Flexible job-shop scheduling

Flexible job-shop scheduling

How can a manufacturing company complete the processing of orders in minimal time when each order requires the use of a specific machine or a choice of specific machines in a specified sequence?

The Flexible Job-shop Scheduling Problem is as extension of the classical Job-shop Scheduling Problem. In the classical version of the job-shop scheduling problem, a finite set of jobs is processed on a finite set of machines. Each job is characterized by a fixed order of operations, each of which is to be processed on a specific machine for a specified duration. Each machine can process at most one operation at a time. Once an operation initiates processing on a particular machine, the operation must complete processing uninterrupted.

The flexible job-shop scheduling problem allows an operation to be processed by any machine from a specified set. The operation processing time might depend on the allocated machine. The problem is to assign each operation to a machine and to order the operations on the machines such that the maximal completion time (makespan) of all operations is minimized.

This problem is an example of a scheduling problem in which operations are represented by special variables that are called interval variables and interval sequence variables. An interval has a start time, an end time, and a duration. An interval sequence variable (also called a sequence variable) represents a total ordering of the interval variables of the set. There are special constraints for intervals and sequences, including precedence constraints, no overlap constraints, and alternative constraints, which are used to model this problem.

The data

The instance of flexible job-shop scheduling problem under consideration is the instance `edata/abz5`[1]. This instance was solved to optimality for the first time in November 2013 using CP Optimizer.

The problem consists of 10 jobs with 10 operations per job and 10 machines. Each operation can be performed on a specific machine or on one of two candidate machines.

The first job is described here:

```
Job 1: {(5 88)}, {(9 68)}, {(7 94)}, {(6 99)}, {(2 67),(8 67)},
       {(3 89)}, {(10 77)}, {(8 99)}, {(1 86)}, {(4 92)}
```

The first operation must be performed on Machine 5 with a duration of 88, the second operation on Machine 9 with duration 68, and so on. The fifth operation must be performed on either Machine 2 or Machine 8 with a duration of 67.

1. J. Hurink, B. Jurisch, and M. Thole. Tabu Search for the Job-Shop Scheduling Problem with Multi-Purpose Machines. OR Spektrum, 15(4):205–215, 1994.

The model

To model this problem, the unknowns, constraints, and objective must be determined. The unknowns are the times that the operations will start/end and on which machine each operation will be performed. In this model, there are constraints on the order of the operations for each job. There are also constraints which state that each operation must be assigned to exactly one machine. There is also a set of constraints to represent that a machine can process only one operation at a time. The objective is to minimize the end (completion) time of the final task.

Modeling the variables

The model uses a set of interval variables to represent the operations. For instance, the first operation of the first job is represented as an interval variable `Op_1_1`.

```
Op_1_1 = intervalVar();
...
Op_1_5 = intervalVar();
...
```

Possible allocation of operation to machines is represented by a set of optional interval variables. These interval variables are optional as they might not be present in the solution. For instance, there is an optional interval variable that represents the possible allocation of `Op_1_1` to machine `M5`:

```
Op_1_1_M5 = intervalVar(optional, size=88);
```

Two optional interval variables represent the possible allocation of `Op_1_5` to either machine `M2` or `M8` :

```
Op_1_5_M2 = intervalVar(optional, size=67);
Op_1_5_M8 = intervalVar(optional, size=67);
```

The sizes of these optional interval variables represent the size of the interval variable (duration of the operation) if the optional interval variable is present (that is, if the operation is allocated to this machine).

Each machine is represented by a sequence variable that is defined on the set of optional interval variables that use the machine. For instance, sequence variable `M8` is defined as:

```
M8 = sequenceVar([Op_1_5_M8, Op_1_8_M8, Op_2_9_M8, Op_3_7_M8, Op_4_1_M8, Op_5_9_M8,
                  Op_6_1_M8, Op_6_7_M8, Op_7_1_M8, Op_8_7_M8, Op_9_4_M8, Op_10_5_M8]);
```

Modeling the constraints

The flexible job-shop scheduling problem involves three types of constraints: precedence constraints between operations, alternative constraints for machine allocation, and no-overlap constraints on machines.

Precedence constraints between consecutive operations of a job are expressed with `endBeforeStart`. For instance:

```
endBeforeStart(Op_1_1, Op_1_2);
```

states that `Op_1_1` must end before `Op_1_2` starts.

Alternative constraints are used to represent the machine allocation. For instance, for operation `Op_1_5` either machine `M2` or machine `M8` can be assigned:

```
alternative(Op_1_5, [Op_1_5_M2, Op_1_5_M8]);
```

This constraint states that exactly one interval variable among `Op_1_5_M2` or `Op_1_5_M8` must be selected by the engine to be present (`Op_1_5_M2` is present means operation `Op_1_5` is allocated to machine `M2`, `Op_1_5_M8` is present means operation `Op_1_5` is allocated to machine `M8`) and the selected interval variable must start and end at the same time as `Op_1_5`.

In the case that there is just one candidate machine, such as for operation `Op_1_1`, the array of possible optional interval variables has only one element:

```
alternative(Op_1_1, [Op_1_1_M5]);
```

This constraint states that `Op_1_1_M5` is necessarily present and starts and ends at the same time as `Op_1_1`.

Because a particular machine can perform only one operation at a time, the interval variables that represent the operations that use the machine cannot overlap. This restriction is expressed with `noOverlap` constraints on the sequence variables, such as:

```
noOverlap(M8);
```

Modeling the objective function

The objective of this problem is to minimize the end time (makespan) of the schedule. The makespan is defined as the end time of the last operation of the schedule. In other words, the goal is to minimize the maximum of the end times of the intervals that represent the final operations of each of the jobs:

```
minimize(max([endOf(Op_1_10), endOf(Op_2_10), ..., endOf(Op_10_10)]));
```

The solution

The optimal solution to the problem has a makespan value of 1167.