House building with worker skills

House building with worker skills

How can you schedule a series of tasks of varying durations where some tasks must finish before others start? And assign workers to each of the tasks such that each worker is assigned to only one task at any given time? And maximize the matching of worker skills to the tasks?

To illustrate scheduling of tasks in an optimal way, consider a house building problem in which five houses must be completed by a given date. The construction of each house includes a number of tasks such as installing a roof and painting. Each of these tasks requires a given duration of time from the start to completion of the task. Some tasks must necessarily take place before others; for example, the roofing must be complete before the windows can be installed. Moreover, there are three workers, and each task requires one of the three workers. The three workers have varying levels of skills with regard to the various tasks; if a worker has no skill for a particular task, he may not be assigned to the task. For some pairs of tasks, if a particular worker performs one of the pair on a house, then the same worker must be assigned to the other of the pair for that house. The objective is to find a solution that maximizes the task-associated skill levels of the workers assigned to the tasks.

This is an example of a scheduling problem in which tasks are represented using a special variable called an interval variable. An interval has a start time, an end time, and a duration. There are special constraints for intervals, including precedence constraints and no overlap constraints, which are used to model this problem.

The data (`sched_optional.dat`)

In this house building problem, there are five houses each with ten tasks of given durations. For each task, there is a list of tasks that must be completed before the task can start. All of the houses must be completed by a deadline of day 318.

Table 1. House construction tasks

Task	Duration	Preceding tasks
masonry	35	
carpentry	15	masonry
plumbing	40	masonry
ceiling	15	masonry
roofing	5	carpentry
painting	10	ceiling
windows	5	roofing
facade	10	roofing, plumbing
garden	5	roofing, plumbing
moving	5	windows, facade, garden, painting

For DropSolve, all data must be defined in the form of tables (which in OPL implies using sets of tuples to define these tables). A table `Planning` defines the number of houses to build and the deadline day.

```
Planning =
<  5, //number of houses to build
  318, // deadline
  >;
```

The durations of the tasks are provided in the table `Tasks` which contains a row (tuple) for each task.

```
Tasks =
{   < "masonry"   , 35>
  , < "carpentry" , 15>
  , < "plumbing"  , 40>
  , < "ceiling"   , 15>
  , < "roofing"   , 05>
  , < "painting"  , 10>
  , < "windows"   , 05>
  , < "facade"    , 10>
  , < "garden"    , 05>
  , < "moving"    , 05>
};
```

The precedences are represented by the set of tuples `Precedences`.

```
Precedences = {
  < "masonry",   "carpentry" >,
  < "masonry",   "plumbing"  >,
  < "masonry",   "ceiling"   >,
  < "carpentry", "roofing"   >,
  < "ceiling",   "painting"  >,
  < "roofing",   "windows"   >,
  < "roofing",   "facade"    >,
  < "plumbing",  "facade"    >,
  < "roofing",   "garden"    >,
  < "plumbing",  "garden"    >,
  < "windows",   "moving"    >,
  < "facade",    "moving"    >,
  < "garden",    "moving"    >,
  < "painting",  "moving"    >
 };
```

There are three workers with varying skill levels with regard to each of the ten tasks. If a worker has a skill level of zero for a task, he may not be assigned to that type of task.

Task	**Joe**	**Jack**	**Jim**
masonry	9	5	0
carpentry	7	0	5
plumbing	0	7	0
ceiling	5	8	0
roofing	6	7	0
painting	0	9	6
windows	8	0	5
facade	5	5	0
garden	5	5	9
moving	6	0	8

The workers are represented by the set `Workers`. The worker skills are represented by the set of tuples `Skills`.

```
Workers = { "Joe",
            "Jack",
            "Jim"
            };

Skills = {
 < "Joe",  "masonry",   9>,
 < "Joe",  "carpentry", 7>,
 < "Joe",  "ceiling",   5>,
 < "Joe",  "roofing",   6>,
 < "Joe",  "windows",   8>,
 < "Joe",  "facade",    5>,
 < "Joe",  "garden",    5>,
 < "Joe",  "moving",    6>,
 < "Jack", "masonry",   5>,
 < "Jack", "plumbing",  7>,
 < "Jack", "ceiling",   8>,
 < "Jack", "roofing",   7>,
 < "Jack", "painting",  9>,
 < "Jack", "facade",    5>,
 < "Jack", "garden",    5>,
 < "Jim",  "carpentry", 5>,
 < "Jim",  "painting",  6>,
 < "Jim",  "windows",   5>,
 < "Jim",  "garden",    9>,
 < "Jim",  "moving",    8>
};
```

There are four continuity requirements in this problem. If Jack performs the roofing task on a house, he must also perform the façade task on that house. Similarly, if he performs the façade task, he must perform the roofing task on the house. If Jim performs one of garden or moving on a house, then he must perform the other task on that house. If Joe performs one of masonry or carpentry on a house, then he must perform the other task on that house. Also, if Joe performs one of carpentry or roofing on a house, then he must perform the other task on that house.

The continuity requirements are represented by the set of tuples `Continuities`.

```
Continuities = {
  < "Joe",  "masonry",   "carpentry" >,
  < "Jack", "roofing",   "facade"    >,
  < "Joe",  "carpentry", "roofing"   >,
  < "Jim",  "garden",    "moving"    >
};
```

Solving the problem consists of determining starting dates for all the tasks and assigning a worker to each task such that sum of the skill levels used for the tasks is maximized.

The model (`sched_optional.mod`)

In order to model this problem, the unknowns, constraints, and objective must be determined. The unknowns are the dates that the tasks will start and which worker will be assigned to each task. In this model, there are constraints to specify that a particular task may not begin until one or more given tasks have been completed. There are additional constraints stating that each task must be assigned one worker who has a nonnegative skill level for the task. No worker may be

assigned to more than one task at a given time, and the assignments must adhere to the continuity requirements. The objective is to maximize the skill levels of the workers assigned to the tasks.

Modeling the variables

The unknowns are represented by two matrices of interval variables in this model. The first, `tasks`, is indexed on the houses and tasks and must be scheduled in the interval `[0..Planning.deadline]`. The other matrix of interval variables is indexed on the houses and the `Skills` tupleset. These interval variables are optional as they may or may not be present in the solution. The intervals that are present in the solution represent which worker performs which task.

```
dvar interval tasks [h in Houses][t in Tasks] in 0..Planning.deadline size t.duration;
dvar interval wtasks[h in Houses][s in Skills] optional;
```

After the problem is solved, the values assigned to these interval variables will represent the solution to the problem.

Modeling the constraints

The constraints appear in the `subject to` section of the model using the `endBeforeStart` constraint construct. The tasks in the model have precedence constraints that are added to the model using the `endBeforeStart` constraint that specifies that the first task must end before the second.

```
  forall(h in Houses) {
    // Temporal constraints
    forall(p in TaskPrecedences)
      endBeforeStart(tasks[h][p.beforeTask], tasks[h][p.afterTask]);
```

To constrain the solution so that exactly one of the interval variables `wtasks` associated with a given task of a given house is to be present in the solution, you use the specialized constraint `alternative`, which specifies only one of the interval variables can be present in the solution.

```
 forall(h in Houses) {
    // Alternative workers
    forall(t in Tasks)
      alternative(tasks[h][t], all(s in Skills: s.task==t) wtasks[h][s]);
  }
```

The expression `presenceOf` is used to represent whether a task is performed by a worker. The expression `presenceOf` is true if the interval variable is present in the solution; it is false if the interval variable is absent from the solution. For each house and each given pair of tasks and worker that must have continuity, you constrain that if the interval variable for one of the two tasks for the worker is present, the interval variable associated with that worker and the other task must also be present.

```
forall(h in Houses) {
    // Continuity constraints
    forall(c in Continuities,
           <c.worker, c.task1, l1> in Skills,
           <c.worker, c.task2, l2> in Skills)
      presenceOf(wtasks[h,<c.worker, c.task1, l1>]) ==
      presenceOf(wtasks[h,<c.worker, c.task2, l2>]);
  }
```

The `noOverlap` constraint is used to enforce the constraints that a given worker can be assigned only one task at a given moment in time.

```
forall(w in Workers)
    noOverlap(all(h in Houses, s in Skills: s.worker==w) wtasks[h][s]);
```

Modeling the objective

The objective of this problem is to maximize the skill levels used for all the tasks. The presence of an interval variable in the interval variable array `wtasks` in the solution must be accounted for in the objective. For each of these possible tasks, the cost is incremented by the product of the skill level and the expression representing the presence of the interval variable in the solution.

```
maximize sum(h in Houses, s in Skills) s.level * presenceOf(wtasks[h][s]);
```

The solution

The search for an optimal solution in this problem could potentiality take a long time, so a fail limit has been placed on the solve process. The search will stop when the fail limit is reached, even if optimality of the current best solution is not guaranteed.

```
execute {
  cp.param.FailLimit = 10000;
}
```

In this model, the optimization engine assigns a start time to each of the `tasks` intervals. In addition, each of the worker assignment interval variables, `wtasks`, is set to present or absent.

The best objective found within the limit for this problem has a value of 357.

Possible customizations

To modify this example, edit the work skill levels for each task.