

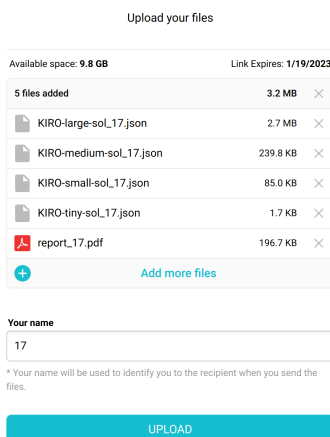
Operations research project

Projects should be done by groups of three students, and submitted on the following pcloud link (<https://e.pcloud.com/#page=puplink&code=aCMZ8WXDJPwmc5BC0hYwq1YohjkMI31V>) before January 18th, 23:59. For each group, please upload the following five files in a single upload on the link above.






- A report with the pdf format named `rapport_GROUP.pdf`. This report must :
 - contain the group number and the name of all the students of the group,
 - be at most 4 pages with a font size of at least 11 point (*-2 points by additional page*)
- The files `KIRO-tiny-sol_GROUP.json`, `KIRO-small-sol_GROUP.json`, `KIRO-medium-sol_GROUP.json`, `KIRO-large-sol_GROUP.json` containing the solution of the instances provided.

In file names, `GROUP` should be replaced by your group number. In the form on the pcloud link, please put your group number in the field “Your name”.

For instance, if you are in group 17, your submission on pcloud should look like this.



The screenshot shows the pcloud upload interface. At the top, it says "Upload your files". Below that, it shows "Available space: 9.8 GB" and "Link Expires: 1/19/2023". A table lists the files to be uploaded:

5 files added		3.2 MB	×
	KIRO-large-sol_17.json	2.7 MB	×
	KIRO-medium-sol_17.json	239.8 KB	×
	KIRO-small-sol_17.json	85.0 KB	×
	KIRO-tiny-sol_17.json	1.7 KB	×
	report_17.pdf	196.7 KB	×

Below the table, there is a blue plus icon and a link "Add more files". Underneath, there is a field labeled "Your name" with the value "17" entered. A small note below the field says: "* Your name will be used to identify you to the recipient when you send the files." At the bottom, there is a large blue button labeled "UPLOAD".

Please respect the nomenclature, the `json` and `pdf` formats, and the single upload per group (*-1 point for each rule not respected*).

1 Context

2 Problem description

We consider a scheduling problem in a plant. The plant has a set $\mathcal{M} = \{1, \dots, M\}$ of machines, and a set $\mathcal{O} = \{1, \dots, O\}$ of operators. A set of jobs $\mathcal{J} = \{1, \dots, J\}$ 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 $S_j = (i_1, \dots, i_{k_j})$ of tasks i . Tasks are not shared between jobs: Each job has its tasks. We denote by $\mathcal{I} = \{1, \dots, I\}$ 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 $S_j = (i_1, \dots, i_{k_j})$ 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.

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

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.

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

Table 1: Keys in instances `json`

- 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 dictionaries.

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 .
- Attribute `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.

tiny-solution1.json						tiny-solution2.json					
Task i	C_i	Job j	C_j	U_j	T_j	Task i	C_i	Job j	C_j	U_j	T_j
1	8	1	19	1	4	1	8	1	14	0	0
2	19	2	13	1	1	2	14	2	19	1	7
3	13					3	19				
Total Cost:						Total Cost:					
$3(19 + 20 + 2 \times 4) + 2(13 + 20 + 2 \times 1) = 211$						$3 \times 14 + 2(19 + 20 + 2 \times 7) = 148$					

Table 2: Solutions costs decomposition

- 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
  }
]
```

4 Instructions

4.1 Questions (6 points)

1. Give an MILP formulation modeling the problem.
2. Is there symmetries in your formulation ? Can you break them ?

3. Solve the instance `KIR0-small.json` with an MILP solver (See Section 5). Give an optimal solution / the best optimality gap you could get.

4.2 Algorithm (14 points)

Using a strategy of your choice, propose a solution for the four instances provided.

- You will turn in a file in the desired format giving your solution for the instance provided.
- The quality of the resolution strategies used and their presentation in the report will be scored out of 8 points.
 - Quality of the approach provided (*4 points*).
 - * Prove any interesting results on these approaches (accuracy, complexity, etc.).
 - Rigor and quality of writing will be given special consideration. (*4 points*)
 - * The algorithms must be provided in a readable way (<https://en.wikipedia.org/wiki/Pseudocode>)
 - * For the numerical results, provide the costs of the solutions, their optimality / gap.
 - * The numerical results must be presented in a readable way (table, captioned figure).
- The quality of the solutions provided will be noted on 6 points.
 - The score of a team is the sum of the cost of the solutions returned for each instance (the large instances therefore have a much stronger weight). The teams will be ranked by score (the team with the lowest score will have the best score). On the 6 points, 4 will be linked to the ranking:
 - * 4 points + 2 bonus points for the first group
 - * 4 points + 1 bonus point for the second
 - * 4 points for the third group
 - * 3 points for those who are in the first quarter
 - * 2 points for those who are in the second quarter
 - * 1 point for those who are in the third quarter
 - * 0 for the others
 - Your solutions will be verified with the verification code provided to you. Please email me at axel.parmentier@enpc.fr if you identify an error in the subject or in the verification code.

The problem is difficult to solve.

- **Don't start the project at the last minute.** Ask the teachers in your groups questions.
- Feel free to simplify / decompose it to get feasible solutions.
- If you wish, you can use solvers to solve the problem. However, since the instance is large, it is unlikely that a solver can find a good solution if applied to the whole instance. This does not prevent you from using a solver for a sub-problem.

5 Resources

5.1 Parser and solution evaluator in julia

Léo Baty and Louis Bouvier have made available parsers in `julia` and `python` and a solution evaluator in `julia`: <https://github.com/BatyLeo/KIRO2022.jl>.

5.2 Language

You should use the language you are the most comfortable with.

- See e.g. <https://gdalle.github.io/phd-resources/tutorials/python/> for more resources mathematics with `python`.
- See e.g. <https://gdalle.github.io/IntroJulia/> for more resources on mathematics with `julia`.

5.3 MILP solvers

You can for instance use the open-source MILP solvers `HiGHS` or `SCIP`. If you wish, you can also use a commercial solver such as `Gurobi` with a free academic license. See e.g.

- <https://www.cvxpy.org/tutorial/advanced/index.html#mixed-integer-programs> on how to use a MILP solver in `python`.
- <https://jump.dev/JuMP.jl/stable/> on how to use a MILP solver in `julia`.