

# Air cargo load and route planning in pickup and delivery operations

## ARTICLE HISTORY

Compiled November 28, 2023

## ABSTRACT

We model and solve the problem of planning the loading and routing of an aircraft according to a utility score, weight and balance principles, fuel consumption, and cost increases due to centre of gravity displacement in a tour of simultaneous pickup and delivery. This is necessary because in the aerial pickup and delivery of goods, transport aviation faces risks of cargo unbalancing and wrong delivery due to the urgency required for loading for mission accomplishment. It is solved by integer programming and four known meta-heuristics to select the method that most fits the operational requirements. We also created a heuristic that quickly finds viable solutions for a wide range of problem sizes in much less than the operationally acceptable time. The output is an essential part of airlift because it guarantees flight safety, makes ground operations more efficient, and makes sure that each item gets to its right destination.

## KEYWORDS

Air Cargo Optimization; Air Palletization; Weight and Balancing; Integer Programming; Meta-heuristics

## 1. Solution strategies

This problem's work's requirements are presented in the mathematical model, and it was verified that ACLP+RPDP is *NP-hard*. Let's dive into the strategies adopted to make the exact solution run in polynomial time. One common strategy is to use approximation algorithms that provide near-optimal solutions with a guaranteed performance ratio. Another approach is to develop heuristics that may not guarantee an optimal solution but can quickly find suitable solutions in practice. These strategies aim to strike a balance between computational efficiency and solution quality.

Throughout our research, we have thoughtfully described the ACLP+RPDP model in standard MIP format and found that no MIP solver can not handle its practical cases in a feasible time. Thus, as ACLP+RPDP is highly complex, involving four intractable sub-problems (APP, WBP, PDP, and TSP), our strategy will be to focus only on *real cases* and to develop quick node-by-node solutions, close to optimal, that allow us to build a complete tour.

ACLP+RPDP solutions define a tour and a corresponding loading and unloading plan at each node. In this way, equipment's use at each node will be the only factor limiting loading and unloading times.

On the other hand, we know that transport aircraft generally have a few dozen pallets, flight itineraries have fewer than six nodes, and each node has hundreds of items to be shipped. We also know that missions with fewer nodes are more frequent than longer ones. Under these circumstances, we can adopt two more strategies:

- We will consider that the number of destinations exceeds 1, to avoid the case of a single and trivial tour, but less than the number of pallets, that is,  $1 < K < m$ . So we can preset pallets' destinations at each node. In this way, we will reserve a number of pallets proportional to volumes for each destination at the shipping node. We could have used another criterion, but it was observed in the experiments that volume is more constrictive in airlift.
- As the number of nodes is small, we have the chance to check the solution corresponding to the tour with the shortest total distance and even test all possible tours, selecting the one that provides the best value for the objective function.

Our complete strategy is summarised in Algorithm 1.

---

**Algorithm 1** Solving the ACLP+RPDP

---

```

1: ACLP+RPDP  in: scenario, surplus, tmax  out: answer1, answer2
2: Let  $M$  be the set of pallets (cfr. Table ??)
3: Let  $K$ ,  $L$  and  $C$  be according to scenario (cfr. Tables ??, ?? and 1)
4: Let  $\pi_{TSP1}$  and  $\pi_{TSP2}$  be the shortest tours (cfr. Table 1)
5:  $N \leftarrow \text{ItemsGeneration}(\text{scenario}, \text{surplus})$ 
6: for each method do
7:    $f_1 \leftarrow \text{SolveTour}(\pi_{TSP1}, L, M, C, N, \text{method}, tmax)$  ▷ Algorithm 2
8:    $f_2 \leftarrow \text{SolveTour}(\pi_{TSP2}, L, M, C, N, \text{method}, tmax)$ 
9:    $\text{answer1}[\text{scenario}, \text{surplus}, \text{method}] \leftarrow \max(f_1, f_2)$ 
10:  for each  $\pi \in S_K$  do
11:     $f_\pi \leftarrow \text{SolveTour}(\pi, L, M, C, N, \text{method}, tmax/K!)$ 
12:  end for
13:   $\text{answer2}[\text{scenario}, \text{surplus}, \text{method}] \leftarrow \max f_\pi$ 
14: end for

```

---

In this algorithm, we use five values for *scenario*, according to Tables ?? and 1, which define the number  $K$  of destinations, the set  $L$  of nodes, costs  $C$ , and the shortest tours  $\pi_{TSP1}$  and  $\pi_{TSP2}$ .

**Table 1.:** Testing scenarios

Scenario	$K$	$L$	$\pi_{TSP1}$	$\pi_{TSP2}$
1	2	{0, 1, 2}	0 1 2 0	0 2 1 0
2	3	{0, 1, 2, 3}	0 1 2 3 0	0 3 2 1 0
3	4	{0, 1, 2, 3, 4}	0 4 1 2 3 0	0 3 2 1 4 0
4	5	{0, 1, 2, 3, 4, 5}	0 4 1 2 5 3 0	0 3 5 2 1 4 0
5	6	{0, 1, 2, 3, 4, 5, 6}	0 4 1 2 6 5 3 0	0 3 5 6 2 1 4 0

The parameter *surplus* is a value in  $\{1.2, 1.5, 2.0\}$ , which corresponds, at  $\pi_k$ , to the ratio between items' sum of volumes and pallets' load capacity ( $\text{surplus} = \sum_{j=1}^{n_k} v_j / \sum_{i=1}^m V_i$ ). This parameter allows us to verify the different behaviours of each *method*, according to the *scenario* and quantities of items available for shipment. It is passed to *ItemsGeneration* (line 5), responsible for selecting items to be shipped, which will be presented in the next section (Algorithm 7).

Parameter *tmax* is a run time limit that will be distributed among tours (lines 7, 8, and 11). *method* corresponds to a MIP solver or a heuristic to the node-by-node solution *SolveTour*, which will be presented in subsection 1.2.

The best results corresponding to the shortest tours are stored in *answer1* (line 9), and those obtained by testing all  $K!$  tours are stored in *answer2* (line 13).

Next, we will present two subsections: in the first, we explain how *SolveTour* is executed, presetting pallets' destinations. In the second, we will present the heuristics developed for node-by-node solutions.

### 1.1. *SolveTour* algorithm

As we commented in the previous subsection, we will adopt the strategy of presetting the destinations of each pallet throughout the tour. This is feasible in practical cases where  $1 < K < m$ . For this, each pallet  $i$  also has a field  $T_i^k$ ,  $0 \leq k \leq K$ , which stores its next destination after being loaded at  $\pi_k$ . For this reason,  $T_i^k \in L_k$ ,  $1 \leq i \leq m$ , and  $0 \leq k \leq K$ .

*SolveTour* is described in Algorithm 2, where  $\pi$  is a node permutation (excluding the base) that defines the order of visits in this tour, *method* corresponds to a MIP solver or a heuristic for solving node-by-node sub-problems, and *tmax* is the run time limit that will be distributed among the  $K + 1$  legs of the tour.

---

#### Algorithm 2 Solving the tour $\pi$ with *method*

---

```

1: SolveTour   in:  $\pi, L, M, C, N, method, tmax$    out: score/cost
2:  $\pi_0 \leftarrow 0$                                       $\triangleright$  all tours start and end at the base.
3:  $\pi_{K+1} \leftarrow 0$ 
4:  $score \leftarrow 0$ 
5:  $cost \leftarrow 0$ 
6: for  $k \leftarrow 0$  to  $K$  do
7:    $L_{\pi_k} \leftarrow L - \{\pi_0, \pi_1, \dots, \pi_k\}$             $\triangleright$  set of remaining nodes is updated.
8:    $T_i^{\pi_k} \leftarrow -1, 1 \leq i \leq m$                       $\triangleright$  pallet destination is unset.
9:   if  $k = 0$  then
10:    Let  $G_1(M \cup N_0, \emptyset)$                                 $\triangleright$  no packed contents at the base.
11:   else
12:     $E_{Q_{\pi_k}}, M \leftarrow UpdatePacked(M, Q_{\pi_k}, \pi_k)$     $\triangleright$  Algorithm 3
13:    Let  $G_1(M \cup N_{\pi_k} \cup Q_{\pi_k}, E_{Q_{\pi_k}})$ 
14:   end if
15:    $M \leftarrow SetPalletsDestinations(M, \pi_k)$                 $\triangleright$  Algorithm 4
16:    $G_2 \leftarrow SolveNode(method, \pi_k, G_1, tmax/(K + 1))$     $\triangleright$  A chosen method for  $\pi_k$ 
17:    $s, \tau \leftarrow ScoreAndTorque(\pi_k, G_2)$                   $\triangleright$  Algorithm 5
18:    $score \leftarrow score + s$ 
19:    $cost \leftarrow cost + c_{\pi_k, \pi_{k+1}} \times (1 + c_g \times |\tau|)$ 
20: end for

```

---

As all tours start and end at the base 0 (lines 2-3), and after initialising the score and cost values (lines 4-5), there is a loop for the  $K + 1$  flights (lines 6-20). The set  $L_{\pi_k}$  of remaining nodes is updated (line 7), and pallet destinations are unset (line 8).

When the aircraft is at the base, the initial graph  $G_1$  is empty, and there are no packed contents 10. Otherwise, *UpdatePacked* (line 12) returns the set of packed contents that have not yet reached their destination and remain on board, rearranging them on pallets to minimise CG deviation. This allocation is stored in graph  $G_1$  (line 13).

*UpdatePacked*, described in Algorithm 3, finds the best packed-pallet allocation, in terms of CG deviation, for packed contents that remain on board.

---

**Algorithm 3** Updating packed contents that remain boarded at  $\pi_k$ 


---

```

1: UpdatePacked  in:  $M, Q_{\pi_k}, \pi_k$    out:  $E_{Q_{\pi_k}}, M$ 
2:  $E_{Q_{\pi_k}} \leftarrow \text{MinCGDeviation}(E_{Q_{\pi_k}})$  ▷ equations 1, 2, and 3
3: for  $i \leftarrow 1$  to  $m$  do
4:   for  $q \leftarrow 1$  to  $m_{\pi_k}$  do
5:      $T_i^{\pi_k} \leftarrow -1$ 
6:     if  $(i, q) \in E_{Q_{\pi_k}}$  then
7:        $T_i^{\pi_k} \leftarrow to_q$  ▷ reassign pallet destinations.
8:     end if
9:   end for
10: end for

```

---

*SetPalletsDestinations* (line 15 from Algorithm 2) presets the destination of each pallet based on the current node's volume demands, without changing the pallet's destination with packed contents, as described in Algorithm 4.

*SolveNode* includes edges corresponding to items shipped at the current node, returning the graph  $G_2$  (line 15). Score and CG deviation of  $G_2$  are calculated (line 17) and accumulated (lines 18-19), allowing the final result of this tour as output.

*MinCGDeviation* (line 2) relocates packed contents on pallets, minimising torque and ensuring that they all remain on board, one packed content on each pallet. It is run through a MIP solver with the objective function (1) and constraints (2) and (3). As there are few variables,  $E_{Q_{\pi_k}}$  is obtained in less than 30 milliseconds. Finally, the destination of each pallet with packed content is updated (lines 3-10).

$$\min \left| \sum_{i=1}^m \sum_{q=1}^{m_{\pi_k}} Y_{iq}^k \times w_q \times D_i^{long} \right| \quad (1)$$

$$\sum_{i=1}^m Y_{iq}^k = 1; \quad q \in \{1, \dots, m_{\pi_k}\} \quad (2)$$

$$\sum_{q=1}^{m_{\pi_k}} Y_{iq}^k \leq 1; \quad i \in \{1, \dots, m\} \quad (3)$$

The parameter *vol* stores the demand volume of items destined for the non-visited nodes (line 2). The destination of empty pallets is defined proportionally to the volume of items to be embarked (lines 14-25). *max* is the destination with maximum volume demand (line 10), and *needed* is the number of necessary pallets to node  $x$  (line 16). The destination with the maximum volume defines any remaining pallets (lines 26-30).

The procedure *ScoreAndTorque*, which is explained in Algorithm 5, looks at the allocation graph  $G$  made by *SolveNode* at  $\pi_k$  and gives back the cargo score and the aircraft torque that go with it.

The evaluation algorithm (5) has a loop that goes through all pallets (lines 5-18), adding up the items' scores (lines 8 and 14) and torques (lines 9 and 15) so that the aircraft torque (line 19) can be found.

### 1.2. Node-by-node solutions

In this subsection, we present two implementations of the *SolveNode* algorithm: with a MIP solver and with heuristics.

---

**Algorithm 4** Setting pallets destination based on items to be embarked at  $\pi_k$ 


---

```

1: SetPalletsDestinations  in:  $M, \pi_k$    out:  $M$ 
2:  $vol_x \leftarrow 0, x \in L_{\pi_k}$ 
3:  $max \leftarrow 0$  ▷ destination with maximum volume demand.
4:  $total \leftarrow 0$ 
5: for  $j \leftarrow 1$  to  $n_{\pi_k}$  do
6:   if  $to_j \in L_{\pi_k}$  then
7:      $vol_{to_j} \leftarrow vol_{to_j} + v_j$ 
8:      $total \leftarrow total + v_j$ 
9:     if  $vol_{to_j} > vol_{max}$  then
10:       $max \leftarrow to_j$ 
11:     end if
12:   end if
13: end for
14: for  $x \in L_{\pi_k}$  do
15:   if  $vol_x \neq 0$  then
16:      $needed \leftarrow \max\{1, \lfloor (m - m_{\pi_k}) \times vol_x / total \rfloor\}$ 
17:      $np \leftarrow 0$ 
18:     for  $i \leftarrow 1$  to  $m$  do
19:       if  $(np < needed)$  and  $(T_i^{\pi_k} = -1)$  then
20:          $T_i^{\pi_k} \leftarrow x$ 
21:          $np \leftarrow np + 1$  ▷ number of necessary pallets to node  $x$ .
22:       end if
23:     end for
24:   end if
25: end for
26: for  $i \leftarrow 1$  to  $m$  do
27:   if  $T_i^{\pi_k} \leftarrow -1$  then
28:      $T_i^{\pi_k} \leftarrow max$  ▷ any remaining pallet is assigned to the maximum demand destination.
29:   end if
30: end for

```

---

### 1.2.1. Node-by-node solutions with a MIP solver

Our strategy adopted in *SolveTour* defines the values of some variables: the set of nodes to be visited is updated, packed contents that remain on board are reallocated to minimise the CG deviation, and pallets' destinations are determined according to the volume of items available for shipment.

In this way, the mathematical model for *SolveNode*( $MIP, \pi_k, G, tmax$ ) becomes simpler, which finds an allocation of available items at  $\pi_k$  using previously defined values of  $L_{\pi_k}$ ,  $T_i^{\pi_k}$ , and  $a_q^{\pi_k}$ . So, we use a MIP solver with a time limit  $tmax$  at  $\pi_k$  to find the best solution for the objective function (4) using the calculus equations (5) and (7), while keeping the constraints (8) and (14). Binary variables  $X_{ij}$  and  $Y_{iq}$  define sets of edges  $E_{N_{\pi_k}}$  and  $E_{Q_{\pi_k}}$ , respectively, included in graph  $G$ .

$$\max f = \tilde{s} / \tilde{c} \quad (4)$$

$$\tilde{s} = \sum_{i=1}^m \sum_{j=1}^{n_{\pi_k}} X_{ij} \times s_j \quad (5)$$

$$\tau_{\pi_k} = \sum_{i=1}^m \left[ D_i^{long} \times \left( \sum_{j=1}^{n_{\pi_k}} X_{ij} \times w_j + \sum_{q=1}^{m_{\pi_k}} Y_{iq} \times w_q \right) \right] / W_{max} \times limit_{long}^{CG} \quad (6)$$

---

**Algorithm 5** Cargo score and aircraft torque
 

---

```

1: ScoreAndTorque  in:  $\pi_k, G$    out:  $s, \tau$ 
2: Let  $G(V_{\pi_k}, E_{Q_{\pi_k}} \cup E_{N_{\pi_k}})$ 
3:  $s \leftarrow 0$ 
4:  $\tau_i \leftarrow 0, 1 \leq i \leq m$ 
5: for  $i \leftarrow 1$  to  $m$  do
6:   for  $j \leftarrow 1$  to  $n_{\pi_k}$  do
7:     if  $X_{ij}^{\pi_k} = 1$  then
8:        $s \leftarrow s + s_j$  ▷ accumulates cargo score
9:        $\tau_i \leftarrow \tau_i + w_j \times D_i^{long}$  ▷ accumulates aircraft torque
10:    end if
11:  end for
12:  for  $q \leftarrow 1$  to  $m_{\pi_k}$  do
13:    if  $Y_{iq}^{\pi_k} = 1$  then
14:       $s \leftarrow s + s_q$  ▷ accumulates cargo score
15:       $\tau_i \leftarrow \tau_i + w_q \times D_i^{long}$  ▷ accumulates aircraft torque
16:    end if
17:  end for
18: end for
19:  $\tau \leftarrow \sum_{i=1}^m \tau_i / (W_{max} \times limit_{long}^{CG})$  ▷ final calculation of aircraft torque

```

---

$$\tilde{c} = c_{\pi_k, \pi_{k+1}} \times (1 + c_g \times |\tau_{\pi_k}|) \quad (7)$$

$$|\tau_{\pi_k}| \leq 1 \quad (8)$$

$$\sum_{j=1}^{n_{\pi_k}} X_{ij} \times w_j + \sum_{q=1}^{m_{\pi_k}} Y_{iq} \times w_q \leq W_i; \quad i \in \{1, \dots, m\} \quad (9)$$

$$\sum_{j=1}^{n_{\pi_k}} X_{ij} \times v_j + \sum_{q=1}^{m_{\pi_k}} Y_{iq} \times v_q \leq V_i; \quad i \in \{1, \dots, m\} \quad (10)$$

$$\sum_{i=1}^m X_{ij} \leq 1; \quad j \in \{1, \dots, n_{\pi_k}\} \quad (11)$$

$$X_{ij} = 0; \quad to_j \notin L_{\pi_k}; \quad i \in \{1, \dots, m\}; \quad j \in \{1, \dots, n_{\pi_k}\} \quad (12)$$

$$X_{ij} \leq X_{ij} \times (T_i^{\pi_k} - to_j + 1); \quad i \in \{1, \dots, m\}; \quad j \in \{1, \dots, n_{\pi_k}\} \quad (13)$$

$$X_{ij} \leq X_{ij} \times (to_j - T_i^{\pi_k} + 1); \quad i \in \{1, \dots, m\}; \quad j \in \{1, \dots, n_{\pi_k}\} \quad (14)$$

Constraints (13) and (14) are equivalent to  $X_{ij} = 1$  if  $to_j = T_i^{\pi_k}$ , and  $X_{ij} = 0$  otherwise .

### 1.2.2. Node-by-node solutions with heuristics

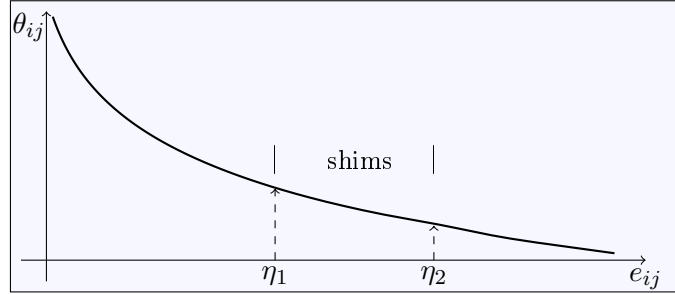
One of the objectives of this work was to find a quick heuristic that offers a good-quality solution for the node-by-node problem. We use well-known meta-heuristics to create algorithms, such as *Ant Colony Optimisation* (ACO) (Dorigo, 1992; Dorigo, Maniezzo and Coloni, 1996), *Noising Method Optimisation* (NMO) (Charon and Hudry, 1993, 2001; Zhan, Wang, Zhang and Zhong, 2020), *Tabu Search* (TS) (Glover, 1986), and *Greedy Randomised Adaptive Search Procedure* (GRASP) (Feo and Resende, 1989). We considered several

ideas from literature (Alonso, Alvarez-Valdes and Parreno, 2019; Fidanova, 2006; Niar and Freville, 1997; Zhan et al., 2020), and we were careful to use the same data structures and procedures in all implementations to enforce fair results comparison.

However, the heuristic that presented better solutions was none of the previous ones. In this subsection, we will present a new heuristic for the node-by-node problem, called *Shims*. Like in mechanics, shims are collections of spacers to fill gaps, which may be composed of parts with different thicknesses. This strategy is based on a practical observation: usually, subsets of smaller and lighter items are saved for later adjustments to the remaining available space.

The selection of edges for  $E_{N_{\pi_k}}$  uses the *edge attractiveness*  $\theta_{ij}$  (15), which can be understood as the tendency to allocate item  $j$  to the pallet  $i$  at  $\pi_k$ . It is directly proportional to the score and inversely proportional to the volume and torque of each item.

$$\theta_{ij} = \frac{s_j}{v_j} \times \left(1 - \frac{w_j \times |D_i^{long}|}{\max(w_j) \times \max(|D_i^{long}|)}\right); i \in \{1, \dots, m\}, j \in \{1, \dots, n_{\pi_k}\} \quad (15)$$



**Figure 1.:**  $n_{\pi_k}$  possible edges  $e_{ij}$  sorted by  $\theta_{ij}$  in non-ascending order

Considering only items that can be shipped at  $\pi_k$ , Figure 1 represents  $n_{\pi_k}$  possible edges  $e_{ij}$  of pallet  $i$  sorted by  $\theta_{ij}$  in non-ascending order. Initially, *Shims* builds a greedy solution for the pallet  $i$  selecting edges up to index  $\eta_1$  (first phase). Then, with edges between  $\eta_1$  and  $\eta_2$ , it elaborates different possible complements (second phase), including later the best ones in the same pallet (third phase). *Shims* is depicted in Algorithm 6.

Initially,  $Q_{\pi_k}$  (line 3) corresponds to packed contents that remain on board.

It's important to keep in mind that the procedures  $UpdatePacked(M, Q_{\pi_k}, \pi_k)$  and  $SetPalletsDestinations(M, \pi_k)$  modified  $E_{Q_{\pi_k}}$  and  $M$ . Then, pallets  $i$  are considered in non-descending order of  $|D_i^{long}|$ .

For each pallet  $i$ , its  $n_{\pi_k}$  possible edges  $e_{ij}$  are considered in non-increasing order of  $\theta_{ij}$ .

In the *greedy phase* (lines 4-21), a partial solution for each pallet  $i$  is constructed by adding edges following  $\theta_{ij}$  non-ascending order. The  $\eta_1$  index corresponds to the accumulated volume equal to  $V_i \times limit$ . In  $\eta_2$  index, this same accumulated volume reaches  $(1 + 2 \times (1 - limit)) \times V_i$ . The size of this range  $[\eta_1, \eta_2]$  was defined empirically, and the value  $limit = 0.92$  (line 7) was determined by the *iRace* tool (Lopez-Ibanez, Dubois-Lacoste, Cáceres, Birattari and Stützle, 2016).

In the *composition phase* (lines 22-28), a set of shims named *Set* is created for each pallet  $i$ , where each shim is formed by a set of edges in the range  $[\eta_1, \eta_2]$ , whose total volume is limited by  $slack_i$ . In this phase, the heuristic that provided the best results, both in terms of time and quality, is based on *First-Fit Decreasing*, which is an approximation algorithm for the *Bin Packing Problem* (Johnson and Garey, 1985). Basically, shims are created by accumulating the following edges, taking  $slack_i$  as a limit.

In the *selection phase* (lines 29-52), the best shim in *Set* is chosen. Initially, two shims

are found:  $sh_w$  with larger weight and  $sh_v$  with larger volume. Between the two, the one with the highest score will be chosen, and its edges will be inserted into  $E_{N_{\pi_k}}$ .



---

**Algorithm 6** *Shims* heuristic at  $\pi_k$ 

---

```
1: SolveNode in:  $Shims, \pi_k, G, tmax$  out:  $G(M \cup N_{\pi_k} \cup Q_{\pi_k}, E_{Q_{\pi_k}} \cup E_{N_{\pi_k}})$ 
2:  $T_{begin} \leftarrow$  current system time
3: Let  $G(M \cup N_{\pi_k} \cup Q_{\pi_k}, E_{Q_{\pi_k}})$ 
4: Sort  $M$  by  $|D_i^{long}|$  in non-descending order
5:  $E_{N_{\pi_k}} \leftarrow \emptyset$ 
6:  $\tau_{max} \leftarrow W_{max} \times limit_{long}^{CG}$ 
7:  $limit \leftarrow 0.92$ 
8: for  $i \leftarrow 1$  to  $m$  do
9:    $\tau_{\pi_k} \leftarrow \sum_{(i,q) \in E_{Q_{\pi_k}}} w_q \times D_i^{long}$ 
10:   $vol_i \leftarrow \sum_{(i,q) \in E_{Q_{\pi_k}}} v_q$ 
11:  Let  $E$  be an array of  $n_{\pi_k}$  possibles edges of pallet  $i$  sorted by  $\theta_{ij}$  in non-ascending order
12:   $\eta_1 \leftarrow 1$ 
13:  repeat
14:     $e_{ij} \leftarrow E_{\eta_1}$ 
15:    if ( $E_{N_{\pi_k}} \cup \{e_{ij}\}$  is feasible) and ( $vol_i \leq V_i \times limit$ ) and ( $|\tau_{\pi_k} + w_j \times D_i^{long}| \leq W_{max} \times limit_{long}^{CG}$ ) then
16:       $E_{N_{\pi_k}} \leftarrow E_{N_{\pi_k}} \cup \{e_{ij}\}$ 
17:       $vol_i \leftarrow vol_i + v_j$ 
18:       $\tau_{\pi_k} \leftarrow \tau_{\pi_k} + w_j \times D_i^{long}$ 
19:       $\eta_1 \leftarrow \eta_1 + 1$ 
20:    end if
21:  until ( $vol_i > V_i \times limit$ ) or ( $\eta_1 > n_{\pi_k}$ )
22:   $slack_i \leftarrow V_i - vol_i$ 
23:   $\eta_2 \leftarrow \eta_1$ 
24:  while ( $\eta_2 \leq n_{\pi_k}$ ) and ( $vol_i < (1 + 2 \times (1 - limit)) \times V_i$ ) do
25:     $e_{ij} \leftarrow E_{\eta_2}$ 
26:     $vol_i \leftarrow vol_i + v_j$ 
27:     $\eta_2 \leftarrow \eta_2 + 1$ 
28:  end while
29:   $vol \leftarrow 0; b \leftarrow 1; shims_b \leftarrow \emptyset; Set \leftarrow \{shims_b\}$ 
30:  for  $x \leftarrow \eta_1$  to  $\eta_2$  do
31:    if  $T_{current} - T_{begin} > tmax$  then
32:      break
33:    end if
34:     $NewShims \leftarrow \text{True}$ 
35:     $e_{ij} \leftarrow E_x$ 
36:    for  $shims \in Set$  do
37:      if ( $e_{ij} \notin (E_{N_{\pi_k}} \cup shims)$ ) and ( $e_{ij}$  is feasible) and ( $(v_j + vol) \leq slack_i$ ) then
38:         $shims \leftarrow shims \cup \{e_{ij}\}$ 
39:         $vol \leftarrow vol + v_j$ 
40:         $NewShims \leftarrow \text{False}$ 
41:        break
42:      end if
43:    end for
44:    if  $NewShims$  then
45:       $vol \leftarrow 0; b \leftarrow b + 1; shims_b \leftarrow \{e_{ij}\}$ 
46:       $Set \leftarrow Set \cup \{shims_b\}$ 
47:    end if
48:  end for
49:   $sh_w \leftarrow shims$ , where  $shims \in Set$  and  $\sum_{e_{ij} \in shims} w_j$  is maximum
50:   $sh_v \leftarrow shims$ , where  $shims \in Set$  and  $\sum_{e_{ij} \in shims} v_j$  is maximum
51:   $sh_{best} \leftarrow shims$ , where  $shims \in \{sh_w, sh_v\}$  and  $\sum_{e_{ij} \in shims} s_j$  is maximum
52:   $E_{N_{\pi_k}} \leftarrow E_{N_{\pi_k}} \cup sh_{best}$ 
53: end for
```

---

## 2. Implementation and results

This section is composed of two parts: the generation of test instances and the results obtained in our implementation.

### 2.1. Instances generation

As we are dealing with a new problem that, until now, had not been modelled in the literature, we have to create our own benchmarks. For this, we based it on the characteristics of real airlifts carried out by the *Brazilian Air Force* with its own aircraft or air cargo charters, as described below.

In the delivery of supplies carried out in Brazil from 2008 to 2010, 23% of items weighed between 10 kg and 20 kg, 22% from 21 kg to 40 kg, 24% from 41 kg to 80 kg, 23% from 81 kg to 200 kg, and 8% between 201 kg and 340 kg. These five groups of items are described in Table 2, where  $P$  represents the group probability. On the other hand, the average density of these items is approximately  $246 \text{ kg/m}^3$ .

**Table 2.:** Items weight distribution

$x$	$P$	<i>low</i> (kg)	<i>high</i> (kg)
1	0.23	10	20
2	0.22	21	40
3	0.24	41	80
4	0.23	81	200
5	0.08	201	340

In the generation of test instances, we use two types of random selections: *RandomInt*( $i_1, i_2$ ), that randomly selects an integer number in  $[i_1, i_2]$ , where  $i_1$  and  $i_2$  are integer numbers; and *Roulette*(), that is biased through  $P$  to select  $x$  in Table 2.

*ItemsGeneration*, which generates  $N$  (all items to be moved among nodes), is described in Algorithm 7.

Variable *scenario* defines  $L$  and  $M$  (line 2), and argument *surplus* sets a limit on the total volume of items at each node (line 3). To avoid simply loading all items, we use  $\text{surplus} \in \{1.2, 1.5, 2.0\}$ . This also represents more instances for tests in each scenario.

For each generated  $\text{item}_j^k$ , its destination is randomly selected (line 12), its weight has a distribution according to Table 2 (lines 14-15), its score varies 100 (highest) and 5 (lowest) according to a logarithmic scale (line 16), and its volume is randomly defined from the density, where we allow a variation of 40% around the average density of  $246 \text{ kg/m}^3$  (line 17).

## References

- Alonso, M.T., Alvarez-Valdes, R., Parreno, F., 2019. A grasp algorithm for multi-container loading problems with practical constraints. *A Quarterly Journal of Operations Research* 18, 49–72.
- Charon, I., Hudry, O., 1993. The noising method: a new method for combinatorial optimization. *Operations Research Letters* 14, 133–137.
- Charon, I., Hudry, O., 2001. The noising methods: A generalization of some metaheuristics. *European Journal of Operational Research* 135, 86–101.

---

**Algorithm 7** Generating items

---

```
1: ItemsGeneration in: scenario, surplus out: N
2: Let  $L$  be the set of nodes and  $M$  the set of pallets
3:  $limit \leftarrow surplus \times \sum_{i=1}^m V_i$ 
4: for  $k \leftarrow 0$  to  $K$  do
5:    $N_{\pi_k} \leftarrow \emptyset$ 
6:    $j \leftarrow 0$ 
7:    $vol \leftarrow 0$ 
8:   while  $vol < limit$  do
9:      $j \leftarrow j + 1$ 
10:    Let  $item_j^{\pi_k}$  be the item  $j$  at  $\pi_k$ 
11:    repeat
12:       $to_j \leftarrow RandomInt(0, K)$ 
13:    until  $to_j \neq \pi_k$ 
14:     $x = Roulette()$  ▷ biased through  $P$  (Table 2)
15:     $w_j \leftarrow RandomInt(low(x), high(x))$ 
16:     $s_j \leftarrow \lfloor 100 \times (1 - \log_{10}(RandomInt(1, 9))) \rfloor$ 
17:     $v_j \leftarrow w_j / RandomInt(148, 344)$ 
18:     $vol \leftarrow vol + v_j$ 
19:     $N_{\pi_k} \leftarrow N_{\pi_k} \cup \{item_j^{\pi_k}\}$ 
20:  end while
21: end for
22:  $N \leftarrow \bigcup_{0 \leq k \leq K} N_{\pi_k}$ 
```

---

- Dorigo, M., 1992. Optimization, Learning and Natural Algorithms. Ph.D. thesis. Politecnico di Milano.
- Dorigo, M., Maniezzo, V., Colorni, A., 1996. The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics* 26, 29–41.
- Feo, T.A., Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* 8, 67–71.
- Fidanova, S., 2006. Ant Colony Optimization and Multiple Knapsack Problem. volume Chapter 33. J–Ph. Renard editor.
- Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* 13, 533–549.
- Johnson, D.S., Garey, M.R., 1985. A 7160 theorem for bin packing. *Journal of Complexity* 1, 65–106.
- Lopez-Ibanez, M., Dubois-Lacoste, J., Cáceres, L., Birattari, M., Stützle, T., 2016. The irace package: iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3, 43–58.
- Niar, S., Freville, A., 1997. A parallel tabu search algorithm for the 0-1 multidimensional knapsack problem, in: *Proceedings 11th International Parallel Processing Symposium*, pp. 512–516.
- Zhan, S., Wang, L., Zhang, Z., Zhong, Y., 2020. Noising methods with hybrid greedy repair operator for 0-1 knapsack problem. *Memetic Computing* 12, 37–50.