Édition 2022

KI024
D. Bienaimé, C. Gouvernet, F. Medina
L. Baty, L. Bouvier and A. Parmentier

1 Context

Pelico provides a factory operations management system to give teams the agility to manage daily volatility and deliver products on time, at cost. The platform has been designed to serve all teams involved in plant operations:

- Production planning
- Supply Chain
- Maintenance Repair and Operations
- Material management
- Customer Support
- Factory management

The Short Term Production Planning (STTP) allows to carry out the production planning at the granularity of the day.

Several times a day, production leaders ask themself: who should do this operation, when, and with which ressources? What are the priorities ? What is critical ? Theses questions arise during the planning of the next day and week but also at any unexpected events: a part did not arrive, a machine is no longer available, a task takes longer than initially planned, a customer asks to bring forward an order...

Answering them in the best way possible is the first critical step to ensure on time delivery for their customers. The second one is the collaboration between the teams, which is promoted by giving a holistic view of the situation to every stakeholder.

2 Problem description

We consider a scheduling problem in a plant. The plant has a set $\mathcal{M} = \{1, \ldots, M\}$ of machines, and a set $\mathcal{O} = \{1, \ldots, O\}$ of operators. A set of jobs $\mathcal{J} = \{1, \ldots, J\}$ must be performed in a plant. The weight $w_j > 0$ provides the importance of job j. Each job j in J consists in a sequence $S_j = (i_1, \ldots, i_{k_j})$ of tasks i. Tasks are not shared between jobs: Each job has its tasks. We denote by $\mathcal{I} = \{1, \ldots, I\}$ the complete set of tasks.

$$\mathcal{I} = \bigsqcup_{j \in \mathcal{J}} S_j.$$

Each task i of each job has to be performed on a single machine. A single operator performs the task on the machine. We denote by $p_i \in \mathbb{Z}_+$ the processing time of task i: It is the time needed to operate i on the machines. Preemption is not allowed: Once a task has been started, it must be completed. We denote by r_j the release date of job j.

Recall that $S_j = (i_1, \ldots, i_{k_j})$ is the sequence of tasks in j. We must decide at which time $B_i \in \mathbb{Z}_+$ we start each task i. Let $C_i \in \mathbb{Z}_+$ be the completion time of task i. And let B_j and C_j be the times at which job j is started and completed, respectively.

$$C_i = B_i + p_i \tag{1}$$
$$B_j = B_{i_1} \tag{2}$$
$$C_j = C_{i_{k_j}} \tag{3}$$

The first task i_1 of j cannot be started before r_j. Any task i_h with $h > 1$ cannot be started before i_{h-1} is completed.

$$B_{i_1} = B_j \geq r_j \tag{4}$$
$$B_{i_h} \geq C_{i_{h-1}} \quad \text{for } h > 1 \tag{5}$$

We also denote by d_j the due date of job j. It is the time at which job j should be finished. We denote by T_j the tardiness of job j, and U_j the unit penalty for job j.

$$T_j = \max(C_j - d_j, 0) \quad \text{and} \quad U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

We want to finish jobs early, and therefore have a cost

$$\sum_{j \in \mathcal{J}} w_j (C_j + \alpha U_j + \beta T_j).$$

In practice, we typically have $1 < \beta < \alpha$. We denote by $\mathcal{M}_i \subseteq M$ the machines on which task i can be performed. Performing a task on a machine requires a single operator, but this operator must have some specific skills. We denote by $\mathcal{O}_{im}$ the set of operators that can operate machine m to perform task i.

We must choose the machine $m_i \in \mathcal{M}_i$ that performs task i, and the operator $o_i \in \mathcal{O}_{im}$ that operates m on task i. Two tasks cannot be processed on the machine at the same time.

$$B_{i'} \notin \{B_i, \dots, B_i + p_i - 1\} \quad \text{for all } i, i' \in \mathcal{I},\ i' \neq i \text{ such that } m_{i'} = m_i \text{ or } o_{i'} = o_i \tag{7}$$

In summary, a solution can be encoded by the vector $(B_i, m_i, o_i)_{i \in \mathcal{I}}$. The goal of this Hackathon is to find an optimal solution of the following optimization problem.

$$\begin{aligned} \min \quad & \sum_{j \in \mathcal{J}} w_j (C_j + \alpha U_j + \beta T_j) \\ \text{subject to} \quad & \text{constraints (1)-(7)} \\ & B_i \in \mathbb{Z}_+,\ m_i \in \mathcal{M}_i, o_i \in \mathcal{O}_{i,m_i} \quad \text{for all } \in \mathcal{I} \end{aligned} \tag{8}$$

3 Instance format and solutions

Instances are given under the `json` format, which basically contains embedded dictionaries. In these dictionaries, the keys are always strings within quotation marks. Here is an example of a `tiny.json`.

Listing 1: `tiny.json`

```
{
  "parameters": {
    "size": {
      "nb_jobs": 2,
      "nb_tasks": 3,
      "nb_machines": 2,
      "nb_operators": 2
    },
    "costs": {
      "unit_penalty": 20,
      "tardiness": 2
    }
  },
  "jobs": [
    {
      "job": 1,
      "sequence": [1,2],
      "release_date": 0,
      "due_date": 15,
      "weight": 3
    },
    {
      "job": 2,
      "sequence": [3],
      "release_date": 5,
      "due_date": 12,
      "weight": 2
    }
  ],
  "tasks": [
    {
      "task": 1,
      "processing_time": 8,
```

Listing 2: `tiny.json` (continuation)

```
      "machines": [
        {
          "machine": 1,
          "operators": [1,2]
        },
        {
          "machine": 2,
          "operators": [1]
        }
      ]
    },
    {
      "task": 2,
      "processing_time": 6,
      "machines": [
        {
          "machine": 1,
          "operators": [1]
        }
      ]
    },
    {
      "task": 3,
      "processing_time": 5,
      "machines": [
        {
          "machine": 1,
          "operators": [1]
        }
      ]
    }
  ]
}
```

Let us now briefly describe its syntax. We start with the dictionary attributes that contain other containers.

- Attribute `parameters` contains a dictionary with the main parameters of the instance.
- Attribute `jobs` contains an array with the jobs in $\mathcal{J}$, each job being described as a dictionary.
- Attribute `tasks` contains an array with the tasks in $\mathcal{I}$, each task being described as a dictionary.
- Attribute `sequence` within a job dictionary contains an array with the sequence of tasks S_j of job j.
- Attribute `machines` within a task dictionary contains an array with the machines in $\mathcal{M}_i$ that can operate i
- Attribute `operators` within a machine m dictionary, itself in a task i dictionary contains the operators in $\mathcal{O}_{im}$ that can operator m on i.

Table 1 describes all the other attributes in these dictionnaries.

Symbol	`json` **key**	**Meaning**
	Instance parameters	
J	`nb_jobs`	number of jobs
I	`nb_tasks`	number of tasks
M	`nb_machines`	number of machines
O	`nb_operators`	number of operator
α	`unit_penalty`	unit penalty cost
β	`tardiness`	tardiness cost
	Job j parameters	
j	`job`	job id
S_j	`sequence`	tasks sequence
r_j	`release_date`	release date
d_j	`due_date`	due date
w_j	`weight`	job weight
	Task i parameters	
i	`task`	task id
p_i	`processing_time`	processing time
$\mathcal{M}_i$	`machines`	machines that can do i
	Machine m in $\mathcal{M}_i$ parameters	
m	`machine`	machine id
O_{im}	`operators`	Operators that can do i on m

Table 1: Keys in instances `json`

The solutions you should return are also `json` files. These solution files should contain an array, each element of the array being a dictionary and corresponding to a task. The dictionary of a task i has the following attributes.

- Attribute `task` contain the id i of task i.
- Attirbute `start` contains the begin time B_i of task i.
- Attribute `machine` contains the id of the machine $m_i \in \mathcal{M}_i$ on which task i is operated.
- Attribute `operators` contains the operator $o_i \in \mathcal{O}_{im}$ which operates tasks i on machine m.

Here are two example of feasible solutions. Their respective costs are provided in Table 2.

Listing 3: `tiny-sol1.json`

```
[
  {
    "task": 1,
    "start": 0,
    "machine": 1,
    "operator": 2
  },
  {
    "task": 2,
    "start": 13,
    "machine": 1,
    "operator": 1
  },
  {
    "task": 3,
    "start": 8,
    "machine": 1,
    "operator": 1
  }
]
```

Listing 4: `tiny-sol2.json`

```
[
  {
    "task": 1,
    "start": 0,
    "machine": 2,
    "operator": 1
  },
  {
    "task": 2,
    "start": 8,
    "machine": 1,
    "operator": 1
  },
  {
    "task": 3,
    "start": 14,
    "machine": 1,
    "operator": 1
  }
]
```

`tiny-solution1.json`					
Task i	C_i	Job j	C_j	U_j	T_j
1	8	1	19	1	4
2	19	2	13	1	1
3	13				
Total Cost: $3(19+20+2\times 4)+2(13+20+2)=211$					

`tiny-solution2.json`					
Task i	C_i	Job j	C_j	U_j	T_j
1	8	1	14	0	0
2	14	2	19	1	7
3	19				
Total Cost: $3\times 14+2(19+20+2\times 7)=148$					

Table 2: Solutions costs decomposition

4 How the ranking will be established

Four instances are provided:

- small.json
- medium.json
- large.json
- huge.json

The score of a team is the sum of the costs of the proposed solutions for each instance, without normalization (in other words, the large instances will weigh more and this is normal). The team with the best score, i.e. the lowest score, wins.

5 Some tips

You have to be very efficient to address such a problem in six hours. The team that is going to win is the one that manages to quickly produce decent solutions.

1. Divide the tasks
2. Immediately start coding the tools to parse an input file and create an output file
3. Keep it simple: it's not hard to build a feasible solution. Start by coding simple methods that give you pretty good solutions and avoid designing a very powerful algorithm that you will not be able to implement in 6 hours.

References

[1] "Pelico's website", Pelico; 2022
https://www.pelico.ai/