Lab 7: House Building Calendar

Problem Description

In this workshop you will model a problem of scheduling the tasks involved in building multiple houses in such a manner that minimizes the overall completion date of the houses.

Principles of the problem:

- There are five houses to be built.
- There are two workers:
  - Joe
  - Jim
- Each worker can only be assigned to a subset of the total list of tasks (different skills).
- Some tasks must take place before other tasks, and each task has a predefined size.
- Each worker has a calendar detailing the days on which he does not work:
  - Weekends
  - Public holidays
  - Other days off
- On a worker's day off, he does no work on his tasks.
- A worker's tasks may not be scheduled to start or end on a day off.
- Tasks that are in process by the worker are suspended during his days off (for example a task may begin on Friday and finish on Monday with a weekend off in between).

Problem data

Here is the list of house construction tasks, giving the length of each task, the worker assigned to it, and the precedence:

Task	Size	Worker	Preceding tasks
Masonry	35	Joe	–
Carpentry	15	Joe	masonry
Plumbing	40	Jim	masonry
Ceiling	15	Jim	masonr
Roofing	5	Joe	carpentry
Painting	10	Jim	ceiling
Windows	5	Jim	roofing
Façade	10	Joe	roofing plumbing
Garden	5	Joe	roofing plumbing
Moving	5	Jim	windows facade garden painting

Exercise folder

`<trainingDir>\OPL63.labs\Scheduling\Calendar\work\sched_calendarWork`

Step 1: Declare data and decision variables

Actions

- Define a calendar for each worker
- Declare the decision variable

Reference

intensity

Define a calendar for each worker

1. Import the **sched_calendarWork** project into the OPL Projects Navigator (Leave the **Copy projects into workspace** box unchecked) and open the **calendar.mod** and **calendar.dat** files for editing.
2. Examine the first data declarations and their instantiations in the **.dat** file:
   - The first two declarations, **NbHouses** and **range Houses** establish simple declarations of how many houses to build, and a range that is constrained between 1 and that total number.
   - The next two declarations instantiate sets of strings that represent, respectively, the names of the workers and the names of the tasks to perform.
   - The declaration **int Duration [t in TaskNames] = ...;** instantiates an array named **Duration** indexed over each **TaskNames** instance.
   - The declaration **string Worker [t in TaskNames] = ...;** instantiates an array named **Worker** indexed over each **TaskNames** instance.
   - The tuple set **Precedences** instantiates task pairings in the tuple **Precedence**, where each tuple instance indicates the temporal relationship between two tasks: the task in **before** must be completed before the task in **after** can begin.
   - The tuple **Break** indicates the start date, **s**, and end date, **e** of a given break period. A list of breaks for each worker is instantiated as the array **Breaks**. Each instance of this array is included in a set named **Break**.
3. Declare a tuple named **Step** with two elements:
   - An integer value, **v**, that represents the worker's availability at a given moment (0 for on a break, 100 for fully available to work, 50 for a half-day)
   - An integer value, **x**, that represents the date at which the availability changes to this value. Make this element the key for the tuple.

```
tuple Step {
int v;
key int x;
};
```

4. Create a sorted tuple set such that at each point in time where the worker's availability changes, an instance of the tuple set is created. Sort the tuple set by date, and use a **stepfuncion** named **calendar** to create the intensity values to be assigned to each **WorkerName**

Use a **stepwise** function

:

```
sorted {Step} Steps[w in WorkerNames] =
{ <100, b.s >| b in Breaks[w] } union
{ <0, b.e >| b in Breaks[w] };
stepFunction Calendar[w in WorkerNames] =
stepwise (s in Steps[w]) { s.v - >s.x; 100 };
```

When two consecutive steps of the function have the same value, these steps are merged so that the function is always represented with the minimal number of steps.

Declare the decision variable

1. Continue looking at the model file – the following **interval** decision variable is declared:

```
dvar interval itvs[h in Houses, t in TaskNames]
  size      Duration[t]
  intensity Calendar[Worker[t]];
```

2. Can you see how the **Calendar** function is associated with the decision variable in order to ensure that the worker's availability is taken into account?
3. Discuss this with your instructor and fellow students.

Step 2: Define the objective function

Action

- Transform the business objective into the objective function

Transform the business objective into the objective function

The business objective requires the model to minimize the total number of days required to build five houses. To do this in the model, you need to write an objective function that minimizes the maximum time needed to build each house and arrive at a minimum final completion date for the overall five-house project.

1. Determine the maximum completion date for each individual house project using the expression **`endOf`** on the last task in building each house (the **`moving`** task) and
2. Minimize the maximum of these expressions.

Check the solution in
`<trainingDir>\OPL63.labs\Scheduling\Calendar\solution\sched_calendarSolution\calendar.mod`

Step 3: Define constraints

Actions

- Write the precedence constraint
- Write the **noOverlap** constraint
- Write the forbidden start/end period constraint

References

noOverlap
forbidStart
forbidEnd

Write the precedence constraint

The precedence constraints in this problem are simple **endBeforeStart** constraints with no delay.

1. Write a single constraint that can be applied via the tuple set **Precedences** to each instance of the interval decision variable **itvs**.

   

   Use filtering on **(p in Precedences)** to separate out start dates and end dates. Use arrays of the form **[p.before]** and **[p.after]**.

2. Check your work against the file
   <trainingDir>\OPL63.labs\Scheduling\Calendar\solution\sched_calendarSolution\calendar.mod

Write the noOverlap constraint

1. Write a constraint that says the interval variables associated with a worker are constrained to not overlap in the solution.
2. Check your work against the file
   <trainingDir>\OPL63.labs\Scheduling\Calendar\solution\sched_calendarSolution\calendar.mod

You may be surprised by the form of the **noOverlap** constraint in the solution. This form is a shortcut that avoids the need to explicitly define the interval sequence variable when no additional constraints are required on the sequence variable.

Write the forbidden start/end period constraint

1. Write a constraint, using **forbidStart** and **forbidEnd**, that forbids a task to start or end on the associated worker's days off (i.e. when **intensity** = 0).
2. Check your work against the file
   <trainingDir>\OPL63.labs\Scheduling\Calendar\solution\sched_calendarSolution\calendar.mod