Lab 5: Steel mill inventory matching

Workshop overview

In this workshop, you will practice using IBM® ILOG CP Optimizer to solve a steel mill inventory matching problem. We first present a description of the problem the production manager faces. Then you'll get a chance to derive the CP model from this description and model the problem using the OPL IDE. You'll also get a chance to experiment with search phases and alternative formulations to improve the solution speed.

Problem description

The steel mill has an inventory of steel slabs, of a finite number of different capacities (sizes), that are used to manufacture different types of steel coil. During production, some of the steel from the slabs is lost. The production manager has to decide which steel slabs to match with which coil orders in order to minimize the total loss. In optimization terms, the problem can be described as follows:

- **Objective**
  - Minimize the total loss
- **Decision variables**
  - Which slab should be matched with which coil order
  - The capacity of each slab used
- **Constraints**
  - A coil order can be built from at most one slab, although each slab can be used to fill several coil orders.
  - Each type of steel coil requires a specific production process, with each such process encoded by a *color* designated by a number between 1 and 88. A slab can be used for at most two different coil production processes or colors.
  - Each steel coil order has an associated weight, and the total weight of all coil orders matched with a slab must be less than the capacity of that slab.
  - The amount of loss from each slab equals the capacity of the slab minus the total weight of all coil orders assigned to that slab.
  - The total loss is the sum of losses from all slabs.
  - An unlimited quantity of steel slabs of each capacity is available.

Problem data

There are 111 coil orders and 21 different slab sizes, namely 12, 14, 17, 18, 19, 20, 23, 24, 25, 26, 27, 28, 29, 30, 32, 35, 39, 42, 43, and 44. The table below gives the **weight** and **color** data for 5 coil orders. The complete data set can be seen in the data file, which can be found in the exercise folder referred to below.

Coil order	Weight	Color
1	30	73
2	30	74
3	30	75
110	2	6
111	2	4

Even though an unlimited quantity of steel slabs is available, you can use an upper bound of 111 on the total number of steel slabs because of the obvious solution of using one slab per coil order.

Exercise folder

`<trainingDir>\OPL63.labs\SteelMill\work`

Step 1: Solve the problem using CP Optimizer and the OPL IDE

Actions

- Declare the data
- Declare the decision variables
- Define the objective function
- Define the constraints
- Solve the problem

Reference

constraint programming

Declare the data

1. In the OPL IDE, import the project by selecting **File >Import >Existing Projects into Workspace**, and choosing the **SteelMill_work** project from the exercise folder **`<trainingDir>\OPL63.labs\Steelmill\work`**. Leave the **Copy projects into workspace** box unchecked.
2. Expand the project to see the contents. For this step, you'll only work with the following:
   - Model file: **`steelmill_work.mod`**
   - Settings file: **`steelmill_work.ops`**
   - Data file: **`steelmill_work.dat`**
   - Run configuration: **`naive model (default)`**
3. Open the model file, **`steelmill_work.mod`**, and see that the following data has been declared:
   - The number of coil orders: **`int nbOrders = ...;`**
   - The weight of each coil order: **`int weight[1..nbOrders] = ...;`**

   Now use similar syntax to declare the following:

   - The integer number of steel slabs available: **`nbSlabs`**
   - The integer number of colors: **`nbColors`**
   - The integer number of distinct slab capacities: **`nbCap`**
   - An integer array for the actual capacities associated with each index in **`nbCap`**: **`capacities`**
   - An integer array for the colors associated with each coil order: **`colors`**
4. The remaining completed items in the data declaration part of the model file are used later in the model, and are as follows:
   - **`maxCap`**: This is the greatest available capacity and will be used to define the domain of the decision variables for slab capacity.
   - **`caps`**: This is the set of available capacities (with the same content as the array **`capacities`**) and is used in a later substep to define the capacity constraints for each slab.
5. Open the data file, **`steelmill_dat.dat`**, to see how the data is instantiated.

Declare the decision variables

1. In the model file, **`steelmill_work.mod`**, tell OPL that you're using CP by adding the line **`using CP;`** at the top.
2. In the section titled **`/* Decision variables */`**, the two decision variables have already been declared as follows:
   - **`dvar int where[1..nbOrders] in 1..nbSlabs`**: This variable is used to determine which slab each coil order is assigned to, and is therefore indexed over all orders with domain of all slab numbers.
   - **`dvar int capacity[1..nbSlabs] in 0..maxCap`**: This variable is used to determine the capacity of each slab, and is therefore indexed over all slab numbers, with domain of all values between 0 and the maximum available capacity.

3. Declare an integer decision expression called **load** indexed over the slabs. Assign the value of **load** to equal the sum of the weights of all coil orders assigned to a slab.

First write the expression to sum the weights of all coil orders. Next, use the **where** variable, together with the logical equals (**==**), to write an expression that evaluates to 1 if an order is assigned to a slab and 0 otherwise. Multiply this expression with the **weight** to add only the weight of orders assigned to the slab.

4. Declare another integer decision expression called **colorAssigned** indexed over the colors and the slabs. Complete the declaration with an expression to assign the value of **colorAssigned** to be 1 if any coil orders assigned to a given slab have a particular color and 0 otherwise.

First use the **where** variable and the logical equals (**==**) that you used for the **load** expression to determine whether an order is assigned to a slab (1) or not (0). Next, in the same expression, use the logical OR statement (**or**) to create an "or" over all assigned orders with the particular color. Be sure to check the syntax for the logical OR statement in the online documentation at **Language Quick Reference > OPL keywords**.

Define the objective function

1. In the section titled **Objective function**, write the objective to minimize total loss. The total loss is defined as the sum, over all slabs, of the slab capacity minus the slab load.

Define the constraints

1. In the section titled **Constraints** and within the **subject to** block, you'll see a **forall** constraint declaration that uses the **caps** set to restrict the domain of the **capacity** variable. Because the **capacity** variable has a domain enumerated from a set, its domain cannot be defined during data declaration as follows:

```
dvar int capacity[1..nbSlabs] in caps; // NOT ALLOWED
```

   Instead, a continuous domain has to be defined at declaration, and the domain then has to be restricted in the constraint block.
2. Add a constraint within this same **forall** block to state that the slab load must be less than or equal to the slab capacity.
3. Add a constraint called **colorCt** for each slab that states that the number of colors assigned to that slab must be less than or equal to 2.

Use the **colorAssigned** decision expression.

4. Your model is complete at this point. If you have any remaining errors, check the solution or check with your instructor before attempting to solve the model.

Solve the model

1. In the **OPL Projects Navigator**, expand the **Run Configurations** for your project. right click **naive model (default)** and select **Run this** from the context menu.
2. Select the **Engine log** output tab to see the solution progress.
3. Let the model run for about a minute and then click the red stop button (you can see the solution time scrolling by periodically on the left side of the log).
4. Scroll to the start of the log and notice that the first solution found (under the **Best** column), is around 1000. Scroll to the end of the log and notice that the best solution found after about a minute is around 30. In the next step you'll get a chance to implement a search phase to improve performance.

Step 2: Implement a search phase

Actions

- Add a search phase
- Solve the model

Reference

what is a search phase?

Add a search phase

In this exercise you'll write a search phase to guide CP Optimizer in the selection of decision variables during the solution search. For the steel mill problem, an intuitive search strategy is to first assign orders to slabs (first search on the **`where`** decision variable), and afterwards assign capacities to each slab (next search on the **`capacity`** decision variable).

If you're confident that your model is correct, you can continue working with it. Otherwise, you can continue with the **`searchPhase_work.mod`** file. These instructions assume you're using the latter file, which contains the model up to this point, together with some instructions on how to do implement the search phase.

1. Scroll down to the section titled **`Search phase`**. You'll write the search phase within the **`execute`** statement.
2. Define the script variable **`f`** to access the CP search modifier factory.
3. Define a search phase, **`phase1`**, on the **`where`** variable using the **`searchPhase`** method.
4. Define another search phase, **`phase2`**, on the capacity variable.
5. Use the **`setSearchPhase`** method to set the search to first use **`phase1`**, and next **`phase2`**.
6. If you have any errors, check the solution or check with your instructor before attempting to solve the model.

Solve the model

1. In the **OPL Projects Navigator**, expand the **Run Configurations** for your project. right click **search phase** and select **Run this** from the context menu.
2. Select the **Engine log** output tab to see the solution progress.
3. Let the model run for about a minute and then click the red stop button (you can see the solution time scrolling by periodically on the left side of the log).
4. Scroll to the start of the log and notice that the first solution found (under the **`Best`** column), is around 20 – much better than the first best solution of 1000 without the search phase. Scroll to the end of the log and notice that the best solution found after about a minute is around 8, again an improvement compared to the solution of 30 without a search phase. In the next step you'll get a chance to try and improve performance by changing the OPL model.

Step 3: Improve the model

Actions

- Improve the model
- Solve the model

Improve the model

The key to understanding the model improvements in this exercise, is realizing that once the load on a slab is known, its capacity becomes a trivial decision. Specifically, if the load is known, the capacity of that slab will simply be the smallest available capacity just bigger than the load on the slab. In this exercise, you'll take advantage of this knowledge to improve the model by removing the **capacity** decision variable.

1. In the same work project in the OPL IDE, open the **betterModel_work.mod** file.
2. In the **Data declaration** section of the model file, notice that a new line has been added:

   ```
   int loss[c in 0..maxCap] = min(i in 1..nbCap : capacities[i] >= c)
   capacities[i] - c;
   ```

   This line takes advantage of the fact that the load (and loss) is an integer and there are therefore a finite number of possible values the load on a slab can take, namely all integer values between **0** and **maxCap**. For each such value, the array above defines the loss to be the smallest capacity just larger than the load (defined by using the **min** function), minus the load. The array uses the index **c** and you'll see next how it can be used with the load values instead.

3. In the **Decision variables** section, remove the **capacity** decision variable. All the other variables remain the same.
4. Change your objective function to minimize the **loss** directly instead of using the **capacity** and **load** decision variables.

   

   Index the **loss** array with the **load** variable.

   

   If you're familiar with MP, you'll notice here one of the major differences between MP modeling and CP modeling: In CP a decision variable can be used to index an array, as is shown here where the **load** variable is used as an index for the **loss** array.

5. In the **Constraints** section, remove the constraints that use the **capacity** decision variable, seeing that they are no longer required when this variable doesn't exist.
6. Finally, in the **Search phase** section, remove **phase2** seeing that you no longer have the **capacity** variable.
7. If you have any errors, check the solution or check with your instructor before attempting to solve the model.

Solve the model

1. In the **OPL Projects Navigator**, expand the **Run Configurations** for your project. right click **Better Model** and select **Run this** from the context mneu.
2. Select the **Engine log** output tab to see the solution progress.
3. See that IBM ILOG CP Optimizer finds the optimal match between orders and slabs, resulting in zero loss, in about 0.1 seconds.

   

   The steel mill problem is a benchmark problem used to benchmark optimization engine performance, and it's worth noting that IBM ILOG CP Optimizer is the first constraint programming engine to be able to find an optimal solution to this problem.

Step 4: Use the pack constraint

Action

- Use the **pack** constraint

Reference

pack

Use the pack constraint

This part of the exercise is optional and shows you how to use one of IBM ILOG CP Optimizer's more advanced constructs, namely the **pack** constraint. This constraint is generally used to assign items into packs of finite capacity. In this sense, the orders are the items, and the slabs are the packs of finite capacity to which the items are assigned (see the **OPL Help** for further information on the **pack** constraint).

1. In the same project you've been working in, open the **polished_work.mod** file.
2. In the **Data declaration** section, see that a new property has been declared, namely **maxLoad**. This is the sum of the weights of all coil orders.
3. In the **Decision variables** section, see that **load** is no longer a decision expression, but is now an integer decision variable with domain between 0 and **maxLoad**. The reason for this change is that the **pack** constraint does not accept a decision expression as an argument and instead requires a decision variable.
4. In the **Constraints** section, look at the definition of the **pack** constraint and try to understand it by comparing it with the explanation in **OPL Help**.
5. Run the model from the **Polished Model Run Configuration** and see that it also finds the optimal solution of zero loss in around 0.1 seconds.