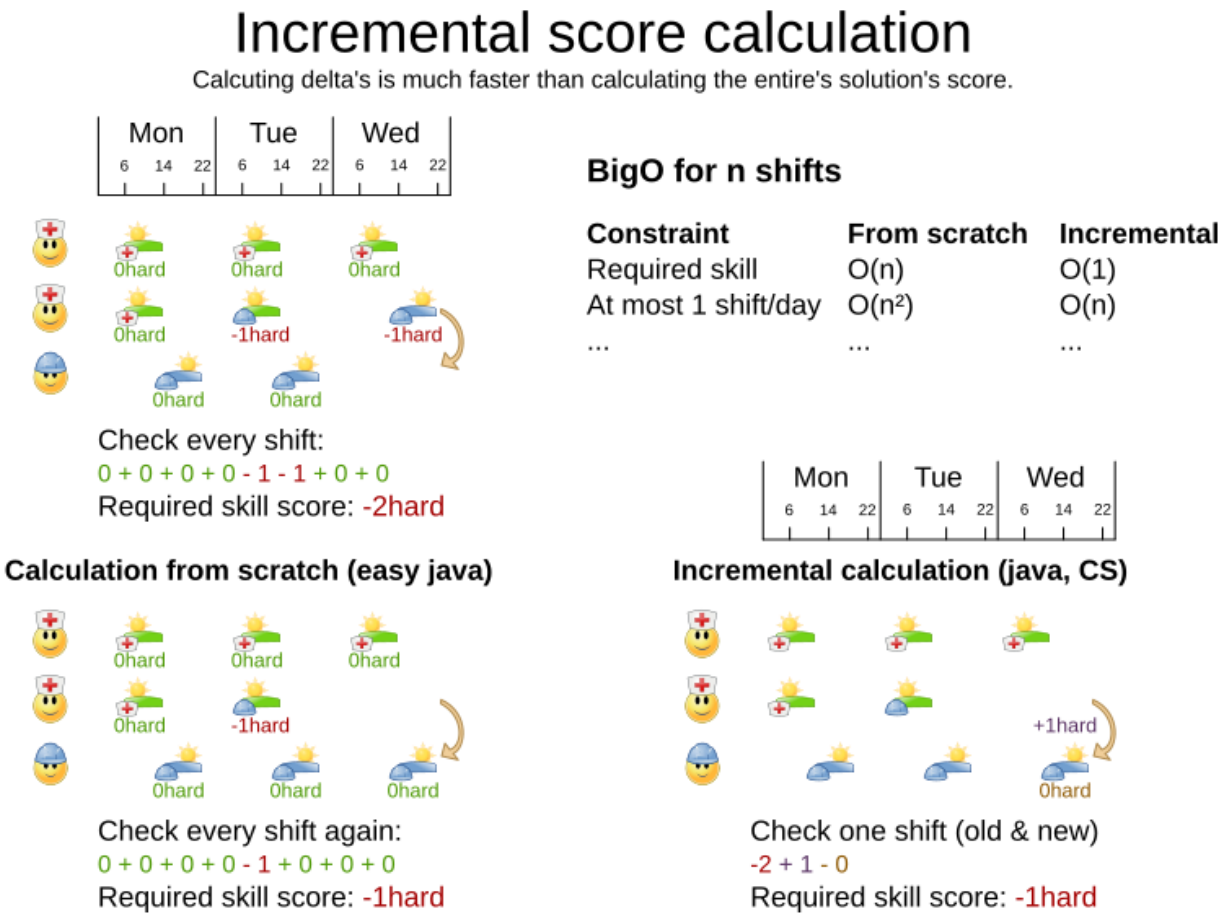
Incremental score calculation is much more scalable because only the delta is calculated.



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)*

次の例は、従業員名簿のインクリメント演算子によるスコア計算を示しています。



インクリメント演算子によるスコア計算により、パフォーマンスとスケーラビリティが大幅に向上します。制約ストリームまたは Drools スコア計算により、複雑なインクリメント演算子によるスコア計算アルゴリズムを作成することなく、このスケーラビリティが向上します。面倒な作業はルールエンジンに任せてください。

計算速度の向上は、計画上の問題のサイズ (n) に比例することに注意してください。これにより、インクリメント演算子によるスコア計算が拡張可能になります。

13.3. REMOTE SERVICES

EasyScoreCalculator クラスをレガシーシステムにブリッジする場合を除き、スコア計算でリモートサービスを呼び出さないでください。ネットワーク遅延により、スコア計算のパフォーマンスが大幅に低下します。可能であれば、それらのリモートサービスの結果をキャッシュします。

制約の一部がソルバーの開始時に一度計算され、解決中に変更されない場合は、それらをキャッシュされた問題ファクトに変換します。

13.4. 無意味な制約

特定の制約が決して破られない、または常に破られることがわかっている場合は、その制約に対してスコア制約を作成しないでください。たとえば、N クイーン問題では、クイーンの列は決して変更されず、すべての解は異なる列の各クイーンから始まるため、スコア計算では複数のクイーンが同じ列を占有するかどうかはチェックされません。

注記

このテクニックを使いすぎないでください。特定の制約を使用しないデータセットと使用するデータセットがある場合は、できるだけ早く制約から抜け出してください。データセットに基づいてスコア計算を動的に変更する必要はありません。

13.5. ビルトインのハード制約

ハード制約を実装する代わりに、ハード制約を組み込むこともできます。たとえば、学校の時間割の例では、**Lecture** A を **Room** X に割り当てるべきではないが、**Solution** で **ValueRangeProvider** クラスを使用している場合、**Solver** はそれを **Room** X に割り当てようとして、ハード制約に違反していることが判明することがよくあります。計画エンティティーまたはフィルターされた選択で **ValueRangeProvider** を使用して、講義 A に X とは異なる **Room** のみを割り当てるように定義します。

これにより、スコア計算が高速になるだけでなく、ほとんどの最適化アルゴリズムが実行不可能なソリューションの評価に費やす時間が短縮されるため、一部のユースケースではパフォーマンスが大幅に向上します。ただし、通常、これは良い考えではありません。なぜなら、短期的な利益と長期的な害を引き換えにするという現実的なリスクがあるからです。

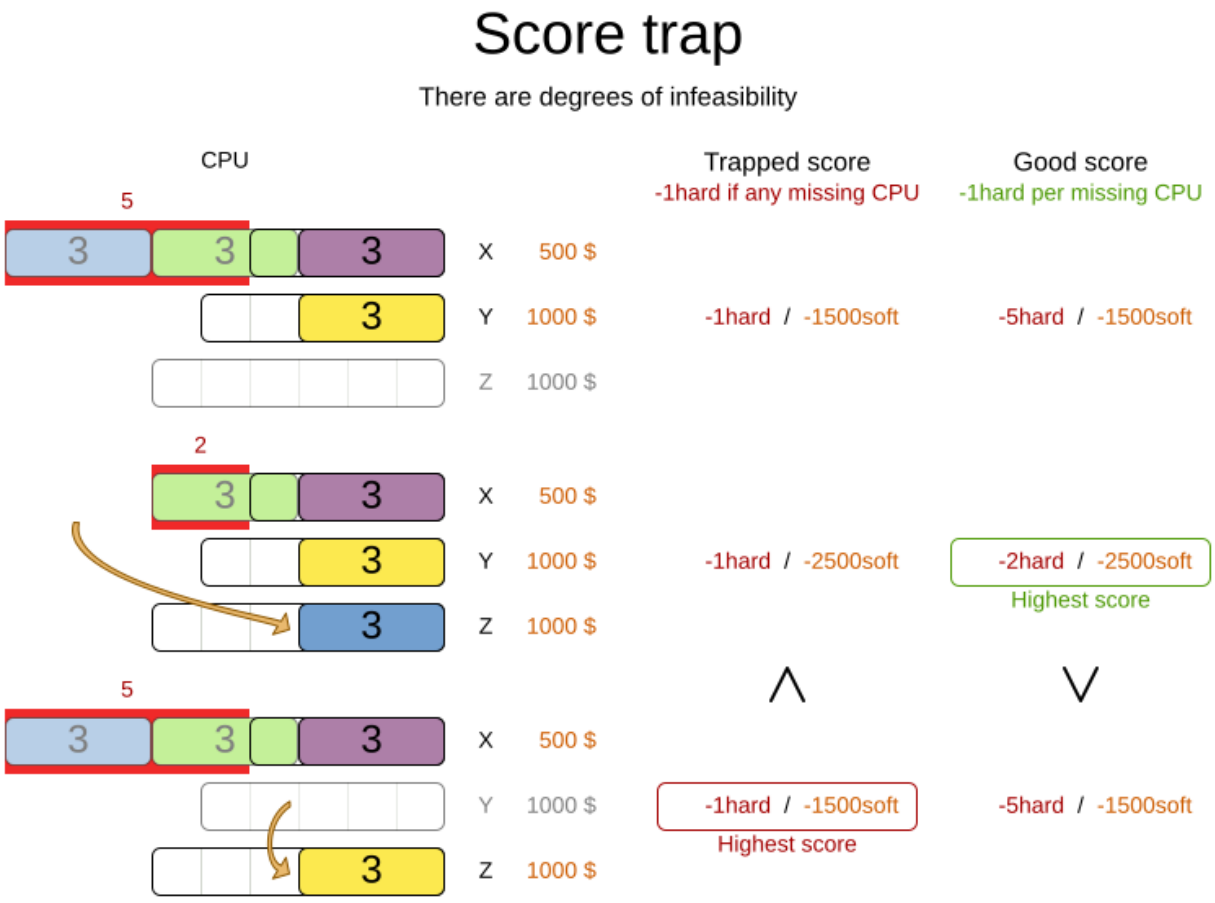
- 多くの最適化アルゴリズムは、計画エンティティーを変更するときに、ローカルオプティマから抜け出すために、厳しい制約を打ち破る自由度に依存しています。
- どちらの実装アプローチにも、機能の互換性や自動パフォーマンス最適化の無効化などの制限があります。

13.6. スコアトラップ

どのスコア制約もスコアトラップを引き起こさないようにしてください。トラップされたスコア制約は、複数の制約の一致に同じ重みを適用します。これを行うと、制約一致がグループ化され、その制約に対して平坦化されたスコア関数が作成されます。これにより、その 1 つの制約の重みを解決または下げるために複数の操作を実行する必要がある解決状態が発生する可能性があります。次の例はスコアトラップを示しています。

- 各手術台には 2 人の医師が必要ですが、一度に移動するのは 1 人の医師だけです。ソルバーには、医師がいないテーブルに医師を移動させる動機がありません。これを修正するには、スコア関数のスコア制約において、医師が 1 人もいないテーブルに対して、医師が 1 人だけいるテーブルよりも多くのペナルティを課します。
- 2 つの試験を同時に実施する必要がありますが、一度に実行できるのは 1 つの試験だけです。ソルバーは、これらの試験の 1 つを同じ移動で他の試験を移動することなく、別のタイムスロットに移動する必要があります。これを修正するには、両方の試験を同時に移動する粗粒度の移動を追加します。

次の図はスコアトラップを示しています。

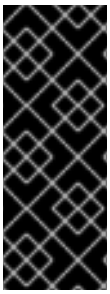


青色のアイテムが過負荷のコンピューターから空のコンピューターに移動すると、ハードスコアが向上するはずです。しかし、トラップスコアの実装ではそれができません。ソルバーは最終的にこの罠から抜け出す必要がありますが、特に過負荷になったコンピューター上にさらに多くのプロセスがある場合には、多大な労力がかかります。そうすることによるペナルティがないため、実際には、その過負荷状態のコンピューターにさらに多くのプロセスを移動し始める可能性があります。

注記

スコアトラップを回避しても、スコア関数がローカルオプティマを回避できるほど賢くなければならないという意味ではありません。ローカルオプティマの処理は最適化アルゴリズムに任せます。

スコアトラップを回避するということは、スコア制約ごとに、フラットライン化されたスコア関数を個別に回避することを意味します。



重要

実行不可能性の程度を必ず指定してください。ビジネスでは、解決策が実行不可能であれば、それがどれほど実行不可能であっても問題ではないとよく言われます。 これはビジネスには当てはまりますが、スコア計算には当てはまりません。スコア計算は、ソリューションがどれほど実行不可能であるかを知ることでメリットが得られるからです。実際には、通常、ソフト制約ではこれが自然に行われ、ハード制約でも同様に行うだけで済みます。

スコアトラップに対処する方法はいくつかあります。

- スコアの重みを区別できるようにスコア制約を改善しました。たとえば、CPU が不足している場合に **-1hard** だけのペナルティを科すのではなく、不足している CPU ごとに **-1hard** ペナルティを科します。

- ビジネスの観点からスコア制約の変更が許可されていない場合は、そのような区別を行うスコア制約を使用して、より低いスコアレベルを追加します。たとえば、CPU が不足している場合は **-1hard** に加えて、不足している CPU ごとに **-1subsoft** のペナルティを科します。ビジネスはサブソフトスコアレベルを無視します。

- 粗粒度の移動を追加し、既存の粒度の細かい移動と union-select します。粗粒度の移動は、複数の移動を効果的に実行して、1回の移動でスコアトラップから直接抜け出します。たとえば、複数のアイテムを同じコンテナーから別のコンテナーに移動します。

13.7. STEPLIMIT ベンチマーク

すべてのスコア制約のパフォーマンスコストが同じであるわけではありません。場合によっては、スコア制約が1つあると、スコア計算のパフォーマンスが完全に低下してしまうことがあります。ベンチマーカーを使用して1分間の実行を行い、1つを除くすべてのスコア制約をコメントアウトした場合にスコア計算速度に何が起こるかを確認します。

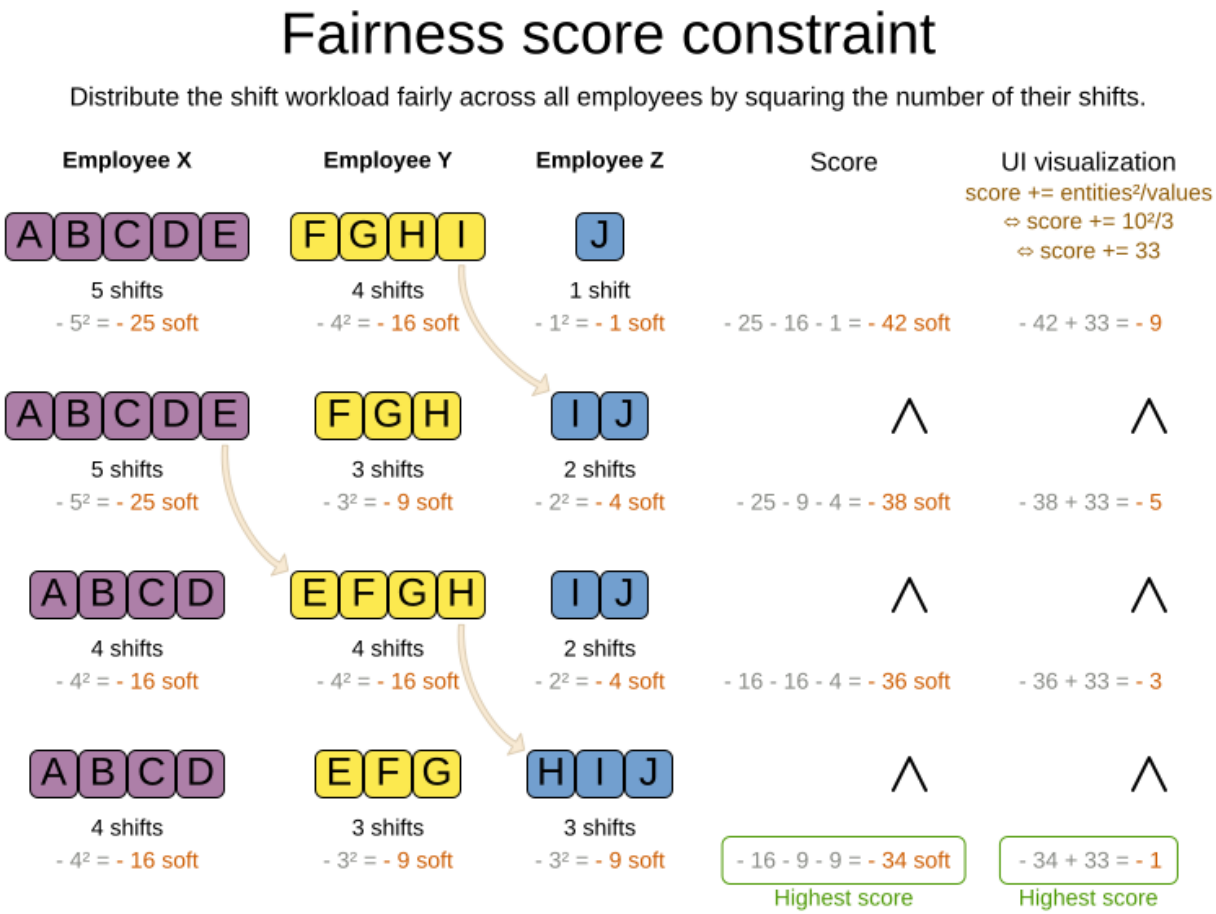
13.8. 公平性スコアの制約

一部のユースケースでは、通常はソフトスコア制約として、公平なスケジュールを提供するというビジネス要件があります。次に例を示します。

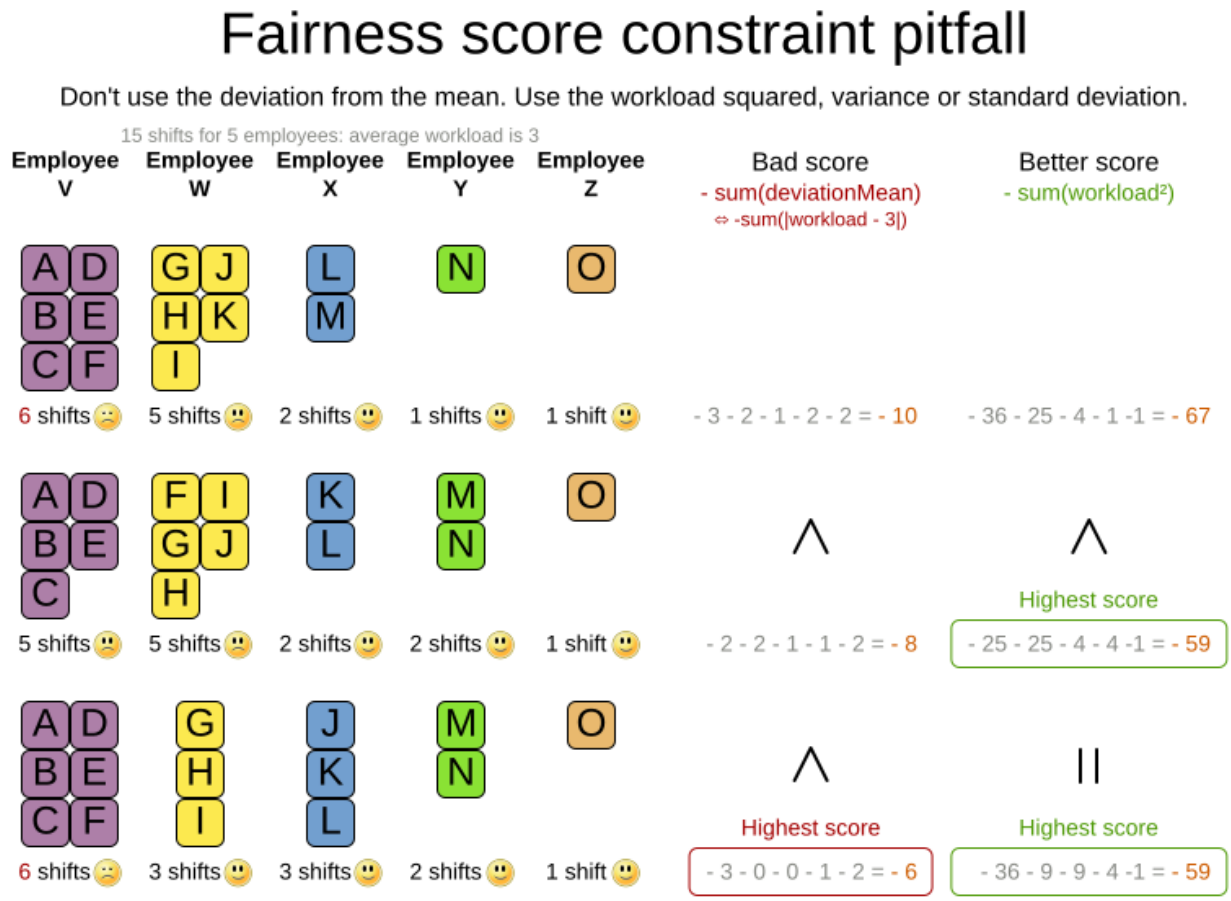
- ねたみを避けるために、従業員間でワークロードを公平に配分します。

- 信頼性を向上させるために、資産間でワークロードを均等に配分します。

このような制約の実装は、特に公平性を形式化するさまざまな方法があるため、難しいように思えるかもしれませんが、通常は **2 乗したワークロード** の実装が最も望ましい方法で動作します。各従業員または資産について、ワークロードを **w** として指定し、スコアから w^2 を減算します。



2乗したワークロードの実装では、指定されたソリューションから2人の従業員を選択し、それらの2人の従業員間の配分をより公平にすると、結果として得られる新しいソリューションの全体的なスコアが向上することが保証されます。次の図に示すように、不公平が生じる可能性があるため、平均のワークロードとの差だけを使用しないでください。



注記

2 乗したワークロード の実装の代わりに、分散 (平均との差の 2 乗) または標準偏差 (分散の平方根) を使用することもできます。平均は計画中に変更されないため、これはスコアの比較には影響しません。平均を知る必要があるため実装の作業が増えるだけでも、計算に少し時間がかかるため、明らかに遅くなります。

ワークロードが完全にバランスされている場合、ユーザーは多くの場合、注意をそらされる **-34soft** ではなく **0** スコアを参照することを好みます。これは、ほぼ完全にバランスがとれた最後のソリューションで上記の図に示されています。これを無効にするには、エンティティーの数を掛けた平均をスコアに追加するか、UI に分散偏差または標準偏差を表示します。

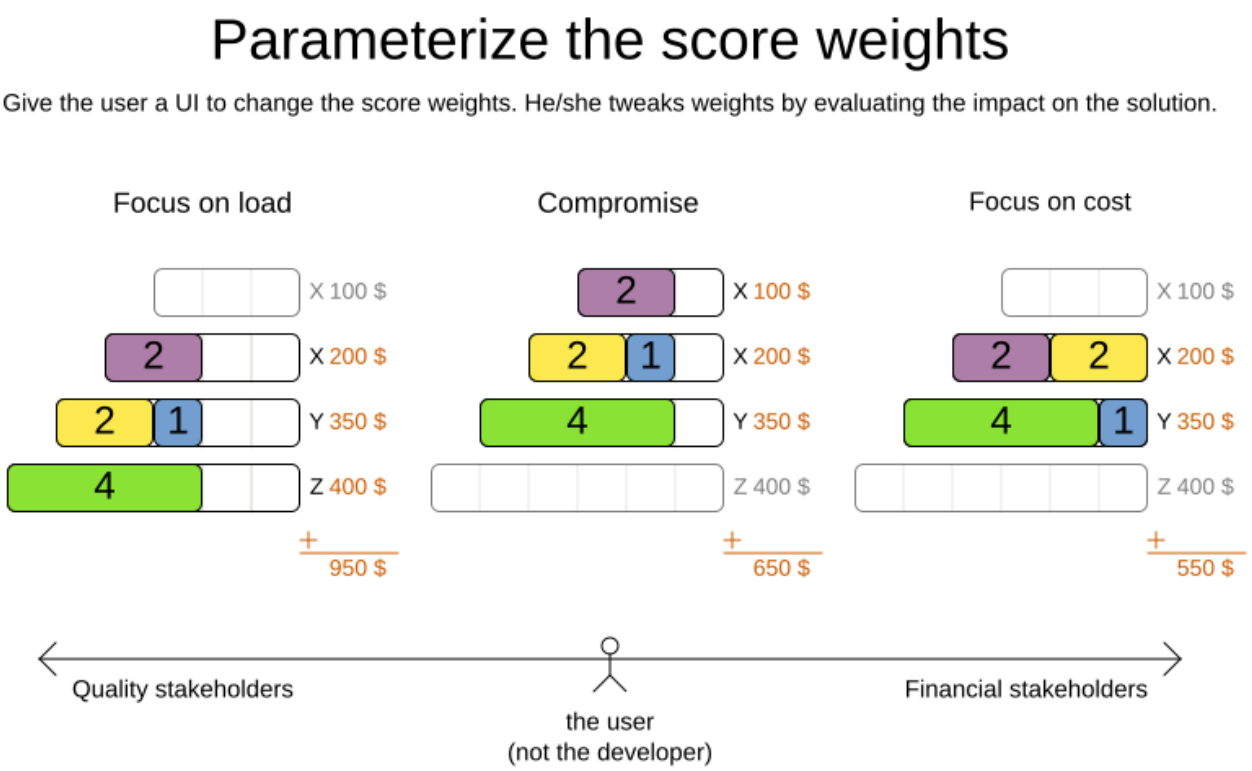
13.9. その他のスコア計算パフォーマンスのコツ

スコア計算のパフォーマンスをさらに向上させるには、次のヒントを使用してください。

- スコア計算が正しい数値タイプで行われていることを確認します。たとえば、**int** 型の値を追加する場合は、計算に時間がかかるため、結果を **double** 型として保存しないでください。
- 最適なパフォーマンスを得るには、最新の Java バージョンを使用します。たとえば、Java 11 から 17 に切り替えると、パフォーマンスが最大 10 % 向上します。
- 時期尚早な最適化は非常に望ましくないことを常に覚えて起きます。設計が設定ベースの調整を可能にするのに十分な柔軟性を持っていることを確認します。

13.10. 制約の設定

各制約の正しい重みとレベルを決定するのは簡単ではありません。多くの場合、さまざまな利害関係者やその優先事項との交渉が必要になります。さらに、ソフト制約の影響を定量化することは、多くの場合、ビジネスマネージャーにとって新しい経験であるため、正しく理解するには何度も反復する必要があります。これを簡単にするには、制約の重みとパラメーターを指定して **@ConstraintConfiguration** クラスを使用します。そして、次の図に示すように、ビジネスマネージャーが制約の重みを自分で調整し、その結果得られるソリューションを視覚化できるように UI を提供します。



たとえば、会議のスケジュール設定の問題では、最小一時停止制約には制約の重みがありますが、同じ話者による 2 つの会議時間の長さを定義する制約パラメーターもあります。一時停止の長さは会議によって異なります。一部の大規模な会議では、ある部屋から別の部屋に移動するのに 20 分では不十分ですが、小規模な会議では 10 分で十分な場合があります。一時停止の長さは、**@ConstraintWeight** アノテーションのない制約設定内のフィールドです。

各制約には制約パッケージと制約名があり、それらを合わせて制約 ID が形成されます。制約 ID は、制約の重みを制約の実装に結び付けます。**制約の重みごとに、同じパッケージと同じ名前の制約実装が必要です。**

- **@ConstraintConfiguration** アノテーションには、制約設定クラスのパッケージをデフォルトとする **constraintPackage** プロパティーがあります。制約ストリームがある場合は、通常、それを指定する必要はありません。

- **@ConstraintWeight** アノテーションには、制約名 (”Speaker conflict” など) の **value** があります。**@ConstraintConfiguration** から制約パッケージを継承しますが、たとえば **@ConstraintWeight(constraintPackage = "...region.france", ...)** をオーバーライドして、他の重みとは異なる制約パッケージを使用することができます。

したがって、すべての制約の重みは、最終的に制約パッケージと制約名になります。各制約の重みは、制約ストリームなどで制約の実装とリンクします。

```
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", ...);
    }

    ...

}
```

各制約の重みは、その制約のスコアレベルとスコアの重みを定義します。制約の実装は、**rewardConfigurable()** または **penalizeConfigurable()** を呼び出し、制約の重みが自動的に適用されます。

制約の実装が一致の重みを提供する場合、その一致の重みは制約の重みと乗算されます。たとえば、**コンテンツ競合** 制約の重みはデフォルトで **100soft** に設定されています。そして、制約の実装では、共有コンテンツタグの数と 2 つの会議の重複時間に基づいて各一致にペナルティが課されます。

```
    @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));
}
```

したがって、2 つの重複する会議が 1 つのコンテンツタグのみを共有し、60 分重複する場合、スコアは **-6000soft** の影響を受けます。ただし、2 つの重複するトークが 3 つのコンテンツタグを共有する場合、一致の重みは 180 となるため、スコアは **-18000soft** の影響を受けます。

手順

1. 制約の重みと他の制約パラメーターを保持する新しいクラス (**ConferenceConstraintConfiguration** など) を作成します。

2. このクラスに **@ConstraintConfiguration** のアノテーションを付けます。

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

3. 計画ソリューションに制約設定を追加し、そのフィールドまたはプロパティーに **@ConstraintConfigurationProvider** のアノテーションを付けます。

```
@PlanningSolution
public class ConferenceSolution {

    @ConstraintConfigurationProvider
    private ConferenceConstraintConfiguration constraintConfiguration;

    ...
}
```

4. 制約設定クラスで、各制約の **@ConstraintWeight** プロパティーを追加し、それらの重みにデフォルト値を与えます。

```
@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);

    ...
}
```

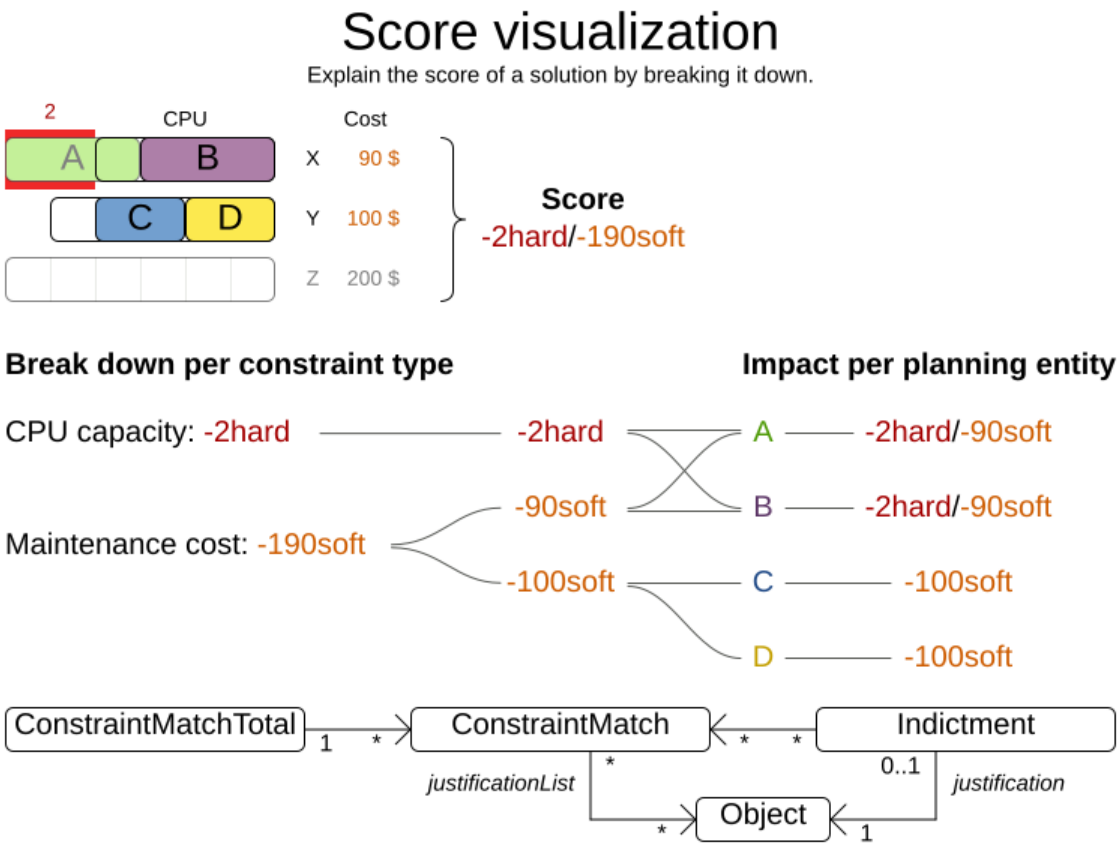
@ConstraintConfigurationProvider アノテーションは、制約設定を問題の事実として自動的に公開します。**@ProblemFactProperty** アノテーションを追加する必要はありません。制約の重みを null にすることはできません。

5. ビジネスユーザーが値を調整できるように、制約の重みを UI に公開します。前述の例では、**ofHard()**、**ofMedium()**、および **ofSoft()** メソッドを使用してこれを行います。**コンテンツの競合** 制約が **テーマトラックの競合** 制約よりも 10 倍重要であるとデフォルトで設定されていることに注目してください。通常、制約の重みでは 1 つのスコアレベルのみが使用されますが、(わずかなパフォーマンスコストで) 複数のスコアレベルを使用することも可能です。

13.11. スコアの説明

OptaPlanner スコアがどのように生成されるかを示す方法はいくつかあります。これをスコアの説明といいます。

- **getSummary()** の戻り値を出力します。これは開発中にスコアを説明する最も簡単な方法ですが、この方法は診断目的でのみ使用します。
- アプリケーションまたは Web UI で **ScoreManager** API を使用します。
- より詳細なビューを得るために、各制約のスコアを分析します。



手順

- スコアを説明するには、次のいずれかの方法を使用します。
  - **getSummary()** の戻り値を出力します。

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

次の会議スケジュールの例では、会議 **S51** が **Speaker required room tag** というハード制約を破る原因であることを出力します。

```
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

この文字列を解析したり、UI や公開サービスで使用したりしないでください。代わりに、ConstraintMatch API を使用してください。

- アプリケーションまたは Web UI で **ScoreManager** API を使用します。

  a. 次の例のようなコードを入力します。

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

  b. ソリューションのスコアを計算する必要がある場合は、このコードを使用します。

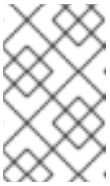
```
HardSoftScore score = scoreExplanation.getScore();
```

- スコアを制約ごとに分類します。

  a. **ScoreExplanation** から **ConstraintMatchTotal** 値を取得します。

```
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();
        ...
    }
}
```

  各 **ConstraintMatchTotal** は 1 つの制約を表し、全体のスコアの一部です。すべての **ConstraintMatchTotal.getScore()** の合計が全体のスコアと等しくなります。



注記

制約ストリームと Drools スコア計算は制約一致を自動的にサポートしますが、Java インクリメント演算子によるスコア計算には追加のインターフェイスを実装する必要があります。

13.12. ホットプランニングエンティティーの視覚化

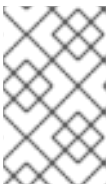
スコアに影響を与える計画エンティティーと問題の事実を強調表示するヒートマップを UI に表示します。

手順

- **ScoreExplanation** から **Indictment** マップを取得します。

```
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();
        ...
    }
}
```

各 **Indictment** は、その justification オブジェクトが関係するすべての制約の合計です。複数の **Indictment** エンティティーが同じ **ConstraintMatch** を共有できるため、すべての **Indictment.getScoreTotal()** の合計は全体のスコアとは異なります。



注記

制約ストリームと Drools スコア計算は制約一致を自動的にサポートしますが、Java インクリメント演算子によるスコア計算には追加のインターフェイスを実装する必要があります。

13.13. スコア制約のテスト

スコア計算のタイプが異なれば、テスト用のツールも異なります。各スコア制約の単体テストを個別に作成して、それが正しく動作することを確認します。

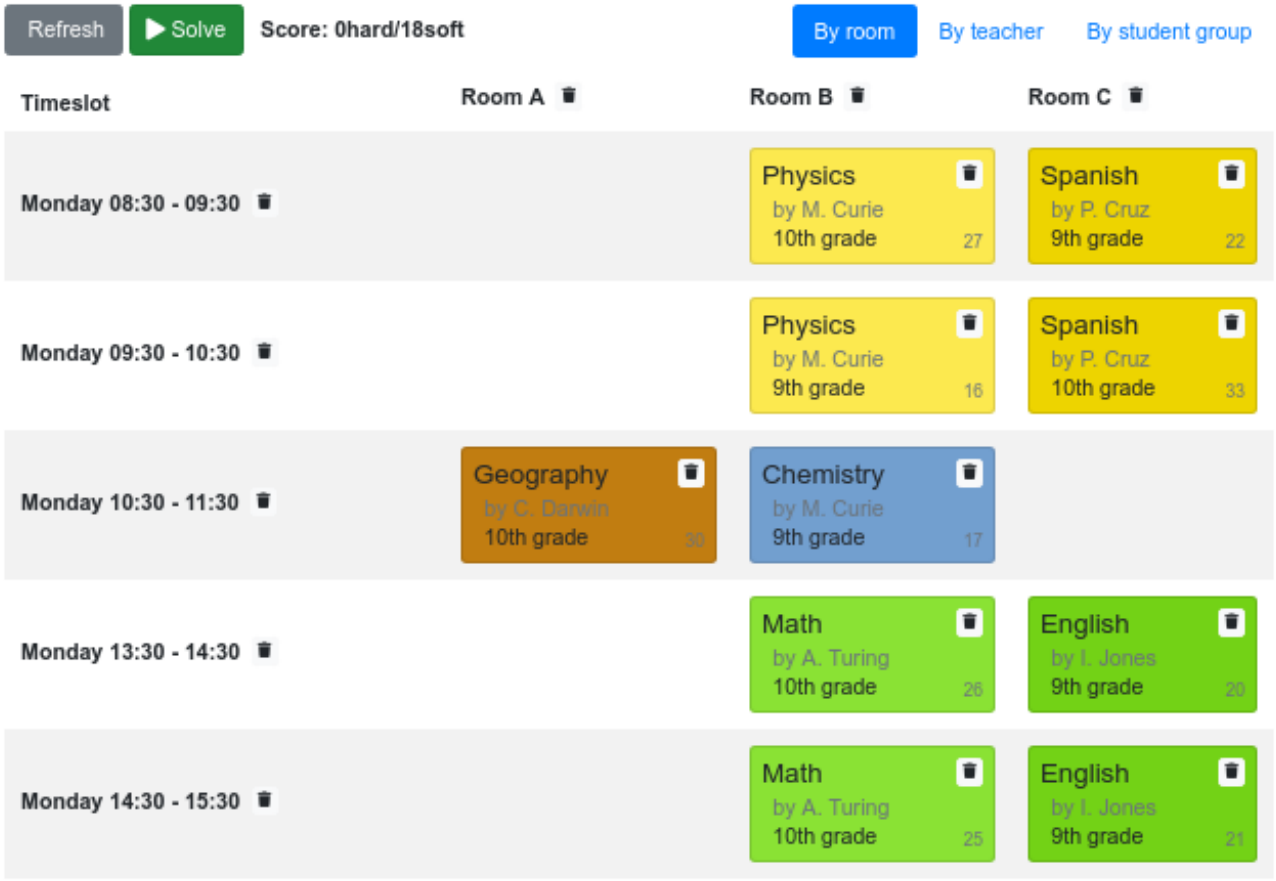
パート V. RED HAT BUILD OF OPTAPLANNER クイックスタートガイド

以下の手順に従って、従業員の勤務表サンプルに、ShiftAssignment データオブジェクトをプランニングエンティティーとして定義します。

- Red Hat build of Quarkus プラットフォーム上の Red Hat build of OptaPlanner: 時間割のクイックスタートガイド
- Red Hat build of Quarkus プラットフォーム上の Red Hat build of OptaPlanner: ワクチン接種予約スケジューラーのクイックスタートガイド
- Red Hat build of Quarkus プラットフォーム上の Red Hat build of OptaPlanner: 従業員スケジューラーのクイックスタートガイド
- Spring Boot 上の Red Hat build of OptaPlanner: 時間割のクイックスタートガイド
- Java Solver を使用した Red Hat build of OptaPlanner: 時間割のクイックスタートガイド

第14章 RED HAT BUILD OF QUARKUS プラットフォーム上の RED HAT BUILD OF OPTAPLANNER: 時間割のクイックスタートガイド

本書では、Red Hat build of OptaPlanner の制約解決人工知能 (AI) を使用して Red Hat build of Quarkus アプリケーションを作成するプロセスを説明します。学生および教師向けの時間割を最適化する REST アプリケーションを構築していきます。



サービスは、AI を使用して、以下のハードおよびソフトの **スケジュール制約** に準拠し、**Lesson** インスタンスを **Timeslot** インスタンスと **Room** インスタンスに自動的に割り当てます。

- 1 部屋に同時に割り当てることができる授業は、最大 1 コマです。
- 教師が同時に一度に行うことができる授業は最大 1 回です。
- 生徒は同時に出席できる授業は最大 1 コマです。
- 教師は、1 つの部屋での授業を希望します。
- 教師は、連続した授業を好み、授業間に時間が空くのを嫌います。

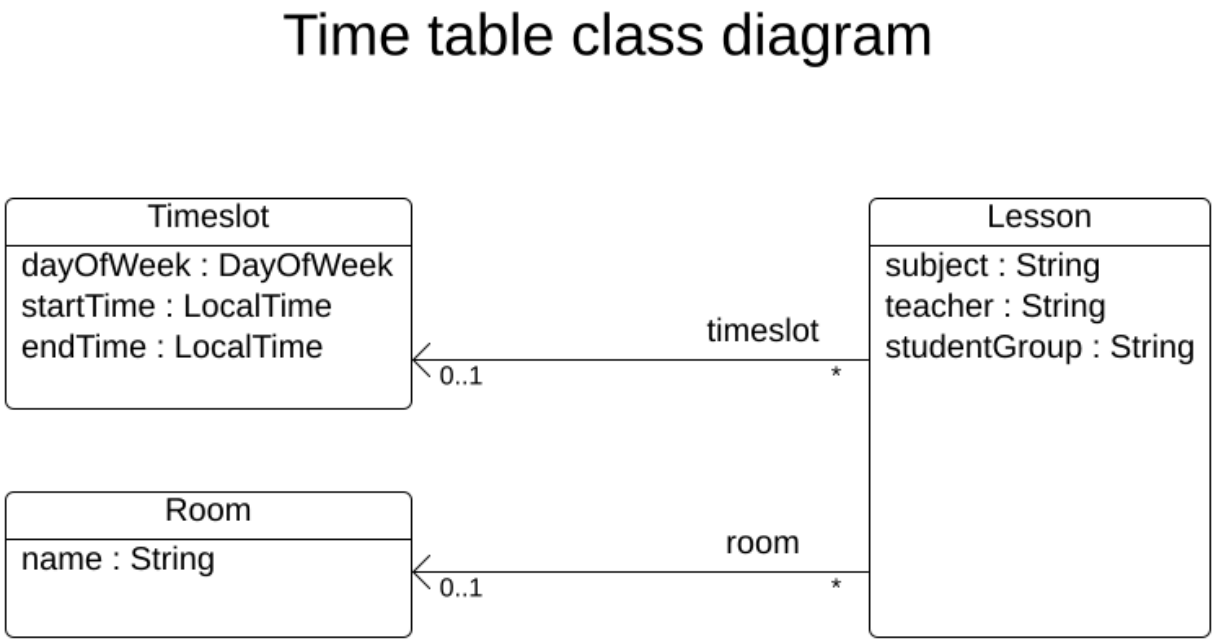
数学的に考えると、学校の時間割は **NP 困難** の問題であります。つまり、スケーリングが困難です。総当たり攻撃で考えられる組み合わせを単純にすべて反復すると、スーパーコンピューターを使用したとしても、非自明的なデータセットを取得するのに数百年かかります。幸い、Red Hat build of OptaPlanner などの AI 制約ソルバーには、妥当な時間内にほぼ最適なソリューションを提供する高度なアルゴリズムがあります。妥当な期間として考慮される内容は、問題の目的によって異なります。

前提条件

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスタマーポータル (ログインが必要) の ソフトウェアダウンロード ページから入手できます。
- Apache Maven 3.8 以降がインストールされている。Maven は Apache Maven Project の Web サイトから入手できます。
- IntelliJ IDEA、VSCode、Eclipse などの IDE が利用できる。
- Red Hat build of OptaPlanner の Red Hat build of Quarkus プロジェクトが利用できる。Red Hat build of OptaPlanner の Red Hat build of Quarkus プロジェクトの作成方法は、**Red Hat build of OptaPlanner のスタートガイド セクションの OptaPlanner および Quarkus の概要** を参照してください。

14.1. ドメインオブジェクトのモデル化

Red Hat build of OptaPlanner の時間割プロジェクトの目標は、レッスンごとに時間枠と部屋に割り当てることです。これには、次の図に示すように、**Timeslot**、**Lesson**、および **Room** の 3 つのクラスを追加します。



Timeslot

Timeslot クラスは、**Monday 10:30 - 11:30**、**Tuesday 13:30 - 14:30** など、授業の長さを表します。この例では、時間枠はすべて同じ長さ (期間) で、昼休みまたは他の休憩時間にはこのスロットはありません。

高校のスケジュールは毎週 (同じ内容が) 繰り返されるだけなので、時間枠には日付がありません。また、継続的プランニング は必要ありません。解決時に **Timeslot** インスタンスが変更しないため、Timeslot は **問題ファクト** と呼ばれます。このようなクラスには OptaPlanner 固有のアノテーションは必要ありません。

Room

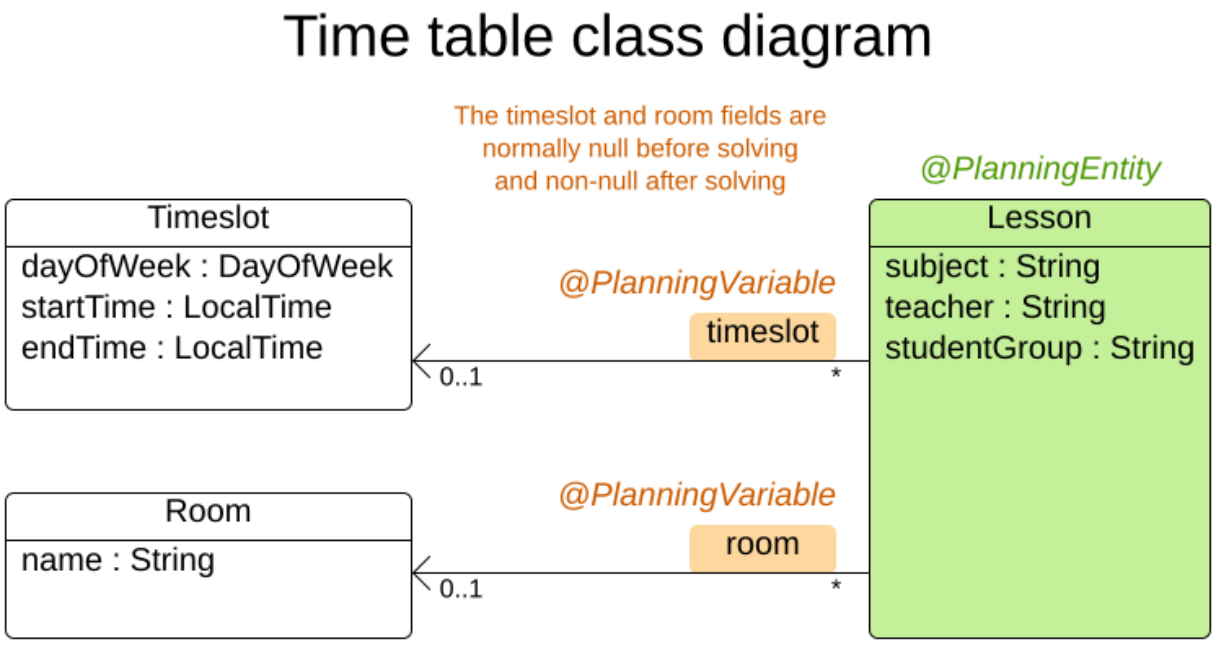
Room クラスは、**Room A**、**Room B** など、授業の場所を表します。以下の例では、どの部屋も定員制限がなく、すべての授業に対応できます。

Room インスタンスは解決時に変化しないため、**Room** は **問題ファクト** でもあります。

Lesson

授業中 (**Lesson** クラスで表現)、教師は複数の生徒に **Math by A.Turing for 9th grade**、**Chemistry by M.Curie for 10th grade** などの教科を指導します。ある教科について、毎週複数回、同じ教師が同じ生徒グループを指導する場合は、**Lesson** インスタンスが複数使用されますが、それらは **id** で識別可能です。たとえば、9 年生の場合は、1 週間に 6 回数学の授業があります。

解決中に、OptaPlanner は、**Lesson** クラスの **timeslot** フィールドと **room** フィールドを変更して、各授業を、時間枠 1 つ、部屋 1 つに割り当てます。OptaPlanner はこれらのフィールドを変更するため、**Lesson** は **プランニングエンティティー** となります。



前図では、オレンジのフィールド以外のほぼすべてのフィールドに、入力データが含まれています。授業の **timeslot** フィールドと **room** フィールドは、入力データに割り当てられておらず (**null**)、出力データに割り当てられて (**null** ではない) います。Red Hat build of OptaPlanner は、解決時にこれらのフィールドを変更します。このようなフィールドはプランニング変数と呼ばれます。このフィールドを OptaPlanner に認識させるには、**timeslot** フィールドと **room** のフィールドに **@PlanningVariable** アノテーションが必要です。このフィールドに含まれる **Lesson** クラスには、**@PlanningEntity** アノテーションが必要です。

手順

1. **src/main/java/com/example/domain/Timeslot.java** クラスを作成します。

```
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;
    }

}
```

後述しているように、**toString()** メソッドで出力を短くするため、OptaPlanner の **DEBUG** ログまたは **TRACE** ログの読み取りが簡単になっています。

2. **src/main/java/com/example/domain/Room.java** クラスを作成します。

```
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. **src/main/java/com/example/domain/Lesson.java** クラスを作成します。

```
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 クラスには **@PlanningEntity** アノテーションが含まれており、その中にプランニング変数が 1 つ以上含まれているため、OptaPlanner はこのクラスが解決時に変化することを認識します。

timeslot フィールドには **@PlanningVariable** アノテーションがあるため、OptaPlanner は、このフィールドの値が変化することを認識しています。このフィールドに割り当てることのできる **Timeslot** インスタンスを見つけ出すために、OptaPlanner は **valueRangeProviderRefs** プロパティーを使用して値の範囲プロバイダーと連携し、**List<Timeslot>** を提供して選択できるようにします。値の範囲プロバイダーに関する詳細は、「プランニングソリューションでのドメインオブジェクトの収集」 を参照してください。

room フィールドにも、同じ理由で **@PlanningVariable** アノテーションが含まれます。

14.2. 制約の定義およびスコアの計算

問題の解決時に **スコア** で導かれた解の質を表します。スコアが高いほど質が高くなります。Red Hat build of OptaPlanner は、利用可能な時間内で見つかった解の中から最高スコアのものを探し出します。これが **最適** 解である可能性があります。

時間割の例のユースケースでは、ハードとソフト制約を使用しているため、**HardSoftScore** クラスでスコアを表します。

- ハード制約は、絶対に違反しないでください。たとえば、**部屋に同時に割り当てることができる授業は、最大 1 コマです。**
- ソフト制約は、違反しないようにしてください。たとえば、**教師は、1 つの部屋での授業を希望します。**

ハード制約は、他のハード制約と比べて、重み付けを行います。ソフト制約は、他のソフト制約と比べて、重み付けを行います。ハード制約は、それぞれの重みに関係なく、常にソフト制約よりも高くなります。

EasyScoreCalculator クラスを実装して、スコアを計算できます。

```
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);
    }

}
```

残念ながら、この解は漸増的ではないので、適切にスケーリングされません。授業が別の時間枠や教室に割り当てられるたびに、全授業が再評価され、新しいスコアが計算されます。

src/main/java/com/example/solver/TimeTableConstraintProvider.java クラスを作成して、漸増的スコア計算を実行すると、解がより優れたものになります。このクラスは、Java 8 Streams と SQL を基にした OptaPlanner の ConstraintStream API を使用します。**ConstraintProvider** は、**EasyScoreCalculator** と比べ、スケーリングの規模が遥かに大きくなっています (**O** (n^2) ではなく **O** (n))。

手順

以下の **src/main/java/com/example/solver/TimeTableConstraintProvider.java** クラスを作成します。

```
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. プランニングソリューションでのドメインオブジェクトの収集

TimeTable インスタンスは、単一データセットの **Timeslot** インスタンス、**Room** インスタンス、および **Lesson** インスタンスをラップします。さらに、このインスタンスは、特定のプランニング変数の状態を持つ授業がすべて含まれているため、このインスタンスは **プランニングソリューション** となり、スコアが割り当てられます。

- 授業がまだ割り当てられていない場合は、スコアが **-4init/0hard/0soft** のソリューションなど、**初期化されていない** ソリューションとなります。
- ハード制約に違反する場合、スコアが **-2hard/-3soft** のソリューションなど、**実行不可** なソリューションとなります。
- 全ハード制約に準拠している場合は、スコアが **0hard/-7soft** など、**実行可能** なソリューションとなります。

TimeTable クラスには **@PlanningSolution** アノテーションが含まれているため、Red Hat build of OptaPlanner はこのクラスに全入出力データが含まれていることを認識します。

具体的には、このクラスは問題の入力です。

- 全時間枠が含まれる **timeslotList** フィールド
  - これは、解決時に変更されないため、問題ファクトリーストです。
- 全部屋が含まれる **roomList** フィールド
  - これは、解決時に変更されないため、問題ファクトリーストです。
- 全授業が含まれる **lessonList** フィールド
  - これは、解決時に変更されるため、プランニングエンティティーです。
  - 各 **Lesson**:
    - **timeslot** フィールドおよび **room** フィールドの値は通常、**null** で未割り当てです。これらの値は、プランニング変数です。
    - **subject**、**teacher**、**studentGroup** などの他のフィールドは入力されます。これらのフィールドは問題プロパティーです。

ただし、このクラスはソリューションの出力でもあります。

- **Lesson** インスタンスごとの **lessonList** フィールドには、解決後は null ではない **timeslot** フィールドと **room** フィールドが含まれます。
- 出力ソリューションの品質を表す **score** フィールド (例: **0hard/-5soft**)

手順

src/main/java/com/example/domain/TimeTable.java クラスを作成します。

```
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;
```

```
    }

}
```

値の範囲のプロバイダー

timeslotList フィールドは、値の範囲プロバイダーです。これは **Timeslot** インスタンスを保持し、OptaPlanner がこのインスタンスを選択して、**Lesson** インスタンスの **timeslot** フィールドに割り当てることができます。**timeslotList** フィールドには **@ValueRangeProvider** アノテーションがあり、**id** を、**Lesson** の **@PlanningVariable** の **valueRangeProviderRefs** に一致させます。

同じロジックに従い、**roomList** フィールドにも **@ValueRangeProvider** アノテーションが含まれています。

問題ファクトとプランニングエンティティーのプロパティー

さらに OptaPlanner は、変更可能な **Lesson** インスタンス、さらに **TimeTableConstraintProvider** によるスコア計算に使用する **Timeslot** インスタンスと **Room** インスタンスを取得する方法を把握しておく必要があります。

timeslotList フィールドと **roomList** フィールドには **@ProblemFactCollectionProperty** アノテーションが含まれているため、**TimeTableConstraintProvider** はこれらのインスタンスから選択できます。

lessonList には **@PlanningEntityCollectionProperty** アノテーションが含まれているため、OptaPlanner は解決時に変更でき、**TimeTableConstraintProvider** はこの中から選択できます。

14.4. SOLVER サービスの作成

REST スレッドで計画問題を解決すると、HTTP タイムアウトの問題が発生します。そのため、Quarkus スターターでは SolverManager を注入することで、個別のスレッドプールでソルバーを実行して複数のデータセットを並行して解決できます。

手順

src/main/java/org/acme/optaplanner/rest/TimeTableResource.java クラスを作成します。

```
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;
    }

}
```

この例では、初期実装はソルバーが完了するのを待つため、HTTP タイムアウトがまだ発生します。complete 実装を使用することで、より適切に HTTP タイムアウトを回避できます。

14.5. ソルバー終了時間の設定

プランニングアプリケーションに終了設定または終了イベントがない場合、理論的には永続的に実行されることになり、実際には HTTP タイムアウトエラーが発生します。これが発生しないようにするには、**optaplanner.solver.termination.spent-limit** パラメーターを使用して、アプリケーションが終了してからの時間を指定します。多くのアプリケーションでは、この時間を最低でも 5 分 (**5m**) に設定します。ただし、時間割の例では、解決時間を 5 分に制限すると、期間が十分に短いため HTTP タイムアウトを回避できます。

手順

以下の内容を含む **src/main/resources/application.properties** ファイルを作成します。

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

14.6. 時間割アプリケーションの実行

時間割プロジェクトを作成したら、開発モードで実行します。開発モードでは、アプリケーションの実行中にアプリケーションソースおよび設定を更新できます。変更が実行中のアプリケーションに反映されます。

前提条件

- 時間割プロジェクトを作成している。

手順

1. 開発モードでアプリケーションをコンパイルするには、プロジェクトディレクトリーから以下のコマンドを入力します。

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

2. REST サービスをテストします。任意の REST クライアントを使用できます。この例では Linux **curl** コマンドを使用して POST 要求を送信します。

```
$ 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"}]}'
```

application.properties ファイルで定義した **終了時間** で指定した期間後に、サービスにより、以下の例のような出力が返されます。

```
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"}
```

アプリケーションにより、4 つの授業がすべて 2 つの時間枠、そして 2 つある部屋のうちの 1 つに割り当てられている点に注目してください。また、すべてのハード制約に準拠することに注意してください。たとえば、M. Curie の 2 つの授業は異なる時間スロットにあります。

3. 解決時間の OptaPlanner の実行内容を確認するには、サーバー側で情報ログを確認します。以下は、情報ログ出力の例です。

```
... 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. アプリケーションのテスト

適切なアプリケーションにはテストが含まれます。timetable プロジェクトで制約とソルバーをテストします。

14.7.1. 学校の時間割の制約をテストする

timetable プロジェクトの各制約を個別にテストするには、単体テストで **ConstraintVerifier** を使用します。これにより、各制約のコーナーケースが他のテストから分離されてテストされるため、適切なテストカバレッジで新しい制約を追加する際のメンテナンスが軽減されます。

このテストは、制約 **TimeTableConstraintProvider::roomConflict** が、同じ部屋で 3 つのレッスンを与えられ、そのうちの 2 つのレッスンが同じタイムスロットを持つ場合、一致の重み 1 でペナルティを課すことを検証します。したがって、制約の重みが **10hard** の場合、スコアは **-10hard** 減少します。

手順

src/test/java/org/acme/optaplanner/solver/TimeTableConstraintProviderTest.java クラスを作成します。

```
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);
    }

}
```

制約の重みが **ConstraintProvider** でハードコーディングされている場合でも、**ConstraintVerifier** がテスト中に制約の重みを無視することに注意してください。これは、実稼動に入る前に制約の重みが定期的に変更されるためです。このように、制約の重みの微調整によって単体テストが中断されることはありません。

14.7.2. 学校の時間割ソルバーをテストする

以下の例では、Red Hat build of Quarkus で Red Hat build of OptaPlanner の時間割プロジェクトをテストします。このアプリケーションは、JUnit テストを使用してテストのデータセットを生成し、**TimeTableController** に送信して解決します。

手順

1. 以下の内容を含む **src/test/java/com/example/rest/TimeTableResourceTest.java** クラスを作成します。

```
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);
    }

}
```

このテストは、解決後にすべての授業がタイムスロットと部屋に割り当てられていることを確認します。また、実行可能解 (ハード制約の違反なし) も確認します。

2. テストプロパティーを **src** / **main** / **resources** /**application.properties** ファイルに追加します。

```
# 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、ソルバーは 200 ミリ秒未満で実行可能解を検索します。**application.properties** が、実行可能なソリューション (**0hard/*soft**) が見つかると同時に終了するように、テスト中のソルバーの終了を上書きします。こうすることで、ユニットテストが任意のハードウェアで実行される可能性があるため、ソルバーの時間をハードコード化するのを回避します。このアプローチを使用することで、動きが遅いシステムであっても、実行可能なソリューションを検索するのに十分な時間だけテストが実行されます。ただし、高速システムでも、厳密に必要とされる時間よりもミリ秒単位で長く実行されることはありません。

14.8. ロギング

Red Hat build of OptaPlanner の時間割プロジェクトを完了後にロギング情報を使用すると、**ConstraintProvider** で制約が微調整しやすくなります。**info** ログファイルでスコア計算の速度を確認して、制約に加えた変更の影響を評価します。デバッグモードでアプリケーションを実行して、アプリケーションが行う手順をすべて表示するか、追跡ログを使用して全手順および動きをロギングします。

手順

1. 時間割アプリケーションを一定の時間 (例: 5 分) 実行します。
2. 以下の例のように、**log** ファイルのスコア計算の速度を確認します。

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

3. 制約を変更して、同じ時間、プランニングアプリケーションを実行し、**log** ファイルに記録されているスコア計算速度を確認します。
4. アプリケーションをデバッグモードで実行して、アプリケーションの全実行ステップをログに記録します。
   - コマンドラインからデバッグモードを実行するには、**-D** システムプロパティーを使用します。
   - デバッグモードを永続的に有効にするには、以下の行を **application.properties** ファイルに追加します。

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

     以下の例では、デバッグモードでの **log** ファイルの出力を表示します。

```
... 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. **trace** ロギングを使用して、全手順、および手順ごとの全動きを表示します。

14.9. データベースを QUARKUS OPTAPLANNER 学校の時間割アプリケーションと統合する

Quarkus OptaPlanner 学校の時間割アプリケーションを作成したら、それをデータベースと統合し、ウェブベースのユーザーインターフェイスを作成して時間割を表示できます。

前提条件

- Quarkus OptaPlanner 学校の時間割アプリケーションがあります。

手順

1. Hibernate と Panache を使用して、**Timeslot**、**Room**、および **Lesson** インスタンスをデータベースに格納します。詳細については、Panache を使用した単純化された Hibernate ORM を参照してください。

2. REST を介してインスタンスを公開します。詳細については、JSON REST サービスの記述 を参照してください。

3. **TimeTableResource** クラスを更新して、単一のトランザクションで **TimeTable** インスタンスの読み取りと書き込みを行います。

```
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());
        }
    }

}
```

この例には **TimeTable** インスタンスが含まれています。ただし、マルチテナンシーを有効にして、複数の学校の **TimeTable** インスタンスを並行して処理できます。

getTimeTable() メソッドは、データベースから最新の時間割を返します。自動的に挿入される **ScoreManager** メソッドを使用して、そのタイムテーブルのスコアを計算し、UI で使用できるようにします。

solve() メソッドは、ジョブを開始して、現在の時間割を解決し、時間枠と部屋の割り当てを

データベースに保存します。このメソッドは、**SolverManager.solveAndListen()** メソッドを使用して、中間の最適解をリッスンし、それに合わせてデータベースを更新します。バックエンドがまだ解決している間、UI はこれを使用して進行状況を表示します。

4. **TimeTableResourceTest** クラスを更新して、**solve()** メソッドがすぐに戻ることを反映し、ソルバーが解決を完了するまで最新の解をポーリングするようにします。

```
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. これらの REST メソッドの上に Web UI を構築して、タイムテーブルを視覚的に表現します。

6. クイックスタートソースコード を確認します。

14.10. MICROMETER と PROMETHEUS を使用して学校の時間割を監視する OPTAPLANNER QUARKUS アプリケーション

OptaPlanner は、Java アプリケーション用のメトリック計測ライブラリーである Micrometer を介してメトリックを公開します。Prometheus で Micrometer を使用して、学校の時間割アプリケーションで OptaPlanner ソルバーを監視できます。

前提条件

- Quarkus OptaPlanner 学校の時間割アプリケーションを作成しました。
- Prometheus がインストールされている。Prometheus のインストールについては、Prometheus の Web サイトを参照してください。

手順

1. 学校の時間割 **pom.xml** ファイルに Micrometer Prometheus 依存関係を追加します。

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

2. 学校の時間割アプリケーションを開始します。

```
mvn compile quarkus:dev
```

3. Web ブラウザーで **http://localhost:8080/q/metric** を開きます。

第15章 RED HAT ビルドの QUARKUS での RED HAT ビルドの OPTAPLANNER: ワクチン接種予約スケジューラーのクイックスタートガイド

OptaPlanner ワクチン接種予約スケジューラーのクイックスタートを使用して、効率性および公平性の高いワクチン接種スケジュールを開発できます。ワクチン接種予約スケジューラーは、人工知能 (AI) を使用して摂取者の優先順位を決定し、複数の制約や優先順位に基づいて時間枠を割り当てます。

前提条件

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスマーポータル (ログインが必要) の ソフトウェアダウンロード ページから入手できます。
- Apache Maven 3.8 以降がインストールされている。Maven は Apache Maven Project の Web サイトから入手できます。
- IntelliJ IDEA、VSCode、Eclipse などの IDE が利用できる。
- 5章*Red Hat build of Quarkus プラットフォームでの Red Hat build of OptaPlanner の使用* の説明に従って、Red Hat build of Quarkus プラットフォームプロジェクトに OptaPlanner プロジェクトを作成している。

15.1. OPTAPLANNER ワクチン接種予約のスケジューラーの仕組み

スケジュール予約には、主に 2 つの方法があります。システムは、ユーザーが予約枠 (ユーザーの選択) を選択できるようにするか、システムが予約枠を割り当ててそのユーザーにワクチン予約の日付と場所 (システムの自動割当) を通知することができます。OptaPlanner のワクチン接種予約スケジューラーは、システム自動割り当てのアプローチを使用します。OptaPlanner ワクチン接種予約スケジューラーでは、システムに情報を提供するアプリケーションを作成して、システムが予約を割り当てます。

このアプローチの特徴は次のとおりです。

- 予約枠は優先順位に基づいて割り当てられます。
- システムは、事前設定したプランニング制約に基づいて、最適な時間と場所を割り当てます。
- システムは、多数のユーザーが少ない予約数に殺到しても、圧迫されることはありません。

このアプローチにより、プランニング制約を使用して各ユーザーのスコアを作成し、できるだけ多くの人がワクチン接種を受けられるようにする問題を解決します。スコアで、予約を取得するタイミングが決まります。スコアが高いほど、早い段階で受信できる可能性が高くなります。

15.1.1. Red Hat build of OptaPlanner ワクチン接種予約のスケジューラーの制約

Red Hat build of OptaPlanner ワクチン接種予約のスケジューラー制約は、ハード、中、またはソフトのいずれかです。

- ハード制約は、絶対に違反できません。ハード制約に違反している場合には、プランは実行不可なので実行できません。
  - 容量: 場所、時間に関わらず、ワクチン量よりも多く予約を取らない。
  - ワクチン接種年齢上限: ワクチンに年齢の上限がある場合には、1 回目のワクチン接種時にワクチンの年齢の上限以上の方には投与しないようにする。年齢にあったワクチンタイプ

を投与するようにします。たとえば、75 歳の方には、年齢の上限が 65 歳のワクチンの予約を割り当てないようにします。

- 必要なワクチンタイプ: 必要なワクチンタイプを使用する。たとえば、2 回目のワクチンは、1 回目と同じタイプを使用する必要があります。
- 準備が整う日: 指定の日付以降にワクチンを投与する。たとえば、2 回目の投与を受ける場合に、特定のワクチンタイプを最短で接種が可能な推奨日時よりも前に投与しないでください (例: 1 回目の接種から 26 日後など)。
- 期日: 指定された日付以前にワクチンを接種する。たとえば、2 回目の投与を受ける場合に、特定のワクチンタイプを接種可能な推奨最終日よりも前に投与してください (例: 1 回目の接種から 3 ヶ月後など)。
- 移動の最大距離の制限: 最寄りのワクチンセンターグループに、各接種者を割り当てる。通常、これは 3 つのセンターのいずれかになります。この制限は、距離ではなく移動時間によって計算されるので、郊外に居住の方に比べ、都市部に居住する方は最大距離が短くなります。

- 中程度の制約は、全員に予約を割り当てる空きがない場合に、どの接種希望者が予約できないかを決定します。これは、過剰制約プランニングと呼ばれます。
  - 2 回目のワクチン接種予約: 理想の日付が計画枠外になってしまう場合を除き、2 回目のワクチン接種予約を割り当てずに放置しないようにする。
  - 優先度評価に基づくスケジュール設定: 各接種者に優先度評価がある。これは、通常年齢ですが、医療関係者などの場合には優先度が高くなります。優先度が最も低い場合にのみ、予約を割り当てずに放置できます。次のワクチン接種のタイミングで考慮されます。2 回目の投与は優先度評価よりも優先されるので、この制約は以前の制約よりもソフトです。

- ソフト制約は、違反しないようにしてください。
  - 希望のワクチンセンター: 接種者が希望するワクチンセンターがある場合は、そのセンターで予約を入れる。
  - 距離: 接種者が割り当てられたワクチンセンターまで移動する距離を最小限に抑える。
  - 理想の日付: できるだけ指定の日付に近い日にワクチンを投与する。たとえば、2 回目の投与を受ける場合に、特定のワクチンで理想の日付に投与します (例: 1 回目の接種から 28 日後など)。この制約は、距離制約よりもソフトで、理想の日付に近づけるために、遠方まで出向かせる必要がないようにします。
  - 優先度評価: 優先度評価の高い方から、予約枠で日付が近い順に割り当てる。この制約は、距離制約よりもソフトで、遠方まで出向かせる必要がないようにします。2 回目の投与は優先度評価よりも優先されるので、この制約も以前の制約よりもソフトです。

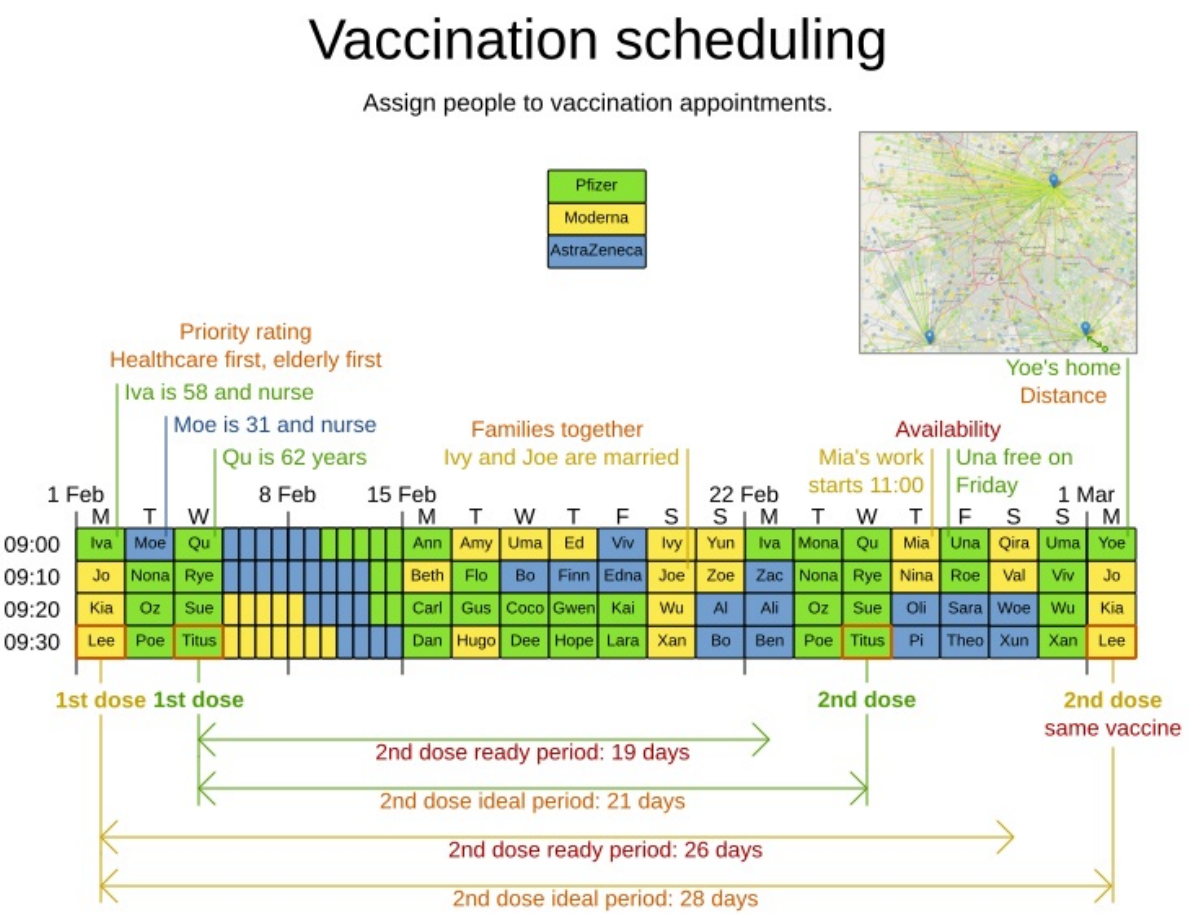
ハード制約は、他のハード制約と比べて、重み付けを行います。ソフト制約は、他のソフト制約と比べて、重み付けを行います。ただし、ハード制約は、常に中程度およびソフト制約よりも優先されます。ハード制約に違反している場合には、プランは実行できません。ただし、ハード制約に違反していない場合には、優先度を判定するためにソフト制約および中程度の制約が考慮されます。空きがある予約枠よりも希望者が多いので、優先付けが必要です。2 回目の投与予約は必ず先に割り当て、後ほどシステムに負荷がかからないようにバックログの作成は回避します。その後、優先度評価をもとに割り当てていきます。優先度評価年齢から評価を開始します。この評価では、若い人よりお年寄りが優先されます。その後、特定の優先度のグループには、200-300 ポイントなど、余分にポイントが追加されます。これはグループの優先順位によって異なります。たとえば、看護師は追加で 1000 ポイントを受け取る可能性があります。こうすることで、年齢の高い看護師は、若い看護師よりも優先され、若い看護師は看護師ではない人よりも優先されます。以下の表は、この概念を示しています。

表15.1 優先順位評価の表

年齢	職業	優先順位の評価
60	看護師	1060
33	看護師	1033
71	定年退職	71
52	オフィスワーカー	52

15.1.2. Red Hat build of OptaPlanner のソルバー

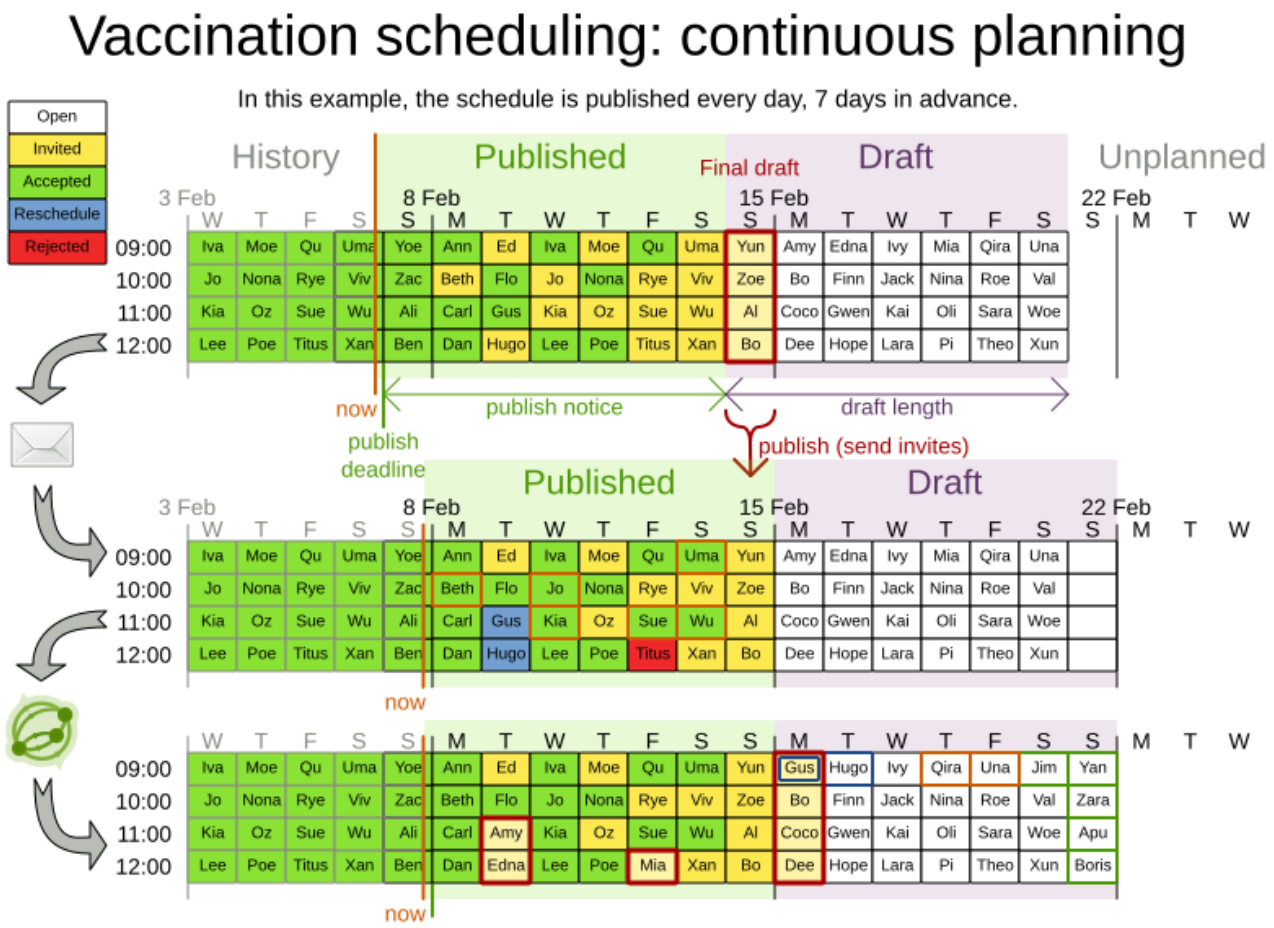
OptaPlanner のコアには Solver があり、エンジンは、問題データセットを取り、プランニングの制約および設定をオーバーレイします。問題データセットには、人、ワクチン、ワクチンセンターなどの情報すべてが含まれます。ソルバーは、さまざまなデータを組み合わせて機能し、最終的に特定のセンターでワクチン接種予約を割り当てる時に、最適な予約スケジュールを決定します。以下は、ソルバーで作成されたスケジュールを示しています。



15.1.3. 継続プランニング

継続プランニングは、1 つ以上の今後のプランニング期間をまとめて管理して、そのプロセスを月単位、週単位、日単位、1 時間単位、またはそれよりも短い単位で繰り返す手法です。プランニング枠は、指定した間隔で増分して進められます。以下は、毎日更新される 2 週間のプランニング枠を示して

います。



2 週間計画枠を半分に分割します。1 週間目は公開状態で、2 週間目がドラフト状態にあります。プランニング枠の公開部分とドラフト部分の両方で、予約を割り当てます。ただし、プランニング枠の公開部分の方だけに、予約の通知が行きます。他の予約は、次の実行でまだ簡単に変更できます。こうすることで、ソルバーを次回実行すると、必要に応じて OptaPlanner が柔軟にドラフトの部分の予約を変更できます。たとえば、2 回目の投与の準備が月曜にできており、理想の日付が水曜の場合には、同じ週のドラフト予約を指定できることが証明できるのであれば、OptaPlanner は月曜に予約を割り当てる必要はありません。

プランニング枠のサイズは指定できますが、問題領域のサイズを認識しておいてください。問題領域は、スケジュールの作成に使用されるさまざまな要素すべてです。計画期間が長いと、問題領域が大きくなります。

15.1.4. 固定されたプランニングエンティティー

1 日単位で継続的にプランニングしている場合には、 2 週間の中ですでに割り当てられている予約枠があります。ダブルブッキングされないように、OptaPlanner は予約枠が割り当て済みと固定することができます。1 つ以上の特有の割当のアンカリングや、OptaPlanner が強制的に固定の割当を除外してスケジュールするのに、固定機能を使用します。予約など、固定されたプランニングエンティティーは、解決時には変更されません。

エンティティーが固定されているかどうかは、予約の状態により分かります。予約の状態には **Open**、**Invited**、**Accepted**、**Rejected** または **Rescheduled** があります。

注記

OptaPlanner エンジンは、予約が固定されているかどうかのみに着目するので、クイックスタートのデモコードに予約の状態は直接表示されません。

スケジュール済みの予約に近い日付で、プランニングできるようにする必要があります。状態が **Invited** または **Accepted** の予約が固定されます。状態が **Open**、**Reschedule** および **Rejected** の予約は固定さず、スケジューリングに利用できます。

この例では、ソルバーが実行されると、公開範囲とドラフト範囲の両方を含めた 2 週間の計画枠全体を検索します。ソルバーは、スケジューリング前の入力データに加え、固定されておらず、状態が **Open**、**Reschedule** または **Rejected** の予約、エンティティーを検討して、最適解を見つけ出します。ソルバーを日次で実行する場合には、ソルバーを実行する前に新しい日付が追加されることが確認できます。

新しい日付に予約が割り当てられ、固定ウィンドウのドラフト部分で、これまでにスケジュールされていた Amy と Edna が公開部分の枠で予約が取れていることが分かります。これは、Gus および Hugo が再スケジュールを依頼したので可能となりました。Amy と Edna にはドラフトの日付が通知されていないので、混乱を招くことはありません。これで、計画枠の公開部分で予約が取れたので、Amy と Edna には通知が送信され、その予約を承諾するか拒否するかが確認され、この 2 人の予約が固定されます。

15.2. OPTAPLANNER ワクチン接種予約スケジューラーのダウンロードおよび実行

OptaPlanner ワクチン接種予約スケジューラークイックスタートをダウンロードし、Quarkus の開発モードで起動して、ブラウザーでアプリケーションを表示します。Quarkus 開発モードを使用すると、実行中にアプリケーションを変更し、更新できます。

手順

1. Red Hat カスタマーポータルの Software Downloads ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
   - **製品:** Red Hat build of OptaPlanner
   - **バージョン:** 8.38
2. **Red Hat build of OptaPlanner 8.38 クイックスタート**をダウンロードします。
3. **rhbop-8.38.0-optaplanner-quickstarts-sources.zip** ファイルを展開します。
   展開先の **org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/use-cases/vaccination-scheduling** ディレクトリーには、サンプルソースコードが含まれています。
4. **org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/use-cases/vaccination-scheduling** ディレクトリーに移動します。
5. 以下のコマンドを入力して、開発モードで OptaPlanner 解析スケジューラーを起動します。

   ```
   $ mvn quarkus:dev
   ```

6. OptaPlanner ワクチン接種予約スケジューラーを表示するには、Web ブラウザーに以下の URL を入力します。

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

7. OptaPlanner ワクチン接種予約スケジューラーを実行するには、**Solve** をクリックします。
8. ソースコードに変更を加えてから、F5 キーを押してブラウザーを更新します。加えた変更が有効になったことを確認してください。

15.3. OPTAPLANNER ワクチン接種予約スケジューラーのパッケージ化および実行

quarkus:dev モードの OptaPlanner ワクチン接種予約スケジューラーで開発作業が完了したら、アプリケーションを従来の jar ファイルとして実行します。

前提条件

- OptaPlanner ワクチン接種予約スケジューラーのクイックスタートをダウンロードしている。

手順

1. **/use-cases/vaccination-scheduling** ディレクトリーに移動します。
2. OptaPlanner 解析スケジューラーをコンパイルするには、以下のコマンドを入力します。

```
$ mvn package
```

3. コンパイルした OptaPlanner ワクチン接種予約スケジューラーを実行するには、以下のコマンドを入力します。

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

注記

ポート 8081 でアプリケーションを実行するには、**-Dquarkus.http.port=8081** を前述のコマンドに追加します。

4. OptaPlanner ワクチン接種予約スケジューラーを起動するには、Web ブラウザーに以下の URL を入力します。

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

15.4. 関連情報

- スケジュールスケジュール動画 (英語版)

第16章 RED HAT BUILD OF QUARKUS 上の RED HAT BUILD OF OPTAPLANNER: 従業員スケジューラーのクイックスタートガイド

従業員スケジューラークイックスタートアプリケーションは、従業員を組織内のさまざまな役職のシフトに割り当てます。たとえば、アプリケーションを使用して、病院での看護師のシフト、さまざまな場所での警備勤務シフト、作業者の組み立てラインのシフトを割り当てます。

最適な従業員のスケジューリングでは、多くの変数を考慮に入れる必要があります。たとえば、業務が異なれば、求められるスキルが異なります。また、従業員の中には、特定の時間帯に勤務できない場合や、特定の時間帯での勤務を希望する場合があります。さらに、従業員によっては、1 回に就業できる時間に制限がある契約を交わしている可能性があります。

このスターターアプリケーションの Red Hat build of OptaPlanner ルールは、ハード制約およびソフト制約を使用します。最適化時に、従業員が勤務できない (または病欠の) 場合や、ある 1 つのシフト内の 2 つのスポットで働くことができない場合など、Planner エンジンはハード制約に違反することができません。Planner エンジンは、ソフト制約 (特定のシフトで勤務しないという従業員の希望など) に順守しようとしますが、最適なソリューションには違反が必要だと判断した場合は、違反することができます。

前提条件

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスタマーポータル (ログインが必要) の ソフトウェアダウンロード ページから入手できます。
- Apache Maven 3.8 以降がインストールされている。Maven は Apache Maven Project の Web サイトから入手できます。
- IntelliJ IDEA、VSCode、Eclipse などの IDE が利用できる。

16.1. OPTAPLANNER 従業員スケジューラーのダウンロードと実行

OptaPlanner 従業員スケジューラークイックスタートアーカイブをダウンロードし、Quarkus 開発モードで起動して、ブラウザーでアプリケーションを表示します。Quarkus 開発モードを使用すると、実行中にアプリケーションを変更し、更新できます。

手順

1. Red Hat カスタマーポータルの Software Downloads ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
   - **製品:** Red Hat build of OptaPlanner
   - **バージョン:** 8.38
2. **Red Hat build of OptaPlanner 8.38 クイックスタート**をダウンロードします。
3. **rhbop-8.38.0-optaplanner-quickstarts-sources.zip** ファイルを展開します。
4. **org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/use-cases/employee-scheduling** ディレクトリーに移動します。
5. 次のコマンドを入力して、OptaPlanner 従業員スケジューラーを開発モードで開始します。

```
$ mvn quarkus:dev
```

6. OptaPlanner 従業員スケジューラーを表示するには、Web ブラウザーに次の URL を入力します。

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

7. OptaPlanner 従業員スケジューラーを実行するには、**Solve** をクリックします。

8. ソースコードに変更を加えてから、F5 キーを押してブラウザーを更新します。加えた変更が有効になったことを確認してください。

16.2. OPTAPLANNER 従業員スケジューラーのパッケージ化および実行

quarkus:dev モードで OptaPlanner 従業員スケジューラーの開発作業が完了したら、アプリケーションを従来の jar ファイルとして実行します。

前提条件

- OptaPlanner 従業員スケジューリングクイックスタートをダウンロードしている。

手順

1. **/use-cases/vaccination-scheduling** ディレクトリーに移動します。

2. OptaPlanner 従業員スケジューラーをコンパイルするには、次のコマンドを入力します。

```
$ mvn package
```

3. コンパイルされた OptaPlanner 従業員スケジューラーを実行するには、次のコマンドを入力します。

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

注記

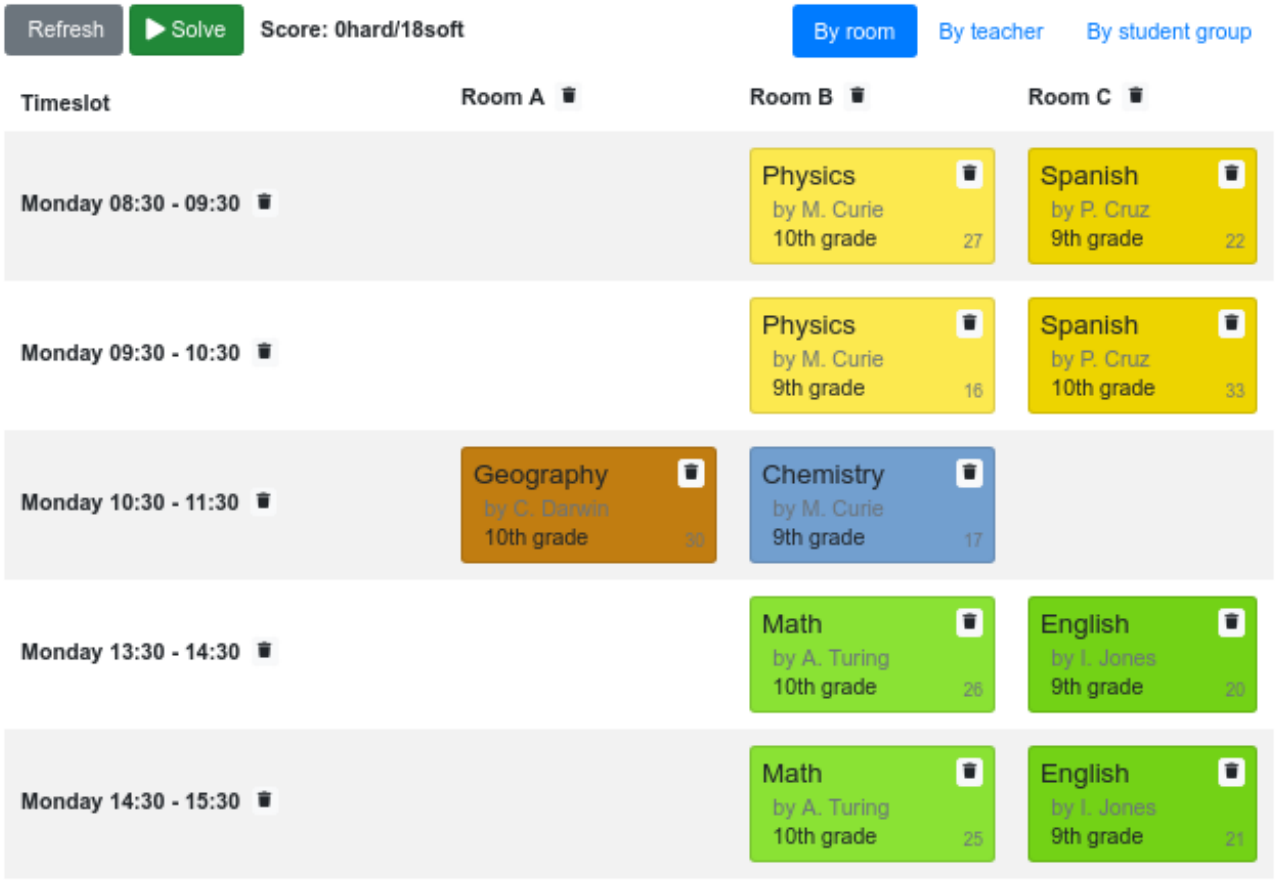
ポート 8081 でアプリケーションを実行するには、**-Dquarkus.http.port=8081** を前述のコマンドに追加します。

4. OptaPlanner 従業員スケジューラーを開始するには、Web ブラウザーに次の URL を入力します。

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

第17章 SPRING BOOT 上の RED HAT BUILD OF OPTAPLANNER: 時間割のクイックスタートガイド

本書では、OptaPlanner の制約解決人工知能 (AI) を使用して Spring Boot アプリケーションを作成するプロセスを説明します。生徒と教師の時間割を最適化する REST アプリケーションを構築します。



サービスは、AI を使用して、以下のハードおよびソフトの **スケジュール制約** に準拠し、**Lesson** インスタンスを **Timeslot** インスタンスと **Room** インスタンスに自動的に割り当てます。

- 1 部屋に同時に割り当てることができる授業は、最大 1 コマです。
- 教師が同時に一度に行うことができる授業は最大 1 回です。
- 生徒は同時に出席できる授業は最大 1 コマです。
- 教師は、1 つの部屋での授業を希望します。
- 教師は、連続した授業を好み、授業間に時間が空くのを嫌います。

数学的に考えると、学校の時間割は **NP 困難** の問題であります。つまり、スケーリングが困難です。総当たり攻撃で考えられる組み合わせを単純にすべて反復すると、スーパーコンピューターを使用したとしても、非自明的なデータセットを取得するのに数百年かかります。幸い、OptaPlanner などの AI 制約ソルバーには、妥当な時間内にほぼ最適なソリューションを提供する高度なアルゴリズムがあります。妥当な期間として考慮される内容は、問題の目的によって異なります。

前提条件

- OpenJDK 11 以降がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスマーポータル (ログインが必要) の ソフトウェアダウンロード ページから入手できます。

- Apache Maven 3.8 以降がインストールされている。Maven は Apache Maven Project の Web サイトから入手できます。
- IntelliJ IDEA、VSCode、Eclipse などの IDE が利用できる。

17.1. SPRING BOOT 時間割のクイックスタートのダウンロードおよびビルド

Spring Boot 製品を備えた Red Hat build of OptaPlanner 向けの授業の時間割プロジェクトの完全な例を表示するには、Red Hat カスタマーポータルからスターターアプリケーションをダウンロードします。

手順

1. Red Hat カスタマーポータルの Software Downloads ページに移動し (ログインが必要)、ドロップダウンオプションから製品およびバージョンを選択します。
   - **製品:** Red Hat build of OptaPlanner
   - **バージョン:** 8.38
2. **Red Hat build of OptaPlanner 8.38 クイックスタート**をダウンロードします。
3. **rhbop-8.38.0-optaplanner-quickstarts-sources.zip** ファイルを展開します。
   展開先の **org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/use-cases/school-timetabling** ディレクトリーには、サンプルソースコードが含まれています。
4. **org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/use-cases/school-timetabling** ディレクトリーに移動します。
5. OptaPlanner 8.38.0 Maven Repositroy の Red Hat ビルド(**rhbop-8.38.0-optaplanner-maven-repository.zip**) をダウンロードします。
6. **rhbop-8.38.0-optaplanner-maven-repository.zip** ファイルを展開します。
7. **rhbop-8.38.0-optaplanner/maven-repository** サブディレクトリーの内容を **~/.m2/repository** ディレクトリーにコピーします。
8. **org.optaplanner.optaplanner-quickstarts-8.38.0.Final-redhat-00004/technology/java-spring-boot** ディレクトリーに移動します。
9. 以下のコマンドを入力して、Spring Boot の授業の時間割プロジェクトをビルドします。

```
mvn clean install -DskipTests
```

10. Spring Boot の授業の時間割プロジェクトをビルドするには、以下のコマンドを入力します。

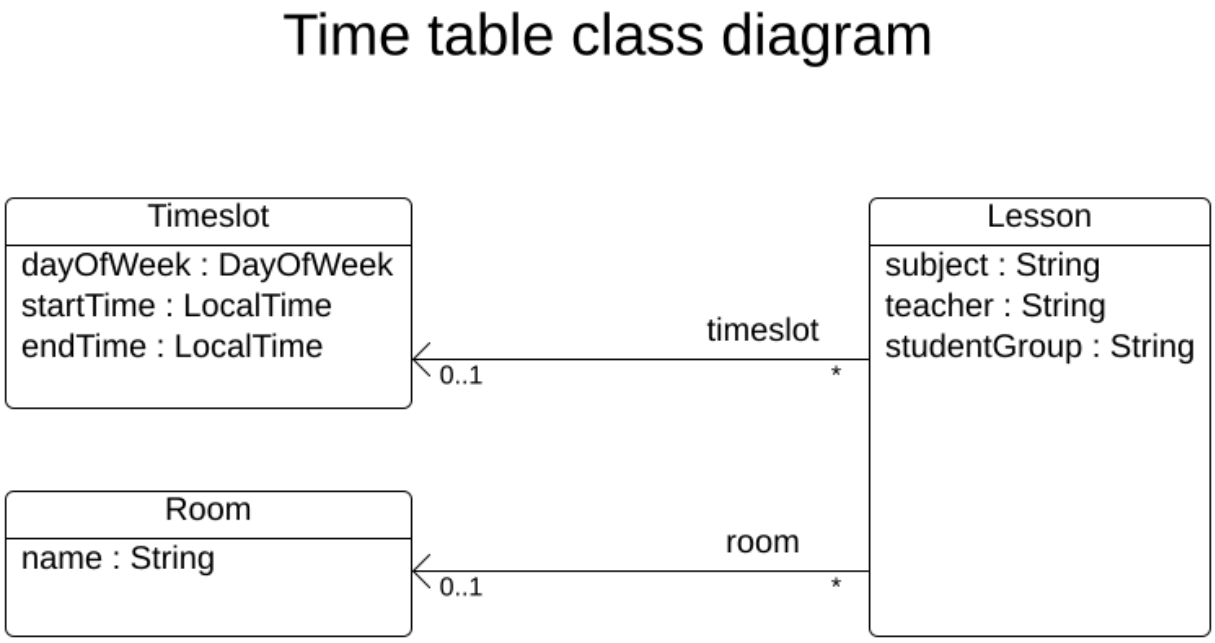
```
mvn spring-boot:run -DskipTests
```

11. プロジェクトを表示するには、Web ブラウザーで以下の URL を入力します。

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

17.2. ドメインオブジェクトのモデル化

Red Hat build of OptaPlanner の時間割プロジェクトの目標は、レッスンごとに時間枠と部屋に割り当てることです。これには、次の図に示すように、**Timeslot**、**Lesson**、および **Room** の 3 つのクラスを追加します。



Timeslot

Timeslot クラスは、**Monday 10:30 - 11:30**、**Tuesday 13:30 - 14:30** など、授業の長さを表します。この例では、時間枠はすべて同じ長さ (期間) で、昼休みまたは他の休憩時間にはこのスロットはありません。

高校のスケジュールは毎週 (同じ内容が) 繰り返されるだけなので、時間枠には日付がありません。また、継続的プランニング は必要ありません。解決時に **Timeslot** インスタンスが変更しないため、Timeslot は **問題ファクト** と呼ばれます。このようなクラスには OptaPlanner 固有のアノテーションは必要ありません。

Room

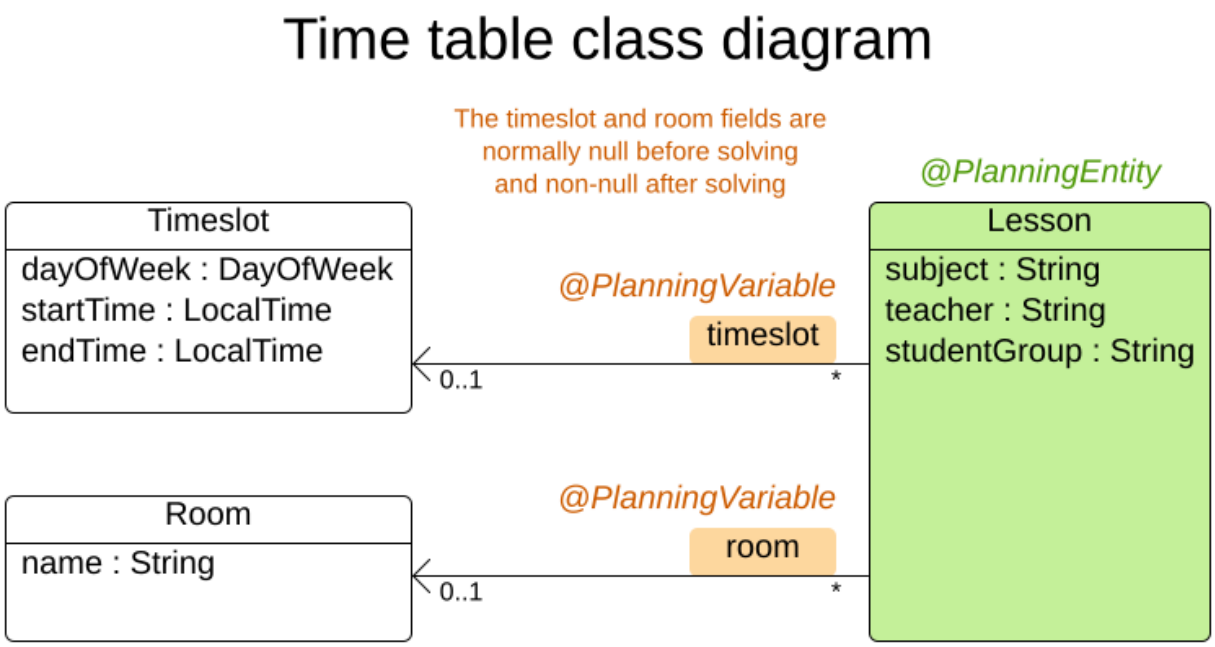
Room クラスは、**Room A**、**Room B** など、授業の場所を表します。以下の例では、どの部屋も定員制限がなく、すべての授業に対応できます。

Room インスタンスは解決時に変化しないため、**Room** は **問題ファクト** でもあります。

Lesson

授業中 (**Lesson** クラスで表現)、教師は複数の生徒に **Math by A.Turing for 9th grade**、**Chemistry by M.Curie for 10th grade** などの教科を指導します。ある教科について、毎週複数回、同じ教師が同じ生徒グループを指導する場合は、**Lesson** インスタンスが複数使用されますが、それらは **id** で識別可能です。たとえば、9 年生の場合は、1 週間に 6 回数学の授業があります。

解決中に、OptaPlanner は、**Lesson** クラスの **timeslot** フィールドと **room** フィールドを変更して、各授業を、時間枠 1 つ、部屋 1 つに割り当てます。OptaPlanner はこれらのフィールドを変更するため、**Lesson** は **プランニングエンティティー** となります。



前図では、オレンジのフィールド以外のほぼすべてのフィールドに、入力データが含まれています。授業の **timeslot** フィールドと **room** フィールドは、入力データに割り当てられておらず (**null**)、出力データに割り当てられて (**null** ではない) います。Red Hat build of OptaPlanner は、解決時にこれらのフィールドを変更します。このようなフィールドはプランニング変数と呼ばれます。このフィールドを OptaPlanner に認識させるには、**timeslot** フィールドと **room** のフィールドに **@PlanningVariable** アノテーションが必要です。このフィールドに含まれる **Lesson** クラスには、**@PlanningEntity** アノテーションが必要です。

手順

1. **src/main/java/com/example/domain/Timeslot.java** クラスを作成します。

```
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;
    }

}
```

後述しているように、**toString()** メソッドで出力を短くするため、OptaPlanner の **DEBUG** ログまたは **TRACE** ログの読み取りが簡単になっています。

2. **src/main/java/com/example/domain/Room.java** クラスを作成します。

```
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. **src/main/java/com/example/domain/Lesson.java** クラスを作成します。

```
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 クラスには **@PlanningEntity** アノテーションが含まれており、その中にプランニング変数が 1 つ以上含まれているため、OptaPlanner はこのクラスが解決時に変化することを認識します。

timeslot フィールドには **@PlanningVariable** アノテーションがあるため、OptaPlanner は、このフィールドの値が変化することを認識しています。このフィールドに割り当てることのできる **Timeslot** インスタンスを見つけ出すために、OptaPlanner は **valueRangeProviderRefs** プロパティーを使用して値の範囲プロバイダーと連携し、**List<Timeslot>** を提供して選択できるようにします。値の範囲プロバイダーに関する詳細は、「プランニングソリューションでのドメインオブジェクトの収集」 を参照してください。

room フィールドにも、同じ理由で **@PlanningVariable** アノテーションが含まれます。

17.3. 制約の定義およびスコアの計算

問題の解決時に **スコア** で導かれた解の質を表します。スコアが高いほど質が高くなります。Red Hat build of OptaPlanner は、利用可能な時間内で見つかった解の中から最高スコアのものを探し出します。これが **最適** 解である可能性があります。

時間割の例のユースケースでは、ハードとソフト制約を使用しているため、**HardSoftScore** クラスでスコアを表します。

- ハード制約は、絶対に違反しないでください。たとえば、**部屋に同時に割り当てることができる授業は、最大 1 コマです。**
- ソフト制約は、違反しないようにしてください。たとえば、**教師は、1 つの部屋での授業を希望します。**

ハード制約は、他のハード制約と比べて、重み付けを行います。ソフト制約は、他のソフト制約と比べて、重み付けを行います。ハード制約は、それぞれの重みに関係なく、常にソフト制約よりも高くなります。

EasyScoreCalculator クラスを実装して、スコアを計算できます。

```
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);
    }

}
```

残念ながら、この解は漸増的ではないので、適切にスケーリングされません。授業が別の時間枠や教室に割り当てられるたびに、全授業が再評価され、新しいスコアが計算されます。

src/main/java/com/example/solver/TimeTableConstraintProvider.java クラスを作成して、漸増的スコア計算を実行すると、解がより優れたものになります。このクラスは、Java 8 Streams と SQL を基にした OptaPlanner の ConstraintStream API を使用します。**ConstraintProvider** は、**EasyScoreCalculator** と比べ、スケーリングの規模が遥かに大きくなっています (**O** (n^2) ではなく **O** (n))。

手順

以下の **src/main/java/com/example/solver/TimeTableConstraintProvider.java** クラスを作成します。

```
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. プランニングソリューションでのドメインオブジェクトの収集

TimeTable インスタンスは、単一データセットの **Timeslot** インスタンス、**Room** インスタンス、および **Lesson** インスタンスをラップします。さらに、このインスタンスは、特定のプランニング変数の状態を持つ授業がすべて含まれているため、このインスタンスは **プランニングソリューション** となり、スコアが割り当てられます。

- 授業がまだ割り当てられていない場合は、スコアが **-4init/0hard/0soft** のソリューションなど、**初期化されていない** ソリューションとなります。
- ハード制約に違反する場合、スコアが **-2hard/-3soft** のソリューションなど、**実行不可** なソリューションとなります。
- 全ハード制約に準拠している場合は、スコアが **0hard/-7soft** など、**実行可能** なソリューションとなります。

TimeTable クラスには **@PlanningSolution** アノテーションが含まれているため、Red Hat build of OptaPlanner はこのクラスに全入出力データが含まれていることを認識します。

具体的には、このクラスは問題の入力です。

- 全時間枠が含まれる **timeslotList** フィールド
  - これは、解決時に変更されないため、問題ファクトリーストです。
- 全部屋が含まれる **roomList** フィールド
  - これは、解決時に変更されないため、問題ファクトリーストです。
- 全授業が含まれる **lessonList** フィールド
  - これは、解決時に変更されるため、プランニングエンティティーです。
  - 各 **Lesson**:
    - **timeslot** フィールドおよび **room** フィールドの値は通常、**null** で未割り当てです。これらの値は、プランニング変数です。
    - **subject**、**teacher**、**studentGroup** などの他のフィールドは入力されます。これらのフィールドは問題プロパティーです。

ただし、このクラスはソリューションの出力でもあります。

- **Lesson** インスタンスごとの **lessonList** フィールドには、解決後は null ではない **timeslot** フィールドと **room** フィールドが含まれます。
- 出力ソリューションの品質を表す **score** フィールド (例: **0hard/-5soft**)

手順

src/main/java/com/example/domain/TimeTable.java クラスを作成します。

```
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;
    }

}
```

値の範囲のプロバイダー

timeslotList フィールドは、値の範囲プロバイダーです。これは **Timeslot** インスタンスを保持し、OptaPlanner がこのインスタンスを選択して、**Lesson** インスタンスの **timeslot** フィールドに割り当てることができます。**timeslotList** フィールドには **@ValueRangeProvider** アノテーションがあり、**id** を、**Lesson** の **@PlanningVariable** の **valueRangeProviderRefs** に一致させます。

同じロジックに従い、**roomList** フィールドにも **@ValueRangeProvider** アノテーションが含まれています。

問題ファクトとプランニングエンティティーのプロパティー

さらに OptaPlanner は、変更可能な **Lesson** インスタンス、さらに **TimeTableConstraintProvider** によるスコア計算に使用する **Timeslot** インスタンスと **Room** インスタンスを取得する方法を把握しておく必要があります。

timeslotList フィールドと **roomList** フィールドには **@ProblemFactCollectionProperty** アノテーションが含まれているため、**TimeTableConstraintProvider** はこれらのインスタンスから選択できます。

lessonList には **@PlanningEntityCollectionProperty** アノテーションが含まれているため、OptaPlanner は解決時に変更でき、**TimeTableConstraintProvider** はこの中から選択できます。

17.5. TIMETABLE サービスの作成

これですべて組み合わせ、REST サービスを作成する準備ができました。しかし、REST スレッドで計画問題を解決すると、HTTP タイムアウトの問題が発生します。そのため、Spring Boot スターターでは **SolverManager** を注入することで、個別のスレッドプールでソルバーを実行して複数のデータセットを並行して解決できます。

手順

src/main/java/com/example/solver/TimeTableController.java クラスを作成します。

```
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;
    }

}
```

この例では、初期実装はソルバーが完了するのを待つので、HTTP タイムアウトがまだ発生します。**complete** 実装を使用することで、より適切に HTTP タイムアウトを回避できます。

17.6. ソルバー終了時間の設定

プランニングアプリケーションに終了設定または終了イベントがない場合、理論的には永続的に実行されることになり、実際には HTTP タイムアウトエラーが発生します。これが発生しないようにするには、**optaplanner.solver.termination.spent-limit** パラメーターを使用して、アプリケーションが終了してからの時間を指定します。多くのアプリケーションでは、この時間を最低でも 5 分 (**5m**) に設定します。ただし、時間割の例では、解決時間を 5 分に制限すると、期間が十分に短いため HTTP タイムアウトを回避できます。

手順

以下の内容を含む **src/main/resources/application.properties** ファイルを作成します。

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

17.7. アプリケーションを実行可能にする手順

Red Hat build of OptaPlanner Spring Boot の時間割プロジェクトを完了すると、標準 Java **main()** メソッドで駆動する 1 つの実行可能 JAR ファイルにすべてをパッケージ化します。

前提条件

- これで OptaPlanner Spring Boot の時間割プロジェクトが完成しました。

手順

1. 以下の内容を含む **TimeTableSpringBootApp.java** クラスを作成します。

```
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. Spring Initializr で作成された **src/main/java/com/example/DemoApplication.java** クラスは **TimeTableSpringBootApp.java** クラスに置き換えます。

3. 通常の Java アプリケーションのメインクラスとして **TimeTableSpringBootApp.java** クラスを実行します。

17.7.1. 時間割アプリケーションの試行

Red Hat build of OptaPlanner Spring Boot 時間割アプリケーションの起動後に、任意の REST クライアントで REST サービスをテストできます。この例では Linux **curl** コマンドを使用して POST 要求を送信します。

前提条件

- OptaPlanner Spring Boot アプリケーションが実行中である。

手順

以下のコマンドを入力します。

```
$ 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"}]}'
```

約 5 秒後 (**application.properties** で定義された終了時間) に、サービスにより、以下の例のような出力が返されます。

```
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"}
```

アプリケーションにより、4 つの授業がすべて 2 つの時間枠、そして 2 つある部屋のうちの 1 つに割り当てられている点に注目してください。また、すべてのハード制約に準拠することに注意してください。たとえば、M. Curie の 2 つの授業は異なる時間スロットにあります。

サーバー側で **info** ログに、この 5 秒間で OptaPlanner が行った内容が記録されます。

```
... 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. アプリケーションのテスト

適切なアプリケーションにはテストが含まれます。以下の例では、Red Hat build of OptaPlanner Spring Boot 時間割アプリケーションをテストします。このアプリケーションは、JUnit テストを使用してテストのデータセットを生成し、**TimeTableController** に送信して解決します。

手順

以下の内容を含む **src/test/java/com/example/solver/TimeTableControllerTest.java** クラスを作成します。

```
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);
    }

}
```

このテストは、解決後にすべての授業がタイムスロットと部屋に割り当てられていることを確認します。また、実行可能解 (ハード制約の違反なし) も確認します。

通常、ソルバーは 200 ミリ秒未満で実行可能解を検索します。**@SpringBootTest** アノテーションの **properties** が、実行可能なソリューション (**0hard/*soft**) が見つかると同時に、ソルバーの終了を上書きします。こうすることで、ユニットテストが任意のハードウェアで実行される可能性があるため、ソルバーの時間をハードコード化するのを回避します。このアプローチを使用することで、動きが遅いシステムであっても、実行可能なソリューションを検索するのに十分な時間だけテストが実行されます。ただし、高速マシンでも、厳密に必要とされる時間よりもミリ秒単位で長く実行されることはありません。

17.7.3. ロギング

Red Hat build of OptaPlanner Spring Boot の時間割アプリケーションアプリケーションを完了後にロギング情報を使用すると、**ConstraintProvider** で制約が微調整しやすくなります。**info** ログファイルでスコア計算の速度を確認して、制約に加えた変更の影響を評価します。デバッグモードでアプリケーションを実行して、アプリケーションが行う手順をすべて表示するか、追跡ログを使用して全手順および動きをロギングします。

手順

1. 時間割アプリケーションを一定の時間 (例: 5 分) 実行します。
2. 以下の例のように、**log** ファイルのスコア計算の速度を確認します。

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

3. 制約を変更して、同じ時間、プランニングアプリケーションを実行し、**log** ファイルに記録されているスコア計算速度を確認します。
4. アプリケーションをデバッグモードで実行して、全手順をログに記録します。
   - コマンドラインからデバッグモードを実行するには、**-D** システムプロパティーを使用します。
   - **application.properties** ファイルのロギングを変更するには、以下の行をこのファイルに追加します。

```
logging.level.org.optaplanner=debug
```

   以下の例では、デバッグモードでの **log** ファイルの出力を表示します。

```
... 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. **trace** ロギングを使用して、全手順、および手順ごとの全動きを表示します。

17.8. データベースと UI 統合の追加

Spring Boot を使用して Red Hat build of OptaPlanner アプリケーションの例を作成したら、データベースと UI 統合を追加します。

前提条件

- OptaPlanner Spring Boot の時間割サンプルが作成されている。

手順

1. **Timeslot**、**Room**、および **Lesson** の Java Persistence API (JPA) リポジトリーを作成します。JPA リポジトリーの作成に関する情報は、Spring の Web サイトの Accessing Data with JPA を参照してください。
2. REST で JPA リポジトリーを公開します。リポジトリーの公開に関する情報は、Spring の Web サイトの Accessing JPA Data with REST を参照してください。
3. **TimeTableRepository** Facade をビルドして、1 回のトランザクションで **TimeTable** を読み取り、書き込みます。
4. 以下の例のように **TimeTableController** を調整します。

```
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);
    }

}
```

簡素化するために、このコードは **TimeTable** インスタンスを 1 つだけ処理しますが、マルチテナントを有効にして、異なる高校の **TimeTable** インスタンスを複数、平行して処理することも簡単にできます。

getTimeTable() メソッドは、データベースから最新の時間割を返します。このメソッドは、**ScoreManager** (自動注入) を使用して、UI でスコアを表示できるように、対象の時間割のスコアを計算します。

solve() メソッドは、ジョブを開始して、現在の時間割を解決し、時間枠と部屋の割り当てをデータベースに保存します。このメソッドは、**SolverManager.solveAndListen()** メソッドを使用して、中間の最適解をリッスンし、それに合わせてデータベースを更新します。こうすることで、バックエンドで解決しながら、UI で進捗を表示できます。

5. **solve()** メソッドが即座に返されるようになったので、以下の例のように **TimeTableControllerTest** を調整します。

```
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. ソルバーの解決が完了するまで、最新のソリューションをポーリングします。
7. 時間割を視覚化するには、これらの REST メソッドの上に適切な Web UI を構築します。

17.9. MICROMETER と PROMETHEUS を使用して学校の時間割を監視する OPTAPLANNER SPRING BOOT アプリケーション

OptaPlanner は、Java アプリケーション用のメトリック計測ライブラリーである Micrometer を介してメトリックを公開します。Prometheus で Micrometer を使用して、学校の時間割アプリケーションで OptaPlanner ソルバーを監視できます。

前提条件

- Spring Boot OptaPlanner 学校の時間割アプリケーションを作成しました。
- Prometheus がインストールされている。Prometheus のインストールについては、Prometheus の Web サイトを参照してください。

手順

1. **technology/java-spring-boot** ディレクトリーに移動します。
2. 学校の時間割 **pom.xml** ファイルに Micrometer Prometheus 依存関係を追加します。

```
<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. 次のプロパティーを application.properties ファイルに追加します。

```
management.endpoints.web.exposure.include=metrics,prometheus
```

4. 学校の時間割アプリケーションを開始します。

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

5. Web ブラウザーで **http://localhost:8080/actuator/prometheus** を開きます。

第18章 OPTAPLANNER と JAVA の RED HAT ビルド: 学校の時間割のクイックスタートガイド

本書では、OptaPlanner の制約解決人工知能 (AI) を使用してシンプル Java アプリケーションを作成するプロセスを説明します。生徒と教師のために学校の時間割を最適化するコマンドラインアプリケーションを作成します。

```
...
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  |------------|------------|------------|------------|
```

アプリケーションは、AI を使用してハードスケジュールとソフトスケジュールの **制約** を順守することにより、**Lesson** インスタンスを **タイムスロット** インスタンスと **Room** インスタンスに自動的に割り当てます。次に例を示します。

- 1 部屋に同時に割り当てることができる授業は、最大 1 コマです。
- 教師が同時に一度に行うことができる授業は最大 1 回です。
- 生徒は同時に出席できる授業は最大 1 コマです。
- 教師は、すべてのレッスンを同じ部屋で教えることを好みます。
- 教師は、連続した授業を好み、授業間に時間が空くのを嫌います。
- 生徒は、同じ科目の連続したレッスンを嫌います。

数学的に考えると、学校の時間割は **NP 困難** の問題であります。これは、スケーリングが困難であることを意味します。すべての可能な組み合わせを単純に総当たりで反復すると、スーパーコンピューター上でさえ、自明ではないデータセットに対して数百万年かかります。幸い、OptaPlanner などの AI 制約ソルバーには、妥当な時間内にほぼ最適なソリューションを提供する高度なアルゴリズムがあります。

前提条件

- OpenJDK (JDK) 11 がインストールされている。Red Hat ビルドの Open JDK は Red Hat カスマーポータル (ログインが必要) の ソフトウェアダウンロード ページから入手できます。

- Apache Maven 3.6 以降がインストールされている。Maven は Apache Maven Project の Web サイトから入手できます。
- IntelliJ IDEA、VSCode、Eclipse などの IDE

18.1. MAVEN または GRADLE ビルドファイルの作成および依存関係の追加

OptaPlanner 学校の時間割アプリケーションには、Maven または Gradle を使用できます。ビルドファイルを作成したら、次の依存関係を追加します。

- 学校の時間割問題を解決するための **optaplanner-core** (コンパイルスコープ)
- **optaplanner-test** (テストスコープ) から JUnit への学校の時間割の制約のテスト
- OptaPlanner が実行するステップを表示するための **logback-classic** (ランタイムスコープ) などの実装

手順

1. Maven または Gradle ビルドファイルを作成します。
2. **optaplanner-core**、**optaplanner-test**、および **logback-classic** の依存関係をビルドファイルに追加します。
   - Maven の場合、次の依存関係を **pom.xml** ファイルに追加します。

```
<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>
```

次の例は、完全な **pom.xml** ファイルを示しています。

```
<?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>
```

- Gradle の場合、次の依存関係を **gradle.build** ファイルに追加します。

```
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}"
}
```

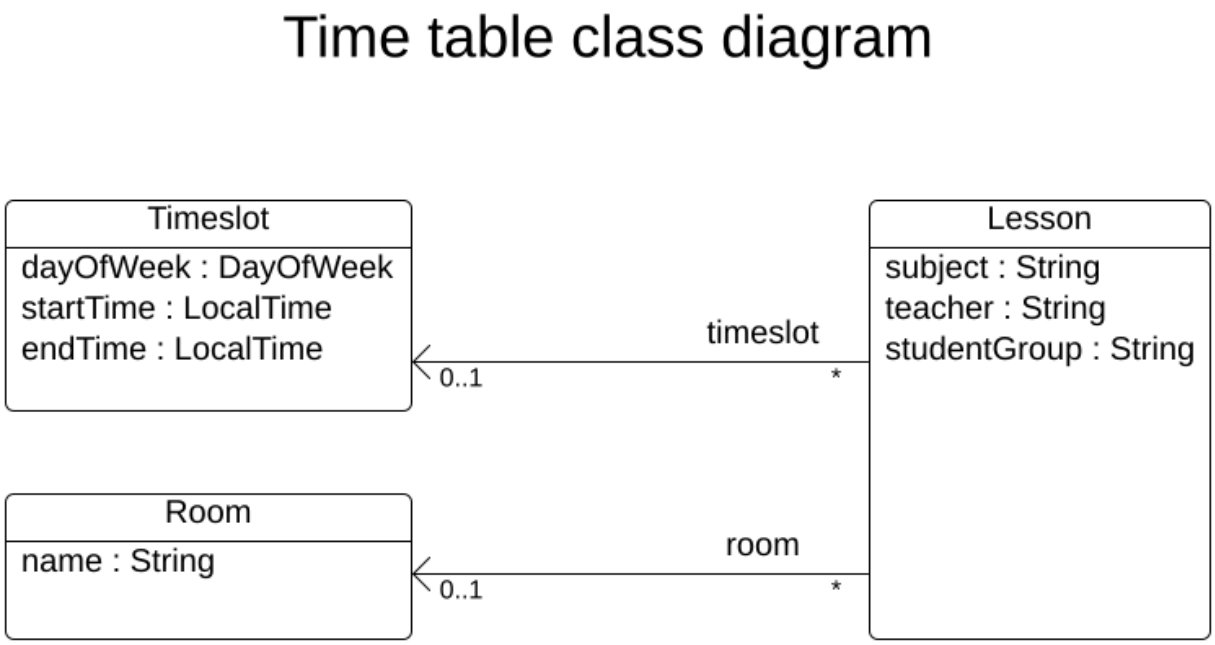
次の例は、完成した **gradle.build** ファイルを示しています。

```
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. ドメインオブジェクトのモデル化

Red Hat build of OptaPlanner の時間割プロジェクトの目標は、レッスンごとに時間枠と部屋に割り当てることです。これには、次の図に示すように、**Timeslot**、**Lesson**、および **Room** の 3 つのクラスを追加します。



Timeslot

Timeslot クラスは、**Monday 10:30 - 11:30**、**Tuesday 13:30 - 14:30** など、授業の長さを表します。この例では、時間枠はすべて同じ長さ (期間) で、昼休みまたは他の休憩時間にはこのスロットはありません。

高校のスケジュールは毎週 (同じ内容が) 繰り返されるだけなので、時間枠には日付がありません。また、継続的プランニング は必要ありません。解決時に **Timeslot** インスタンスが変更しないため、Timeslot は **問題ファクト** と呼ばれます。このようなクラスには OptaPlanner 固有のアノテーションは必要ありません。

Room

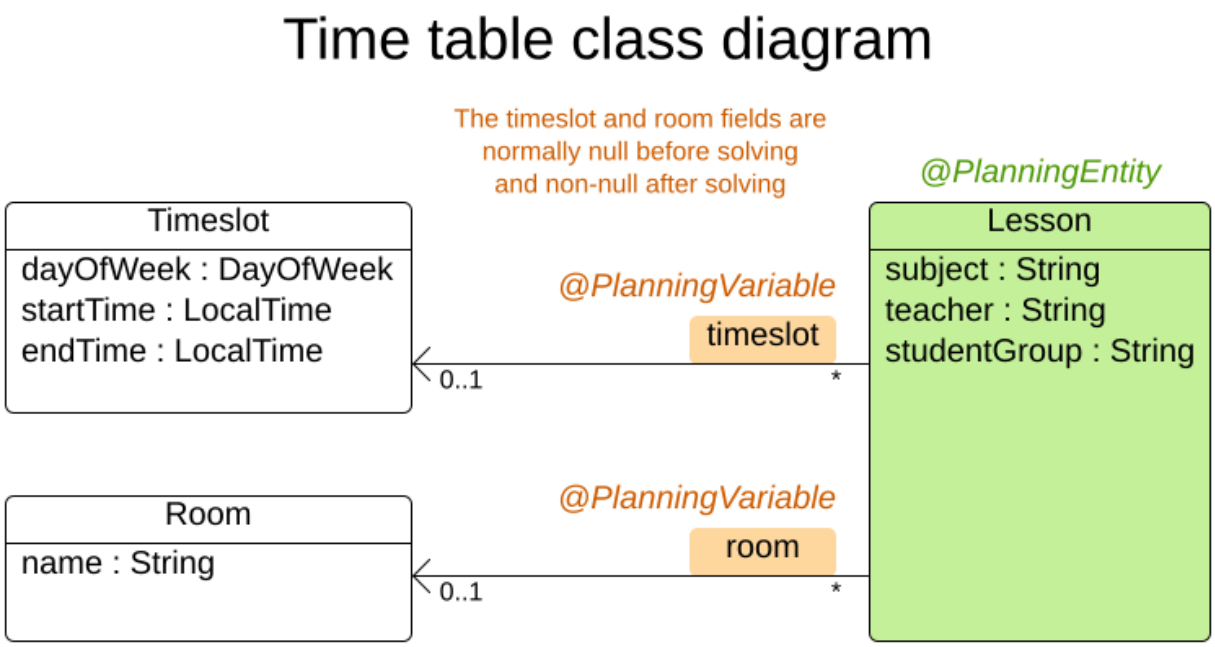
Room クラスは、**Room A**、**Room B** など、授業の場所を表します。以下の例では、どの部屋も定員制限がなく、すべての授業に対応できます。

Room インスタンスは解決時に変化しないため、**Room** は **問題ファクト** でもあります。

Lesson

授業中 (**Lesson** クラスで表現)、教師は複数の生徒に **Math by A.Turing for 9th grade**、**Chemistry by M.Curie for 10th grade** などの教科を指導します。ある教科について、毎週複数回、同じ教師が同じ生徒グループを指導する場合は、**Lesson** インスタンスが複数使用されますが、それらは **id** で識別可能です。たとえば、9 年生の場合は、1 週間に 6 回数学の授業があります。

解決中に、OptaPlanner は、**Lesson** クラスの **timeslot** フィールドと **room** フィールドを変更して、各授業を、時間枠 1 つ、部屋 1 つに割り当てます。OptaPlanner はこれらのフィールドを変更するため、**Lesson** は **プランニングエンティティー** となります。



前図では、オレンジのフィールド以外のほぼすべてのフィールドに、入力データが含まれています。授業の **timeslot** フィールドと **room** フィールドは、入力データに割り当てられておらず (**null**)、出力データに割り当てられて (**null** ではない) います。Red Hat build of OptaPlanner は、解決時にこれらのフィールドを変更します。このようなフィールドはプランニング変数と呼ばれます。このフィールドを OptaPlanner に認識させるには、**timeslot** フィールドと **room** のフィールドに **@PlanningVariable** アノテーションが必要です。このフィールドに含まれる **Lesson** クラスには、**@PlanningEntity** アノテーションが必要です。

手順

1. **src/main/java/com/example/domain/Timeslot.java** クラスを作成します。

```
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;
    }

}
```

後述しているように、**toString()** メソッドで出力を短くするため、OptaPlanner の **DEBUG** ログまたは **TRACE** ログの読み取りが簡単になっています。

2. **src/main/java/com/example/domain/Room.java** クラスを作成します。

```
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. **src/main/java/com/example/domain/Lesson.java** クラスを作成します。

```
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 クラスには **@PlanningEntity** アノテーションが含まれており、その中にプランニング変数が 1 つ以上含まれているため、OptaPlanner はこのクラスが解決時に変化することを認識します。

timeslot フィールドには **@PlanningVariable** アノテーションがあるため、OptaPlanner は、このフィールドの値が変化することを認識しています。このフィールドに割り当てることのできる **Timeslot** インスタンスを見つけ出すために、OptaPlanner は **valueRangeProviderRefs** プロパティーを使用して値の範囲プロバイダーと連携し、**List<Timeslot>** を提供して選択できるようにします。値の範囲プロバイダーに関する詳細は、「プランニングソリューションでのドメインオブジェクトの収集」 を参照してください。

room フィールドにも、同じ理由で **@PlanningVariable** アノテーションが含まれます。

18.3. 制約の定義およびスコアの計算

問題の解決時に **スコア** で導かれた解の質を表します。スコアが高いほど質が高くなります。Red Hat build of OptaPlanner は、利用可能な時間内で見つかった解の中から最高スコアのものを探し出します。これが **最適** 解である可能性があります。

時間割の例のユースケースでは、ハードとソフト制約を使用しているため、**HardSoftScore** クラスでスコアを表します。

- ハード制約は、絶対に違反しないでください。たとえば、**部屋に同時に割り当てることができる授業は、最大 1 コマです。**
- ソフト制約は、違反しないようにしてください。たとえば、**教師は、1 つの部屋での授業を希望します。**

ハード制約は、他のハード制約と比べて、重み付けを行います。ソフト制約は、他のソフト制約と比べて、重み付けを行います。ハード制約は、それぞれの重みに関係なく、常にソフト制約よりも高くなります。

EasyScoreCalculator クラスを実装して、スコアを計算できます。

```
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);
    }

}
```

残念ながら、この解は漸増的ではないので、適切にスケーリングされません。授業が別の時間枠や教室に割り当てられるたびに、全授業が再評価され、新しいスコアが計算されます。

src/main/java/com/example/solver/TimeTableConstraintProvider.java クラスを作成して、漸増的スコア計算を実行すると、解がより優れたものになります。このクラスは、Java 8 Streams と SQL を基にした OptaPlanner の ConstraintStream API を使用します。**ConstraintProvider** は、**EasyScoreCalculator** と比べ、スケーリングの規模が遥かに大きくなっています (**O** (n^2) ではなく **O** (n))。

手順

以下の **src/main/java/com/example/solver/TimeTableConstraintProvider.java** クラスを作成します。

```
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. プランニングソリューションでのドメインオブジェクトの収集

TimeTable インスタンスは、単一データセットの **Timeslot** インスタンス、**Room** インスタンス、および **Lesson** インスタンスをラップします。さらに、このインスタンスは、特定のプランニング変数の状態を持つ授業がすべて含まれているため、このインスタンスは **プランニングソリューション** となり、スコアが割り当てられます。

- 授業がまだ割り当てられていない場合は、スコアが **-4init/0hard/0soft** のソリューションなど、**初期化されていない** ソリューションとなります。
- ハード制約に違反する場合、スコアが **-2hard/-3soft** のソリューションなど、**実行不可** なソリューションとなります。
- 全ハード制約に準拠している場合は、スコアが **0hard/-7soft** など、**実行可能** なソリューションとなります。

TimeTable クラスには **@PlanningSolution** アノテーションが含まれているため、Red Hat build of OptaPlanner はこのクラスに全入出力データが含まれていることを認識します。

具体的には、このクラスは問題の入力です。

- 全時間枠が含まれる **timeslotList** フィールド
  - これは、解決時に変更されないため、問題ファクトリーストです。
- 全部屋が含まれる **roomList** フィールド
  - これは、解決時に変更されないため、問題ファクトリーストです。
- 全授業が含まれる **lessonList** フィールド
  - これは、解決時に変更されるため、プランニングエンティティーです。
  - 各 **Lesson**:
    - **timeslot** フィールドおよび **room** フィールドの値は通常、**null** で未割り当てです。これらの値は、プランニング変数です。
    - **subject**、**teacher**、**studentGroup** などの他のフィールドは入力されます。これらのフィールドは問題プロパティーです。

ただし、このクラスはソリューションの出力でもあります。

- **Lesson** インスタンスごとの **lessonList** フィールドには、解決後は null ではない **timeslot** フィールドと **room** フィールドが含まれます。
- 出力ソリューションの品質を表す **score** フィールド (例: **0hard/-5soft**)

手順

src/main/java/com/example/domain/TimeTable.java クラスを作成します。

```
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;
    }

}
```

値の範囲のプロバイダー

timeslotList フィールドは、値の範囲プロバイダーです。これは **Timeslot** インスタンスを保持し、OptaPlanner がこのインスタンスを選択して、**Lesson** インスタンスの **timeslot** フィールドに割り当てることができます。**timeslotList** フィールドには **@ValueRangeProvider** アノテーションがあり、**id** を、**Lesson** の **@PlanningVariable** の **valueRangeProviderRefs** に一致させます。

同じロジックに従い、**roomList** フィールドにも **@ValueRangeProvider** アノテーションが含まれています。

問題ファクトとプランニングエンティティーのプロパティー

さらに OptaPlanner は、変更可能な **Lesson** インスタンス、さらに **TimeTableConstraintProvider** によるスコア計算に使用する **Timeslot** インスタンスと **Room** インスタンスを取得する方法を把握しておく必要があります。

timeslotList フィールドと **roomList** フィールドには **@ProblemFactCollectionProperty** アノテーションが含まれているため、**TimeTableConstraintProvider** はこれらのインスタンスから選択できます。

lessonList には **@PlanningEntityCollectionProperty** アノテーションが含まれているため、OptaPlanner は解決時に変更でき、**TimeTableConstraintProvider** はこの中から選択できます。

18.5. TIMETABLEAPP.JAVA クラス

学校の時間割アプリケーションのすべてのコンポーネントを作成したら、それらをすべて **TimeTableApp.java** クラスにまとめます。

main() メソッドは以下のタスクを実行します。

1. **SolverFactory** を作成して、各データセットの **Solver** を構築します。
2. データセットをロードします。
3. **Solver.solve()** で解決します。
4. そのデータセットの解を視覚化します。

通常、アプリケーションには、解決する問題データセットごとに新しい **Solver** インスタンスを構築するための **SolverFactory** が 1 つあります。**SolverFactory** はスレッドセーフですが、**Solver** はそうではありません。学校の時間割アプリケーションの場合、データセットは 1 つしかないため、**Solver** インスタンスは 1 つだけです。

完成した **TimeTableApp.java** クラスは次のとおりです。

```
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 () メソッドは最初に **SolverFactory** を作成します。

```
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 の作成により、以前に作成した **@PlanningSolution** クラス、**@PlanningEntity** クラス、および **ConstraintProvider** クラスが登録されます。

終了設定または **terminationEarly()** イベントがない場合、ソルバーは永久に実行されます。これを避けるために、ソルバーは解決時間を 5 秒に制限します。

5 秒後、**main()** メソッドは問題をロードして解決し、解決策を出力します。

```
        // 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() メソッドはすぐには戻りません。最適解を返す前に 5 秒間実行されます。

OptaPlanner は、利用可能な終了時間内に見つかった**最適なソリューション**を返します。NP 困難な問題の性質上、特に大規模なデータセットの場合、最適解が最適ではない可能性があります。終了時間を増やして、より良い解決策を見つけてください。

generateDemoData() メソッドは、解決する学校の時間割の問題を生成します。

printTimetable() メソッドは時刻表をコンソールにきれいに出力するので、スケジュールが適切かどうかを視覚的に簡単に判断できます。

18.6. 学校の時間割アプリケーションの作成と実行

学校の時間割 Java アプリケーションのすべてのコンポーネントが完成したので、それらをすべて **TimeTableApp.java** クラスにまとめて実行する準備が整いました。

前提条件

- 学校の時間割アプリケーションに必要なすべてのコンポーネントを作成しました。

手順

1. **src/main/java/org/acme/schooltimetabling/TimeTableApp.java** クラスを作成します。

```
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. **TimeTableApp** クラスを通常の Java アプリケーションのメインクラスとして実行します。次の出力が得られるはずです。

```
...
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. コンソール出力を確認します。すべての厳しい制約に準拠しています
か?**TimeTableConstraintProvider** の **roomConflict** 制約をコメントアウトするとどうなります
か?

info ログは、OptaPlanner がその 5 秒間に何をしたかを示しています。

```
... 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. アプリケーションのテスト

適切なアプリケーションにはテストが含まれます。timetable プロジェクトで制約とソルバーをテストします。

18.7.1. 学校の時間割の制約をテストする

timetable プロジェクトの各制約を個別にテストするには、単体テストで **ConstraintVerifier** を使用します。これにより、各制約のコーナーケースが他のテストから分離されてテストされるため、適切なテストカバレッジで新しい制約を追加する際のメンテナンスが軽減されます。

このテストは、制約 **TimeTableConstraintProvider::roomConflict** が、同じ部屋で 3 つのレッスンを与えられ、そのうちの 2 つのレッスンが同じタイムスロットを持つ場合、一致の重み 1 でペナルティを課すことを検証します。したがって、制約の重みが **10hard** の場合、スコアは **-10hard** 減少します。

手順

src/test/java/org/acme/optaplanner/solver/TimeTableConstraintProviderTest.java クラスを作成します。

```
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);
    }

}
```

制約の重みが **ConstraintProvider** でハードコーディングされている場合でも、**ConstraintVerifier** がテスト中に制約の重みを無視することに注意してください。これは、実稼動に入る前に制約の重みが定期的に変更されるためです。このように、制約の重みの微調整によって単体テストが中断されることはありません。

18.7.2. 学校の時間割ソルバーをテストする

以下の例では、Red Hat build of Quarkus で Red Hat build of OptaPlanner の時間割プロジェクトをテストします。このアプリケーションは、JUnit テストを使用してテストのデータセットを生成し、**TimeTableController** に送信して解決します。

手順

1. 以下の内容を含む **src/test/java/com/example/rest/TimeTableResourceTest.java** クラスを作成します。

```
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);
    }

}
```

このテストは、解決後にすべての授業がタイムスロットと部屋に割り当てられていることを確認します。また、実行可能解 (ハード制約の違反なし) も確認します。

2. テストプロパティーを **src** / **main** / **resources** /**application.properties** ファイルに追加します。

```
# 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、ソルバーは 200 ミリ秒未満で実行可能解を検索します。**application.properties** が、実行可能なソリューション (**0hard/*soft**) が見つかると同時に終了するように、テスト中のソルバーの終了を上書きします。こうすることで、ユニットテストが任意のハードウェアで実行される可能性があるため、ソルバーの時間をハードコード化するのを回避します。このアプローチを使用することで、動きが遅いシステムであっても、実行可能なソリューションを検索するのに十分な時間だけテストが実行されます。ただし、高速システムでも、厳密に必要とされる時間よりもミリ秒単位で長く実行されることはありません。

18.8. ロギング

Red Hat build of OptaPlanner の時間割プロジェクトを完了後にロギング情報を使用すると、**ConstraintProvider** で制約が微調整しやすくなります。**info** ログファイルでスコア計算の速度を確認して、制約に加えた変更の影響を評価します。デバッグモードでアプリケーションを実行して、アプリケーションが行う手順をすべて表示するか、追跡ログを使用して全手順および動きをロギングします。

手順

1. 時間割アプリケーションを一定の時間 (例: 5 分) 実行します。
2. 以下の例のように、**log** ファイルのスコア計算の速度を確認します。

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

3. 制約を変更して、同じ時間、プランニングアプリケーションを実行し、**log** ファイルに記録されているスコア計算速度を確認します。
4. アプリケーションをデバッグモードで実行して、アプリケーションの全実行ステップをログに記録します。
   - コマンドラインからデバッグモードを実行するには、**-D** システムプロパティーを使用します。
   - デバッグモードを永続的に有効にするには、以下の行を **application.properties** ファイルに追加します。

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

以下の例では、デバッグモードでの **log** ファイルの出力を表示します。

```
... 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. **trace** ロギングを使用して、全手順、および手順ごとの全動きを表示します。

18.9. MICROMETER と PROMETHEUS を使用して学校の時間割を監視する OPTAPLANNER JAVA アプリケーション

OptaPlanner は、Java アプリケーション用のメトリック計測ライブラリーである Micrometer を介してメトリックを公開します。Prometheus で Micrometer を使用して、学校の時間割アプリケーションで OptaPlanner ソルバーを監視できます。

前提条件

- OptaPlanner 学校の時間割アプリケーションを Java で作成しました。
- Prometheus がインストールされている。Prometheus のインストールについては、Prometheus の Web サイトを参照してください。

手順

1. Micrometer Prometheus 依存関係を学校の時間割 **pom.xml** ファイルに追加します。**<MICROMETER_VERSION>** は、インストールした Micrometer のバージョンです。

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

注記

micrometer-core 依存関係も必要です。ただし、この依存関係は **optaplanner-core** 依存関係に含まれているため、**pom.xml** ファイルに追加する必要はありません。

2. 次の import ステートメントを **TimeTableApp.java** クラスに追加します。

```
import io.micrometer.core.instrument.Metrics;
import io.micrometer.prometheus.PrometheusConfig;
import io.micrometer.prometheus.PrometheusMeterRegistry;
```

3. **TimeTableApp.java** クラスのメインメソッドの先頭に次の行を追加して、ソリューションが開始する前に Prometheus が **com.sun.net.httpserver.HttpServer** からデータを破棄できるようにします。

```
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. 次の行を追加して、解決時間を制御します。解決時間を調整することで、解決に費やされた時間に基づいて指標がどのように変化するかを確認できます。

```
withTerminationSpentLimit(Duration.ofMinutes(5)));
```

5. 学校の時間割アプリケーションを開始します。

6. Web ブラウザーで **http://localhost:8080/prometheus** を開き、Prometheus で timetable アプリケーションを表示します。

7. 監視システムを開いて、OptaPlanner プロジェクトのメトリックを表示します。
次のメトリックが公開されます。

   - **optaplanner_solver_errors_total**: 測定開始以降に解決中に発生したエラーの総数。
   - **optaplanner_solver_solve_duration_seconds_active_count**: 現在解決しているソルバーの数。
   - **optaplanner_solver_solve_duration_seconds_max**: 現在アクティブなソルバーの実行時間が最も長い実行時間。
   - **optaplanner_solver_solve_duration_seconds_duration_sum**: アクティブな各ソルバーの解決時間の合計。たとえば、アクティブなソルバーが 2 つあり、一方が 3 分間実行され、もう一方が 1 分間実行されている場合、合計計算時間は 4 分です。

付録A バージョン情報

本書の最終更新日: 2023 年 7 月 14 日 (金)