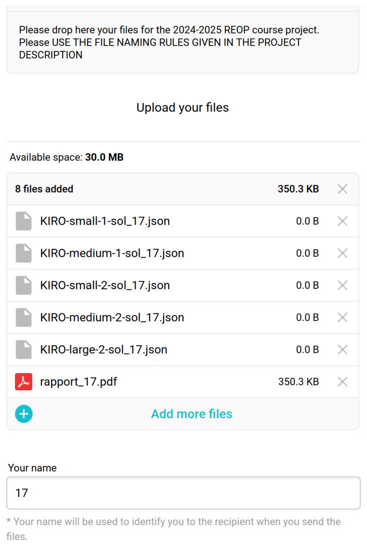
Operations research project

Projects should be done by groups of three students, and submitted on the following pcloud link (https://e.pcloud.com/#page=puplink&code=BEp7ZGrRSrR9N3KJKQWGsoRLdszDtQaOV) before January 21st, 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-1-sol_GROUP.json`, `KIRO-small-2-sol_GROUP.json`, `KIRO-medium-1-sol_GROUP.json`, `KIRO-medium-2-sol_GROUP.json`, `KIRO-large-1-sol_GROUP.json`, `KIRO-large-2-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.


Please drop here your files for the 2024-2025 REOP course project.
Please USE THE FILE NAMING RULES GIVEN IN THE PROJECT DESCRIPTION

Upload your files

Available space: **30.0 MB**

8 files added	**350.3 KB**
KIRO-small-1-sol_17.json	0.0 B
KIRO-medium-1-sol_17.json	0.0 B
KIRO-small-2-sol_17.json	0.0 B
KIRO-medium-2-sol_17.json	0.0 B
KIRO-large-2-sol_17.json	0.0 B
rapport_17.pdf	350.3 KB

Your name
17

* Your name will be used to identify you to the recipient when you send the files.

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

1 Context and Needs

The Renault Group, composed of four brands (Renault, Dacia, Alpine, and Mobilize), is a major player in the global automotive industry. The group produces more than 2 million vehicles each year, sold in over 130 countries. Its supply chain is highly complex: more than 6,000 suppliers supply the group's 26 factories daily, enabling the production of over 10,000 vehicles. The current project focuses on the daily sequencing of production in the factories, with up to 700 vehicles to be scheduled per day and per factory. This sequencing is subject to various requirements, including synchronization between the assembly and body workshops, taking into account the specificities of each factory and certain vehicles. Optimizing this process is crucial for Renault, both in terms of costs and industrial performance.

(a) Racks between shops

(b) Paint shop

(c) Body shop

2 Problem statement

We consider the problem of sequencing cars on an assembly line. In most plants, every car has to go through three shops. They start with the body shop, where the structure of the car is built. They then go to the paint shop, where the car is painted. The last shop is the assembly line, where the car is assembled and completed. We may consider the scheduling of the full three shops, or only one shop or two consecutive shops.

We consider a sequence $(1,\ldots,S)$ of shops s. Let $V=\{1,\ldots,n\}$ be a set of n vehicles v that have to go through the shops. Each vehicle has to go through shop 1 to shop S in this order.

Given an integer $k\in\mathbb{Z}_+$, let us denote by $[k]$ the set $\{1,\ldots,k\}$. A *permutation* σ of $[n]$ is a bijection from $[n]$ onto itself. The notation $v_1\ldots v_n$ encodes the permutation $\sigma : t\mapsto v_t$. We denote by σ^{-1} the *inverse permutation* of σ. We have $\sigma^{-1}\circ\sigma = \mathrm{id} = 1\ldots n$. Let us denote by $T=\{1,\ldots,n\}$ the set of positions in a sequence. We encode a car sequence by a permutation $\sigma : T\to V$ with $\sigma(t)$ being the vehicle in position t. In other words, $\sigma = v_1\ldots v_n$ where v_i is the vehicle in position i. The inverse permutation $\sigma^{-1} : V\to T$ gives the position $\sigma^{-1}(v)$ of vehicle v in the sequence. In other words, $\sigma^{-1}=t_1\ldots t_n$, where vehicle v is in position $t_v=\sigma^{-1}(v)$ in the sequence.

A solution x is a sequence $(\sigma_s,\tilde\sigma_s)_{s\in\{1,\ldots,s\}}$ of permutations of $[n]$, where $\sigma_s=v_1\ldots v_n$ and $\tilde\sigma_s=\tilde v_1\ldots\tilde v_n$ are the sequences of vehicles at the entrance and the exit of shop s, respectively.

Two-tone vehicles. If s is not a paint shop, then the sequences at the entrance and the exit of the shop are the same.

$$\sigma_s=\tilde\sigma_s$$

Two-tone vehicles have to go twice through the paint shop, to be able to paint the different colors. They therefore spend more time in the paint shop. Let $D\subset V$ be the set of two-tone vehicles. If v belongs to D, then it moves backward by $\delta\in\mathbb{Z}_+$ positions in the sequence between σ_s and $\tilde\sigma_s$. Here is an example with $\delta=3$. Two-tone vehicles are in bold.

$$\underbrace{\boldsymbol{v}_1,v_2,\boldsymbol{v}_3,v_4,v_5,v_6,v_7,\boldsymbol{v}_8,\boldsymbol{v}_9,v_{10}}_{\sigma_s}\qquad\underbrace{v_2,v_4,\boldsymbol{v}_1,v_5,\boldsymbol{v}_3,v_6,v_7,v_{10},\boldsymbol{v}_8,\boldsymbol{v}_9}_{\tilde\sigma_s}\tag{1}$$

The formal definition of σ'_s from σ_s requires the notion of cycle. A *cycle permutation* σ, denoted by $(v_0,v_1,\ldots,v_{k-1})$, gives the positions of element v_0 after successive applications of σ, with $v_i=\underbrace{\sigma\circ\ldots\circ\sigma}_{i\text{ times}}(v_0)$ and $v_0=\underbrace{\sigma\circ\ldots\circ\sigma}_{k\text{ times}}(v_0)$. All the elements that are not in the cycle are fixed points of σ. Let $u_1,\ldots,u_k$ be the elements of D ordered by increasing position $t_u=\sigma_s^{-1}(u)$ in the sequence σ_s. We have

$$\tilde\sigma_s^{-1}=\tau_{t_{u_k}}\circ\ldots\circ\tau_{t_{u_1}}\circ\sigma_s^{-1}\quad\text{where}\quad\begin{cases}\tau_t=(t,t+\tilde\delta(t),t+\tilde\delta(t)-1,\ldots,t+1)\\ \tilde\delta(t)=\min(\delta,n-t)\\ t_{u_i}=\tau_{t_{u_{i-1}}}\circ\ldots\circ\tau_{t_{u_1}}\circ\sigma_s^{-1}(u_i)\end{cases}\tag{2}$$

Looking at the inverse permutation σ^{-1}, we get

i	1	2	3	4	5	6	7	8	9	10
$\sigma(i) = v_i$	1	2	3	4	5	6	7	8	9	10
$\sigma^{-1}(i) = t_i$	1	2	3	4	5	6	7	8	9	10

Let us apply to it the first cycle permutation $(t_1, t_1 + 3, t_1 + 2, t_1 + 1)$, we get

i	1	2	3	4	5	6	7	8	9	10
$\sigma(i) = v_i$	2	3	4	1	5	6	7	8	9	10
$\sigma^{-1}(i) = t_i$	4	1	2	3	5	6	7	8	9	10

We then apply $(t_3, t_3 + 3, t_3 + 2, t_3 + 1)$ and get

i	1	2	3	4	5	6	7	8	9	10
$\sigma(i) = v_i$	2	4	1	5	3	6	7	8	9	10
$\sigma^{-1}(i) = t_i$	3	1	5	2	4	6	7	8	9	10

We then apply $(t_8, t_8 + 2, t_8 + 1)$ and get

i	1	2	3	4	5	6	7	8	9	10
$\sigma(i) = v_i$	2	4	1	5	3	6	7	9	10	8
$\sigma^{-1}(i) = t_i$	3	1	5	2	4	6	7	10	8	9

We finally apply $(t_9, t_9 + 2, t_9 + 1)$ and get

i	1	2	3	4	5	6	7	8	9	10
$\sigma(i) = v_i$	2	4	1	5	3	6	7	10	8	9
$\sigma^{-1}(i) = t_i$	3	1	5	2	4	6	7	9	10	8

and we fall back on the solution presented in Equation (1).

A solution $x = (\sigma_s, \tilde{\sigma}_s)_{s\in\{1,\ldots,s\}}$ is *feasible* if σ_s and $\tilde{\sigma}_s$ satisfy (2) for each paint shop s, and $\sigma_s = \tilde{\sigma}_s$ for each other shop s.

Resequencing between shops. Shop $s + 1$ operates with a lag of $\Delta_s \in \mathbb{Z}_+$ timesteps compared to shop s. Between two shops, vehicles are stored in racks. At every timestep t, any vehicle that got out from s is in the rack and can be inserted in the sequence of $s + 1$. It means that sequences $\tilde{\sigma}_s$ and σ_{s+1} must satisfy

$$t_v = \sigma_{s+1}^{-1}(v) \geq \tilde{\sigma}_s^{-1}(v) - \Delta_s = \tilde{t}_v - \Delta_s$$

where $\tilde{t}_v$ is the position of vehicle v in the sequence $\tilde{\sigma}_s$, and t_v is the position of vehicle v in the sequence σ_{s+1}. This constraint models the fact that a vehicle that has not exited s cannot enter $s + 1$. Failure to satisfy this constraint in a solution $x = (\sigma_s, \tilde{\sigma}_s)_{s\in\{1,\ldots,S\}}$ leads to delays and results in a penalty

$$C^{\mathrm{s}}(x) = \sum_{s=1}^{S-1} \sum_{v\in[n]} c^{\mathrm{s}} \max(0, \tilde{\sigma}_s^{-1}(v) - \Delta_s - \sigma_{s+1}^{-1}(v))$$

where $c^{\mathrm{s}} > 0$ is a given constant.

Lot change constraints. Each shop s has a set L_s of *lot change* constraints ℓ. Each lot change constraint comes with a partition $\mathcal{U}_\ell$ of V into lots of vehicles sharing the same feature. We denote by $U_\ell(v)$ the lot of vehicle v. Whenever the vehicles in position t and $t + 1$ do not belong to the same lot, a penalty c_ℓ^{l} is incurred. For instance, lots may correspond to vehicles of the same color, and the penalty is incurred when two vehicles of different colors are in sequence because paint spray guns must be purged. The lot change cost of a solution $x = (\sigma_s, \tilde{\sigma}_s)_{s\in\{1,\ldots,s\}}$ is therefore

$$C^{\mathrm{l}}(x) = \sum_{s=1}^{S} \sum_{\ell\in L_s} \sum_{t=1}^{n-1} c_\ell^{\mathrm{l}} \mathbb{1}\Big(U_\ell\big(\sigma(t)\big) \neq U_\ell\big(\sigma(t+1)\big)\Big) \quad \text{where} \quad \mathbb{1}(i \neq j) = \begin{cases} 1 & \text{if } i \neq j, \\ 0 & \text{otherwise.} \end{cases}$$

Rolling window constraints. Each shop s has a set R_s of rolling window constraints r. Each rolling window constraint comes with a set V_r of vehicles subject to the constraint, a window size w_r and a maximum number M_r of vehicles. On each subsequence of w_r vehicles in the sequence, at most M_r vehicles can belong to V_r. Failure to satisfy this constraint results in a penalty. The larger the number of violations of such a constraint, the larger the marginal penalty for an additional violation. In practice, for k violations, the cost is $c^{\mathrm{r}}_r k^2$ where $c^{\mathrm{r}}_r > 0$ is a given constant cost. The rolling window cost of a solution $x = (\sigma_s, \tilde{\sigma}_s)_{s\in\{1,\dots,s\}}$ is therefore

$$C^{\mathrm{r}}(x) = \sum_{s=1}^{S} \sum_{r\in R_s} \sum_{t=1}^{n-w_r+1} c^{\mathrm{r}}_r \left[\max\Big(0, \Big[\sum_{t'=t}^{t+w_r-1} \mathbb{1}\big(\sigma('t) \in V_r\big) \Big] - M_r \Big) \right]^2.$$

Batch size constraints. Finally, each shop has a set B_s of batch size constraints b. Each batch size constraint comes with a set V_b of vehicles with a given feature subject to the constraint, a minimum size m_b and a maximum size M_b. The constraint stands that sequences of consecutive vehicles in V_b must have a size between m_b and M_b. The larger the violation, the larger the marginal penalty. This is encoded by a cost

$$\gamma^{\mathrm{b}}_b(k) = c^{\mathrm{b}}_b \big[\max(0, m_b - k, k - M_b) \big]^2$$

associated with a sequence of length k, where $c^{\mathrm{b}}_b > 0$ is a given constant cost. The batch size cost of a solution $x = (\sigma_s, \tilde{\sigma}_s)_{s\in\{1,\dots,s\}}$ is therefore

$$C^{\mathrm{b}}(x) = \sum_{s=1}^{S} \sum_{b\in B_s} \sum_{t=1}^{N-1} \sum_{t'=t}^{N} \mathbb{1}_b(t,t') \gamma^{\mathrm{b}}_b(t'-t+1) \quad \text{where} \quad \mathbb{1}_b(t,t') = \begin{cases} 1 & \text{if } \begin{cases} & \sigma(t-1) \notin V_b \text{ if } t \geq 2 \\ \text{and} & \sigma(t), \dots, \sigma(t') \in V_b \\ \text{and} & \sigma(t'+1) \notin V_b \text{ if } t' \leq n-1 \end{cases} \\ 0 & \text{otherwise.} \end{cases}$$

Problem. The *car sequencing problem* aims at finding a solution x of minimum cost

$$\min_{x\in\mathcal{X}} C^{\mathrm{s}}(x) + C^{\mathrm{l}}(x) + C^{\mathrm{r}}(x) + C^{\mathrm{b}}(x). \tag{3}$$

3 Instance format and solutions

Instances. 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`

```
{
    "shops": [
        {
            "name": "body",
            "resequencing_lag": 1
        },
        {
            "name": "paint",
            "resequencing_lag": 2
        },
        {
            "name": "assembly",
            "resequencing_lag": 0
        }
    ],
    "parameters": {
        "two_tone_delta": 2,
        "resequencing_cost": 20
    },
    "vehicles": [
        {"id": 1, "type": "regular"},
        {"id": 2, "type": "regular"},
        {"id": 3, "type": "two-tone"},
        {"id": 4, "type": "regular"},
        {"id": 5, "type": "two-tone"}
    ],
    "constraints": [
        {
            "id": 1,
            "type": "batch_size",
            "shop": "body",
            "cost": 2,
            "min_vehicles": 2,
            "max_vehicles": 4,
            "vehicles": [1, 2, 4]
        },
```

Listing 2: `tiny.json` (continuation)

```
        {
            "id": 2,
            "type": "lot_change",
            "shop": "paint",
            "cost": 3,
            "partition": [
                [1, 2],
                [3, 4, 5]
            ]
        },
        {
            "id": 3,
            "type": "rolling_window",
            "shop": "assembly",
            "cost": 2,
            "window_size": 3,
            "max_vehicles": 2,
            "vehicles": [1, 2, 5]
        },
        {
            "id": 4,
            "type": "batch_size",
            "shop": "assembly",
            "cost": 3,
            "min_vehicles": 2,
            "max_vehicles": 3,
            "vehicles": [1, 2, 3]
        },
        {
            "id": 5,
            "type": "batch_size",
            "shop": "assembly",
            "cost": 3,
            "min_vehicles": 1,
            "max_vehicles": 2,
            "vehicles": [4, 5]
        }
    ]
}
```

Let us now briefly describe its syntax. The JSON file describing an instance of the car sequencing problem contains the following attributes:

- `shops`: A list of strings representing the names of the shops involved in the process. For example, "body", "paint", and "assembly". This corresponds to the sequence of shops $s \in \{1, \dots, S\}$.
- `parameters`: An object containing general parameters for the problem.
  - `two_tone_delta` (δ): An integer representing the delta value for two-tone vehicles, which specifies by how many positions a two-tone vehicle is shifted backward.
  - `resequencing_cost` (c^s): An integer representing the cost of resequencing vehicles between shops.
- `vehicles`: A list of objects, each representing a vehicle with the following attributes:
  - `id` ($v \in V$): An integer representing the unique identifier of the vehicle.

- `two_tone`: A binary attribute (0 or 1) indicating whether the vehicle is two-tone (1) or not (0). This corresponds to the set $D \subset V$ of two-tone vehicles.

- `constraints`: A list of objects, each representing a constraint with the following attributes:
  - `id`: An integer representing the unique identifier of the constraint.
  - `type`: A string representing the type of the constraint. Possible values include "batch_size", "lot_change", and "rolling_window".
  - `shop` (s): A string representing the shop to which the constraint applies.
  - `cost` (c_c): An integer representing the cost associated with the constraint.
  - `vehicles` (V_c) (for `batch_size` and `rolling_window` constraints): A list of integers representing the IDs of the vehicles V_b and V_r concerned by the constraint.
  - `partition` (P_c) (for `lot_change` constraints): A list of lists, each representing a partition of vehicle IDs.
  - `window_size` (w_r) (for `rolling_window` constraints): An integer representing the size of the rolling window.
  - `min_vehicles` (m) (for `batch_size` constraints): An integer representing the minimum number of vehicles allowed (0 if not given).
  - `max_vehicles` (M) (for `rolling_window` and `batch_size` constraints): An integer representing the maximum number of vehicles allowed.

Solutions. The json file containing the solution contains the sequence $\sigma(1)\sigma(2)\ldots\sigma(n) = v_1 \ldots v_n$ at the entry and the exit of each shop.

Listing 3: `tiny_sol.json`

```
{
    "body": {
        "entry": [1, 2, 3, 4, 5],
        "exit": [1, 2, 3, 4, 5]
    },
    "paint": {
        "entry": [1, 2, 3, 4, 5],
```

Listing 4: `tiny_sol.json` (continuation)

```
        "exit": [1, 2, 4, 3, 5]
    },
    "assembly": {
        "entry": [1, 2, 4, 3, 5],
        "exit": [1, 2, 4, 3, 5]
    }
}
```

4 Instructions

4.1 Questions (6 points)

In this section, **we simplify the problem by making the following assumptions**:

- We drop the **two-tone constraints**. Then, the sequences at the entrance and the exit of all shops are the same, i.e., $\sigma_s = \tilde{\sigma}_s$ for all $s \in \{1, \ldots, S\}$. In that case, we can drop $\tilde{\sigma}_s$ and a solution is only defined with the sequences $(\sigma_s)_{s\in\{1,\ldots,S\}}$.

- In the **lot change constraints**, the penalty becomes linear. Hence, the rolling window cost of a solution becomes:

$$C^{\mathrm{r}}(x) = \sum_{s=1}^{S} \sum_{r \in R_s} \sum_{t=1}^{n-w_r+1} c_r^{\mathrm{r}} \Big[\max\Big(0, \Big[\sum_{t'=t}^{t+w_r-1} \mathbb{1}\big(\sigma('t) \in V_r\big) \Big] - M_r \Big) \Big].$$

The goal of this section is to derive a MILP formulation for the problem obtained with these hypotheses. Let x_{svt} be a binary variable equal to 1 if vehicle v is in position t at the entrance of s, i.e., $\sigma_s^{-1}(v) = t$ and $\sigma_s(t) = v$, and 0 otherwise. **All the constraints and the objective must be linear**

1. Write linear constraint(s) on x ensuring that, for each s, $(x_{svt})_{vt}$ encodes a permutation.
2. We introduce a variable y_{sv} for each $s \in \{1, \dots, S-1\}$ and v. Write linear constraint(s) on x and y ensuring that
$$y_{sv} \geq \max(0, \tilde{\sigma}_s^{-1}(v) - \Delta_s - \sigma_{s+1}^{-1}(v)).$$
3. We introduce a variable $f_{s\ell t}$ for each shop $s \in \{1, \dots, S\}$, lot change constraint $\ell \in L_s$ and position $t \in \{1, \dots, n-1\}$. Write linear constraint(s) on f and x ensuring that
$$f_{s\ell t} \geq \mathbb{1}\Big(U_\ell\big(\sigma(t)\big) \neq U_\ell\big(\sigma(t+1)\big)\Big)$$
4. We introduce a variable z_{srt} for each s, $r \in R_s$ and $t \in \{1, n - w_r + 1\}$. Write linear constraint(s) ensuring that $z_{srt} \geq \max\Big(0, \Big[\sum_{t'=t}^{t+w_r-1} \mathbb{1}\big(\sigma('t) \in V_r\big)\Big] - M_r\Big)$
5. We introduce a variable $g_{sbtt'}$ for each s, $b \in B_s$, $t \in \{1, \dots, n\}$ and $t' \in \{t, \dots, n\}$. Write linear constraint(s) ensuring that
$$g_{sbtt'} \geq 1 \text{ if } \begin{cases} & \sigma(t-1) \notin V_b \text{ if } t \geq 2 \\ \text{and} & \sigma(t), \dots, \sigma(t') \in V_b \\ \text{and} & \sigma(t'+1) \notin V_b \text{ if } t' \leq n-1 \end{cases}$$
6. Write the full problem (without two-tone constraints and linear penalty in the lot change constraints) as a Mixed Integer Linear Program (MILP).
7. Test the formulation on the small instances. 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

Mathis Brichet has made available parsers in `julia` and a solution evaluator in julia: https://github.com/mathbri/Kiro24Base.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`.