



# 2024 edition, sponsored by Renault Car sequencing at Renault

KI026

T. LEONARD, A. NGUYEN, C. SERRANO, N. YOUNES  
M. BRICHET, T. PRUNET and A. PARMENTIER

## 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  $(1, \dots, S)$  of shops  $s$ . Let  $V = \{1, \dots, n\}$  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  $\{1, \dots, k\}$ . A *permutation* sigma of  $[n]$  is a bijection from  $[n]$  onto itself. The notation  $v_1 \dots v_n$  encodes the permutation  $\sigma : t \mapsto v_t$ . We denote by  $\sigma^{-1}$  the *inverse permutation* of  $\sigma$ . We have  $\sigma^{-1} \circ \sigma = \text{id} = 1 \dots n$ . Let us denote by  $T = \{1, \dots, n\}$  the set of positions in

a sequence. We encode a car sequence by a permutation  $\sigma : T \rightarrow V$  with  $\sigma(t)$  being the vehicle in position  $t$ . In other words,  $\sigma = v_1 \dots v_n$  where  $v_i$  is the vehicle in position  $i$ . The inverse permutation  $\sigma^{-1} : V \rightarrow T$  gives the position  $\sigma^{-1}(v)$  of vehicle  $v$  in the sequence. In other words,  $\sigma^{-1} = t_1 \dots t_n$ , where vehicle  $v$  is in position  $t_v = \sigma^{-1}(v)$  in the sequence.

A solution  $x$  is a sequence  $(\sigma_s, \tilde{\sigma}_s)_{s \in \{1, \dots, s\}}$  of permutations of  $[n]$ , where  $\sigma_s = v_1 \dots v_n$  and  $\tilde{\sigma}_s = \tilde{v}_1 \dots \tilde{v}_n$  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  $\sigma_s$  and  $\tilde{\sigma}_s$ . Here is an example with  $\delta = 3$ . Two-tone vehicles are in bold.

$$\underbrace{v_1, v_2, \mathbf{v_3}, v_4, v_5, v_6, v_7, \mathbf{v_8}, \mathbf{v_9}, v_{10}}_{\sigma_s} \quad \underbrace{v_2, v_4, \mathbf{v_1}, v_5, \mathbf{v_3}, v_6, v_7, v_{10}, \mathbf{v_8}, \mathbf{v_9}}_{\tilde{\sigma}_s} \quad (1)$$

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

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

Looking at the inverse permutation  $\sigma^{-1}$ , we get

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

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

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

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

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

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

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

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

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

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

A solution  $x = (\sigma_s, \tilde{\sigma}_s)_{s \in \{1, \dots, S\}}$  is *feasible* if  $\sigma_s$  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  $\sigma_{s+1}$  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  $\sigma_{s+1}$ . 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  $x = (\sigma_s, \tilde{\sigma}_s)_{s \in \{1, \dots, S\}}$  leads to delays and results in a penalty

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

where  $c^s > 0$  is a given constant.

**Lot change constraints.** Each shop  $s$  has a set  $L_s$  of *lot change* constraints  $\ell$ . 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  $c_\ell^1$  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  $x = (\sigma_s, \tilde{\sigma}_s)_{s \in \{1, \dots, S\}}$  is therefore

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

**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  $c_r^x k^2$  where  $c_r^x > 0$  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

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

**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

$$\gamma_b^b(k) = c_b^b [\max(0, m_b - k, k - M_b)]^2$$

associated with a sequence of length  $k$ , where  $c_b^b > 0$  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

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

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

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

### 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

```

1 {
2   "shops": [
3     {
4       "name": "body",
5       "resequencing_lag": 1
6     },
7     {
8       "name": "paint",
9       "resequencing_lag": 2
10    },
11    {
12      "name": "assembly",
13      "resequencing_lag": 0
14    }
15  ],
16  "parameters": {
17    "two_tone_delta": 2,
18    "resequencing_cost": 20
19  },
20  "vehicles": [
21    {"id": 1, "type": "regular"},
22    {"id": 2, "type": "regular"},
23    {"id": 3, "type": "two-tone"},
24    {"id": 4, "type": "regular"},
25    {"id": 5, "type": "two-tone"}
26  ],
27  "constraints": [
28    {
29      "id": 1,
30      "type": "batch_size",
31      "shop": "body",
32      "cost": 2,
33      "min_vehicles": 2,
34      "max_vehicles": 4,
35      "vehicles": [1, 2, 4]
36    },

```

Listing 2: tiny.json (continuation)

```

37    {
38      "id": 2,
39      "type": "lot_change",
40      "shop": "paint",
41      "cost": 3,
42      "partition": [
43        [1, 2],
44        [3, 4, 5]
45      ]
46    },
47    {
48      "id": 3,
49      "type": "rolling_window",
50      "shop": "assembly",
51      "cost": 2,
52      "window_size": 3,
53      "max_vehicles": 2,
54      "vehicles": [1, 2, 5]
55    },
56    {
57      "id": 4,
58      "type": "batch_size",
59      "shop": "assembly",
60      "cost": 3,
61      "min_vehicles": 2,
62      "max_vehicles": 3,
63      "vehicles": [1, 2, 3]
64    },
65    {
66      "id": 5,
67      "type": "batch_size",
68      "shop": "assembly",
69      "cost": 3,
70      "min_vehicles": 1,
71      "max_vehicles": 2,
72      "vehicles": [4, 5]
73    }
74  ]
75 }

```

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** ( $\delta$ ): 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  $\sigma(1)\sigma(2)\dots\sigma(n) = v_1\dots v_n$  at the entry and the exit of each shop.

Listing 3: tiny\_sol.json

```

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

```

Listing 4: tiny\_sol.json (continuation)

```

8     "exit": [1, 2, 4, 3, 5]
9   },
10  "assembly": {
11    "entry": [1, 2, 4, 3, 5],
12    "exit": [1, 2, 4, 3, 5]
13  }
14 }

```

## 4 Who wins

The score of a team is the sum of the value of the solution returned on each instance. If you do not return a solution on an instance, a very high “default” score will be attributed to you on this instance. The team with 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.