Visualizing Indictments

Explaining the solution to the user and identifying deficient resources

Christopher Chianelli

September 1st, 2020

Outline

Outline

The Problem

You have created a solution for the optimization of your business resources using OptaPlanner. Upon showing the solution to your boss, you are immediately asked the following questions:

- Why isn't Amy working on Monday?
- Is it profitable to hire additional workers?
- What would happen if Bob worked five days in a row?

Given just the solution from OptaPlanner, it is hard to answer these questions since they require insight with how the problem interact with the constraints. We know the solution is the best OptaPlanner found in its allocated time, but we want to know *why* is the solution good.

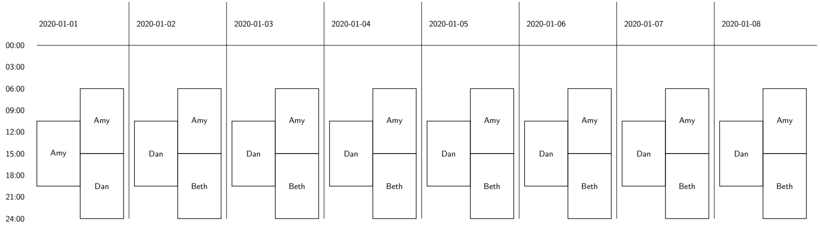
Score Visualization

Score Visualization refer to a variety of techniques used to visualize the score impact of planning entities. Use cases include:

- Being able to quickly tell what constraints a particular entity satisfies (ex: Shift A is assigned to Amy, and Amy desires to work during Shift A). This is useful for showing how good a solution is.
- Visualizing improvements that can be gained by additional resources. This is useful for business decisions such as purchasing an additional vehicle or hiring another worker.
- Checking for misbehaving constraints (for instance, you might be double counting a constraint, causing it to be weighed double its constraint weight).

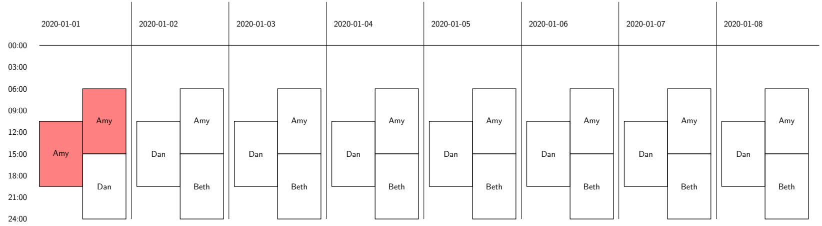
An Example

In the schedule below, are any employees working multiple shifts at the same time?



An Example

If we highlight any shifts that share an employee with another shift happening at the same time, the question become easier:



Outline

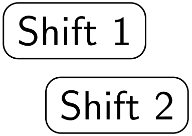
Constraint Matches

In OptaPlanner, every solution has a score. If you use incremental score calculation (Drools, Constraint Stream), the score is the sum of all the constraint matches.

Constraint Matches

Whenever a constraint is matched...

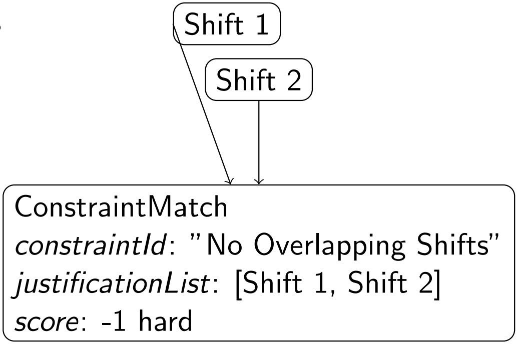
No Overlapping Shifts



Constraint Matches

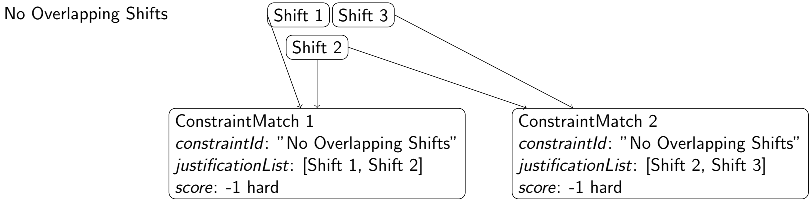
...a *ConstraintMatch* describing the constraint match is created.

No Overlapping Shifts


Shift 1
Shift 2
ConstraintMatch
constraintId: "No Overlapping Shifts"
justificationList: [Shift 1, Shift 2]
score: -1 hard

Constraint Matches

Entities/problem facts can be in multiple *ConstraintMatches*.

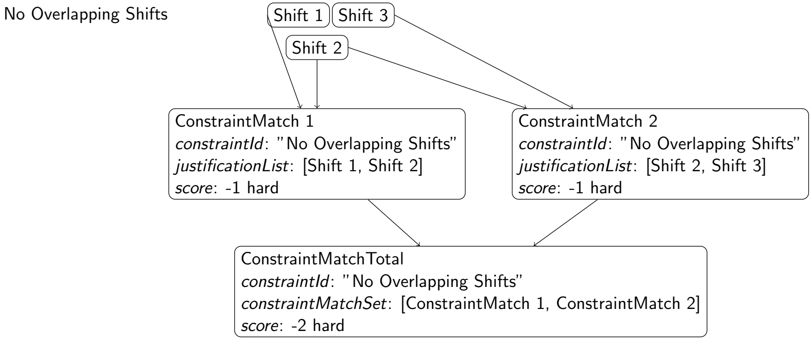


ConstraintMatch 1
constraintId: "No Overlapping Shifts"
justificationList: [Shift 1, Shift 2]
score: -1 hard

ConstraintMatch 2
constraintId: "No Overlapping Shifts"
justificationList: [Shift 2, Shift 3]
score: -1 hard

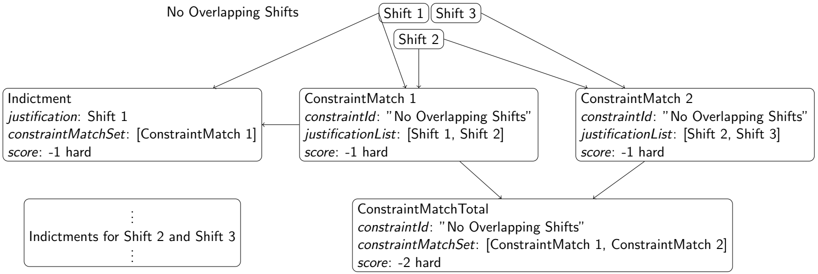
Constraint Matches

All the constraint matches for a specific constraint goes to a *ConstraintMatchTotal*.


No Overlapping Shifts
Shift 1
Shift 3
Shift 2
ConstraintMatch 1
constraintId: "No Overlapping Shifts"
justificationList: [Shift 1, Shift 2]
score: -1 hard
ConstraintMatch 2
constraintId: "No Overlapping Shifts"
justificationList: [Shift 2, Shift 3]
score: -1 hard
ConstraintMatchTotal
constraintId: "No Overlapping Shifts"
constraintMatchSet: [ConstraintMatch 1, ConstraintMatch 2]
score: -2 hard

Constraint Matches

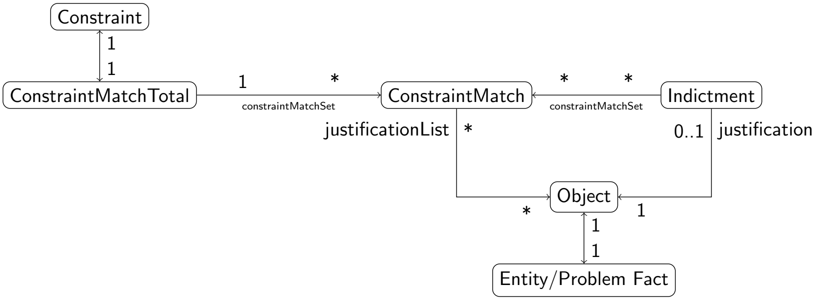
Additionally, for every entity/problem fact that triggered at least one constraint, an *Indictment* is created, mapping each entity/problem fact to the Constraint Matches it is involved in.



No Overlapping Shifts
Shift 1
Shift 3
Shift 2
Indictment
justification: Shift 1
constraintMatchSet: [ConstraintMatch 1]
score: -1 hard
ConstraintMatch 1
constraintId: "No Overlapping Shifts"
justificationList: [Shift 1, Shift 2]
score: -1 hard
ConstraintMatch 2
constraintId: "No Overlapping Shifts"
justificationList: [Shift 2, Shift 3]
score: -1 hard
⋮
Indictments for Shift 2 and Shift 3
⋮
ConstraintMatchTotal
constraintId: "No Overlapping Shifts"
constraintMatchSet: [ConstraintMatch 1, ConstraintMatch 2]
score: -2 hard

Constraint Matches

A summary of the relationship



Explaining the Score with OptaPlanner

You can get an explanation of the score by calling the *ScoreManager*'s *explainScore(solution)* method, which returns a *ScoreExplanation* that has:

- *getConstraintMatchTotalMap()*, which returns a map from Constraint Id to its *ConstraintMatchTotal*.
- *getIndictmentMap()*, which returns a Map from Planning Entities and Problem Facts to the constraint matches they are involved in.

When to use Constraint Match Totals

You should use Constraint Match Totals when you want a summary of constraint matches.

When to use Constraint Match Totals

How many PTO requests were we able to satisfy?

```
int satisfiedRequests =
   scoreManager.explainScore(roster)
               .getConstraintMatchTotalMap()
               .get("Day Off Requests")
               .getConstraintMatchCount();
```

When to use Constraint Match Totals

How much money is spent on fuel using this route? (assuming 1 soft score = 1 dollar)

```
HardSoftScore score = (HardSoftScore)
   (scoreManager.explainScore(route)
                .getConstraintMatchTotalMap()
                .get("Fuel Cost")
                .getScore());
int totalFuelCost = score.getSoft();
```

When to use Indictments

You should use Indictments when you want information about a particular entity:

When to use Indictments

How does the current assignment of the morning shift impact the score?

```
score = scoreManager.explainScore(roster)
                    .getIndictmentMap()
                    .get(morningShift)
                    .getScore();
```

When to use Indictments

Who desires
to join the OptaPlanner meeting but cannot attend due to a conflict?

```
// Get the indictment for the
// optaplannerMeeting
Collection<Attendee> attendees =
    scoreManager.explainScore(roster)
        .getIndictmentMap()
        .get(optaplannerMeeting)
        // ...
```

When to use Indictments

Who desires
to join the OptaPlanner meeting but cannot attend due to a conflict?

```
// ...
// Get optaplannerMeeting's
// Constraint Matches
.getConstraintMatchSet()
.stream()
// Only consider
// "Desired attendee cannot attend"
// Constraint Matches
.filter(cm -> cm.getConstraintName()
    .equals("Desired attendee
       cannot attend"))
```

When to use Indictments

Who desires
to join the OptaPlanner meeting but cannot attend due to a conflict?

```
// ...
// In a Constraint Stream like
// constraintFactory
//     .from(Talk.class)
//     .join(Attendee.class)
//     ...
// the Attendee is the second item
// in the justification list
.map(cm -> (Attendee)
   (cm.getJustificationList().get(1)))
.collect(Collectors.toList());
```

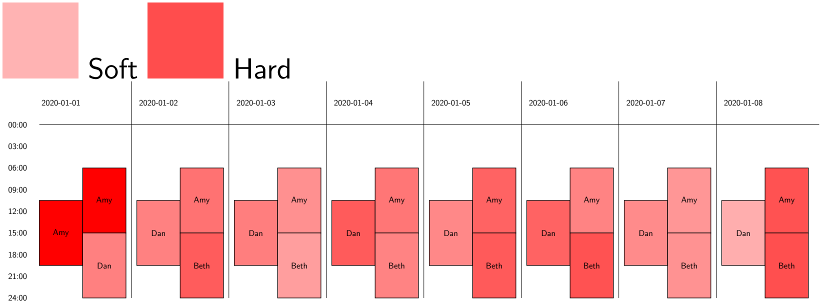
Outline

Notes

When visualizing the score, be wary of information overload.

- Make sure the things that important to the user stand out.
- Show less important details when the user inspect an entity (mouse over, tool tip, "what this", etc.).
- Hide details that are not important to the user.

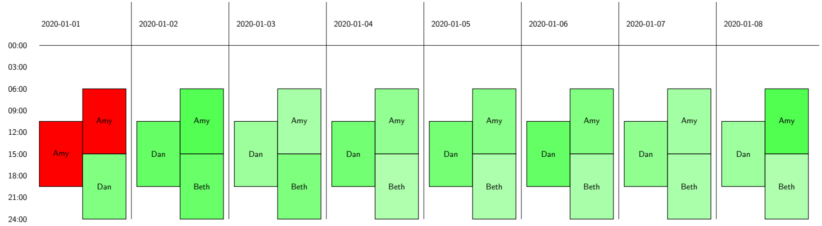
Bad Example



- Uses a single color to represent both soft and hard constraints
- Hard to identify the problem spots in the schedule

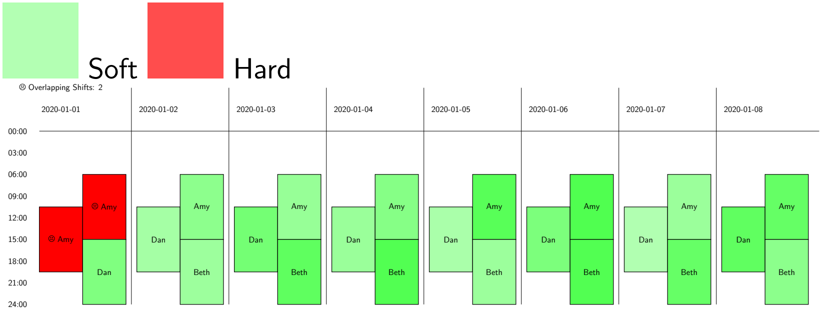
Better Example

🟩 Soft 🟥 Hard



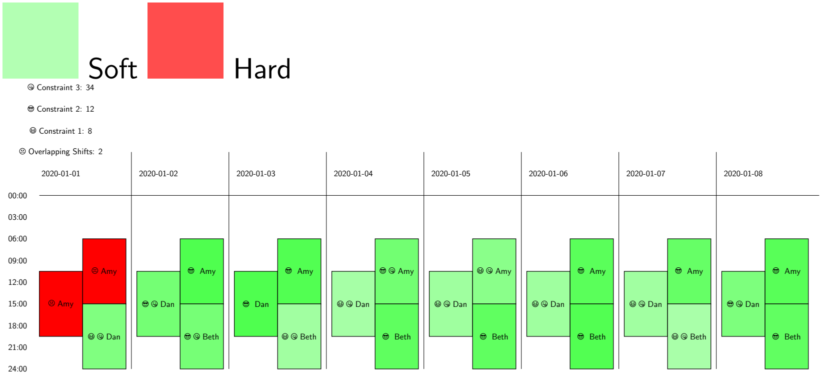
- Can clearly see what shifts violate hard constraints

Even Better Example



- Can clearly see what hard constraints are broken
- Summary of constraints broken at the top

What not to do



- Too many symbols can distract the user and make the screen cluttered
- Hard to tell what important

When to show more detail

- In general, you should hide details that are not important to the user, and show them when the user inspect an object.
- So instead of summarizing all your constraints, you show the top three that have the most impact, hiding the rest in a dialog. And instead of showing all the constraints an entity breaks, you show the top two, and show a detail dialog when the user interacts with the entity.

Outline

You can use *virtual* resources to represent missing or lacking resources. When a virtual resource is used, you penalize the score by how much it would cost to acquire said resource. This is useful for determining if there is any benefit to hiring an additional employee or buying additional vehicles.

Example - Domain

```java
public class Employee {
    String name;
    int cost;
    boolean isVirtual;

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

Example - Constraint

```
Constraint
    hireAdditionalEmployee(ConstraintFactory
    cf) {
     return cf.from(Employee.class)
     // Get Virtual Employees
        .filter(Employee::isVirtual)
     // That have at least one shift
        .ifExists(Shift.class,
            Joiners.equal(e -> e,
            Shift::getEmployee))
        .penalize("Hire additional employee",
            HardMediumSoftScore.ONE_MEDIUM,
            Employee::getCost);
}
```

Example - Calling the solver

```
// Get the problem
Roster roster = getRoster();
List<Employee> employeeList = new
   ArrayList<>(roster.getEmployeeList());
```

Example - Calling the solver

```
// Add Virtual Employees to the problem
employeeList.add(new Employee("New Nurse
   1", 2000, true));
employeeList.add(new Employee("New Doctor
   1", 5000, true));
roster.setEmployeeList(employeeList);
```

Example - Calling the solver

```
// Solve the problem
Roster solution = solverManager.solve(0L,
    roster).getFinalBestSolution();
```

Additional Notes

The score level the virtual resource constraint on is crucial. If it above all soft constraints, (i.e. a Medium Score level), virtual resources won't be used if it is possible to solve without them. If you want to consider solutions that include virtual resources even if it is possible to solve without them, put the virtual resource constraint on the soft score level.

If you use virtual resources, limit the amount you add – each virtual resource added slow down the solver.

Outline

Checking if a constraint is correct

Typically, *ConstraintVerifier* is used to create unit tests for your constraints. However, that only checks the correctness of your implementation, not that you implemented the constraint the business user wanted.

Checking if a constraint is correct

For instance, given the following description of a constraint:

Two Employees that do not like each other cannot work the same shift.

It could be implemented as...

Checking if a constraint is correct

```
Constraint
    employeeNotLikeOther(ConstraintFactory
    cf) {
     // Two shifts are the same if they
        share the same timeslot
     return cf.fromUniquePair(Shift.class,
            Joiners.equal(Shift::getTimeSlot))
         .filter((s1, s2) ->
            s1.getEmployee().getDislikeSet()
           .contains(e2))
         .penalize("Employees dislike each
            other", HardSoftScore.ONE_HARD);
}
```

Checking if a constraint is correct

Or...

```
Constraint
    employeeNotLikeOther(ConstraintFactory
    cf) {
     // Two shifts are the same if they
        share the same spot and timeslot
     return cf.fromUniquePair(Shift.class,
           Joiners.equal(Shift::getSpot),
           Joiners.equal(Shift::getTimeSlot))
        .filter((s1, s2) ->
           s1.getEmployee().getDislikeSet()
             .contains(e2))
        .penalize("Employees dislike each
           other", HardSoftScore.ONE_HARD);
}
```

Checking if a constraint is correct

The issue being, "the same shift" is ambiguous. Do they mean two shifts that share the same time slot, or two shifts that share the same timeslot slot and in the same spot? With score visualization, the business user can easily detect and report an issue for a particular constraint.

Outline

Summary

- Use *ScoreManager.explainScore(solution)* to get *ConstraintMatchTotals* and *Indictments* you can use to explain the score.
- When showing the explanation in the UI, hide out less important details and highlight more important ones.
- Use virtual resources to help identify deficient resources.
- You can use score visualization to help verify the constraint you implemented is what the user actually wanted.

Q & A

Thanks for Watching!
Questions?