Constraint Systems

DFS in Or-tools

Some Global Constraints in Or-Tools

There are two methods to customize search in or-tools

The first one consists in using search phases:

- A search phase is built with:

```
slv.Phase(variables, var_heuristic, value_heuristic)
```

- So far, we have seen only one option for the var. selection:

```
slv.INT_VAR_DEFAULT # Pick the first unbound variable
```

- ...And only one option for the value selection:

```
slv.INT_VALUE_DEFAULT # Assign min value
```

But there are many more possibilities!

Some Global Constraints in Or-Tools

For the variable selection, we have:

```
slv.CHOOSE_FIRST_UNBOUND
slv.CHOOSE_RANDOM
slv.CHOOSE_MIN_SIZE_LOWEST_MIN
slv.CHOOSE_MIN_SIZE_HIGHEST_MIN
slv.CHOOSE_MIN_SIZE_LOWEST_MAX
slv.CHOOSE_MIN_SIZE_HIGHEST_MAX
slv.CHOOSE_LOWEST_MIN
slv.CHOOSE_HIGHEST_MAX
slv.CHOOSE_MIN_SIZE
slv.CHOOSE_MAX_SIZE
slv.CHOOSE_MAX_REGRET_ON_MIN
```

- `MIN_SIZE` and `MAX_SIZE` refer to the domain size
- They break ties based on the order of variables

Some Global Constraints in Or-Tools

For the variable selection, we have:

```
slv.CHOOSE_FIRST_UNBOUND
slv.CHOOSE_RANDOM
slv.CHOOSE_MIN_SIZE_LOWEST_MIN
slv.CHOOSE_MIN_SIZE_HIGHEST_MIN
slv.CHOOSE_MIN_SIZE_LOWEST_MAX
slv.CHOOSE_MIN_SIZE_HIGHEST_MAX
slv.CHOOSE_LOWEST_MIN
slv.CHOOSE_HIGHEST_MAX
slv.CHOOSE_MIN_SIZE
slv.CHOOSE_MAX_SIZE
slv.CHOOSE_MAX_REGRET_ON_MIN
```

- The **`MIN_SIZE_...`** strategies break ties using different criteria
- The tie-breaking rule may sometimes be very important

Some Global Constraints in Or-Tools

For the variable selection, we have:

```
slv.CHOOSE_FIRST_UNBOUND
slv.CHOOSE_RANDOM
slv.CHOOSE_MIN_SIZE_LOWEST_MIN
slv.CHOOSE_MIN_SIZE_HIGHEST_MIN
slv.CHOOSE_MIN_SIZE_LOWEST_MAX
slv.CHOOSE_MIN_SIZE_HIGHEST_MAX
slv.CHOOSE_LOWEST_MIN
slv.CHOOSE_HIGHEST_MAX
slv.CHOOSE_MIN_SIZE
slv.CHOOSE_MAX_SIZE
slv.CHOOSE_MAX_REGRET_ON_MIN
```

- `MAX_REGRET_ON_MIN` picks the variable with the largest difference…
- …Between the min and the following value

Some Global Constraints in Or-Tools

For the value selection, we have:

```
ASSIGN_MIN_VALUE
ASSIGN_MAX_VALUE
ASSIGN_RANDOM_VALUE
ASSIGN_CENTER_VALUE
SPLIT_LOWER_HALF
SPLIT_UPPER_HALF
```

- The `SPLIT_...` strategies use the domain splitting scheme

Search phases on different variables can be combined:

```
db = slv.Compose([phase1, phase2, ...])
```

- `phase2` starts once all `phase1` vars are assigned, and so on

Some Global Constraints in Or-Tools

The second method consists in writing a custom DecisionBuilder

```
class Example(pywrapcp.PyDecisionBuilder):
    def __init__(self, vars):
        pywrapcp.PyDecisionBuilder.__init__(self)
        self.vars = vars

    def Next(self, slv):
        if [all vars are assigned]:
            return None
        else:
            decision = [build decision object]
            return decision
```

- Caveat: this method will often invoke a Python callback...
- ...Which is very slow!

Some Global Constraints in Or-Tools

A decision build should repeatedly return a decision object

There are several types of decision objects, including:

- Binary choice point ($x = v \vee x \neq v$)

```
slv.AssignVariableValue(var, value)
```

- Domain splitting ($x \leq v \vee x > v$)

```
slv.SplitVariableDomain(var, value, start_lower_half)
```

- Probing ($x = v$)

```
slv.AssignVariableValueOrFail(var, value)
```

- This is the only way to use probing from the Python wrapper

Constraint Systems

Lab 6 - Discrete Lot Sizing

Discrete Lot Sizing

Let's consider the following problem

Similarly to our production scheduling scenario:

- There are n product units to be produced
- Each unit belongs to a specific product type and has a deadline
- We can produce only one unit per time instant

Unlike in our production scheduling scenario:

- We pay a sequence-dependent transition cost...
- ...For switching from a product type to another...
- ...Even if there is a gap between the two time instants
- We pay a stocking cost for each time instant...
- ...Between the production of a unit and its deadline

Discrete Lot Sizing

We start from a given model:

The main constraints:

$$
\begin{aligned}
\min \; & z = \text{trans. cost} + \text{stocking cost} \\
\text{subject to: } & \textsc{alldifferent}(date) \\
& date_i \leq d_i & \forall i = 0..n \\
& \textsc{circuit}(succ) \\
& date_i < date_{succ_i} & \forall i = 0..n-1 \\
& date_n = n_{periods}
\end{aligned}
$$

- For each unit i we keep track of the production time $date_i$...
- ...And the next unit produced $succ_i$
- To ensure a complete chain, we use the CIRCUIT constraint...
- ...Which is yet another global constraint available in or-tools!

Discrete Lot Sizing

We start from a given model:

The main constraints:

$$
\begin{aligned}
\min \ z = \ & \text{trans. cost} \ + \ \text{stocking cost} \\
\text{subject to: } & \textsc{AllDifferent}(date) \\
& date_i \leq d_i & \forall i = 0..n \\
& \textsc{Circuit}(succ) \\
& date_i < date_{succ_i} & \forall i = 0..n-1 \\
& date_n = n_{periods}
\end{aligned}
$$

- Except that CIRCUIT enforces a cycle...
- ...And we have a path, instead
- Solution: add a fake product unit, scheduled at the very last time
- The fake unit has index n

Discrete Lot Sizing

We start from a given model:

The variable domains:

$$date_i \in \{0..n_{periods}\} \qquad \forall i = 0..n$$
$$succ_i \in \{0..n\} \qquad \forall i = 0..n$$

The cost expressions:

$$\text{trans. cost} = \sum_{i=0..n-1} T_{i,succ_i}$$
$$\text{stocking cost} = c_{stocking} \sum_{i=0..n-1} (d_i - date_i)$$

- $T_{i,j}$ is the transition cost from unit i to j
- The cost for switching to/from the fake unit is 0
- $c_{stocking}$ is the stocking cost

Discrete Lot Sizing

We start from a given model:

Some symmetry breaking constraints:

$$date_i < date_j \qquad \forall i, j = 0..n-1, i \neq j, p_i = p_j, d_i < d_j$$
$$succ_j \neq i \qquad \forall i, j = 0..n-1, i \neq j, p_i = p_j, d_i < d_j$$
$$succ_i \neq i \qquad \forall i = 0..n$$

Some general comments:

- This is not the best possible model for this problem
- In fact, it is not even very good
- But that's ok: the search strategy will be even more important

Discrete Lot Sizing

Objective: design a search strategy for the problem

- You can change the var/value section strategy in `Phase`
- You can reorder the problem entities
  - This affect all strategies based on the input order
- You can design a custom `DecisionBuilder`

Some comments:

- A `DecisionBuilder` written in Python is very slow
- This makes it difficult to have a fair comparison
- (Partial) Solution: use a branch limit instead of a fail limit
- Compare the performance in terms of number of branches

Discrete Lot Sizing

Objective: design a search strategy for the problem

- You can change the var/value section strategy in `Phase`
- You can reorder the problem entities
  - This affect all strategies based on the input order
- You can design a custom `DecisionBuilder`

Some comments:

- In most cases, you won't be able to prove optimality
- But you will have access to the best know solution from the literature
- Use it to compare the quality of the solution that you will get