Parallel air palletization and tour planning for simultaneous pickup and delivery

A.C.P. Mesquita, C.A.A. Sanches*

Instituto Tecnológico de Aeronáutica - DCTA/ITA/IEC
Praça Mal. Eduardo Gomes, 50
São José dos Campos - SP - 12.228-900 - Brazil

Abstract

Transport aviation faces risks of cargo imbalance in the set of circumstances of aerial pickup and shipment of goods in a distribution system given the urgency required for loading for fast takeoff and mission accomplishment, notably in moments of crisis, public emergency, client contract short turnaround times, or any external forces for instant takeoff. Additionally, there is no computer support to aid load and trip programmers with the massive transportation demands in each location.

Other dangers are made possible as a result, including poor delivery, excessive fuel consumption, and longer turnaround times than necessary.

We developed an efficient solution by means of computer parallelism to the problem involving scheduling the loading and routing of an airplane in a tour of simultaneous pickup and delivery at intermediate hubs while taking into account a utility score, weight and balance rules, and fuel usage.

This hard problem, named *Air Cargo Load Planning with Routing, Pickup, and Delivery Problem* considers using standardized pallets in fixed positions, obeying the center of gravity constraints, delivering each item to its destination, and minimizing fuel consumption costs.

We also contributed by carrying out multiple experiments with the well-known Ant Colony Optimization on synthetic data based on real data from the *Brazilian Air Force* transportation history and a new procedure to minimize the distance from the next node destined pallet to the ramp door.

We also created a process-based parallel computing heuristic that quickly finds good solutions for a wide range of problem sizes, an essential contribution as it was the unique method that managed to solve all testing scenarios.

Keywords: Air Cargo, Air Palletization, Weight and Balance, Pickup and Delivery, Vehicle Routing, Parallel computing

1. Introduction

Air cargo transport involves several sub-problems that are difficult to solve. Recently, Mesquita and Sanches (2023) modeled and solved the *Air Cargo Load Planning with Routing, Pickup and Delivery* (ACLP+RPDP) composed by four sub-problems: *Build-up Scheduling Problem*, *Air Palletization Problem* (APP), *Weight and Balance Problem* (WBP), and a special case of the *Traveling Salesman Problem*, similar to the proposed by Kaspi et al. (2019), where the profit per unit time is defined as the profit gained after completing the tour divided by the total time (cost in our case) required to complete the tour.

However, there are still other important challenges in air cargo transport that go beyond the definition of the ACLP+RPDP, especially with regard to algorithms performance and the easiness of loading operations at each destination.

Considering air cargo transport, Table 1 lists the main works in the literature and the corresponding sub-problems addressed. We also indicate whether the dimensions of the items were taken into account (**3D** or **2D**) and which solution method was used: heuristics (**Heu**), integers (**Int**), or linear programming (**Lin**).

*Corresponding author.
Email addresses: celio@ita.br (A.C.P. Mesquita), alonso@ita.br (C.A.A. Sanches)

Table 1: Air cargo transport: literature, problems and features

	APP	WBP	SPDP	TSP	2D	3D	Heu	Int	Lin	Parallel heuristics
Larsen and Mikkelsen (1980)	.	★	.	.	.	.	★	.	.	.
Brosh (1981)	.	★	.	.	.	.	.	.	★	.
Ng (1992)	.	★	.	.	.	.	.	★	.	.
Heidelberg et al. (1998)	.	★	.	.	★	.	★	.	.	.
Mongeau and Bes (2003)	★	★	.	.	.	.	.	★	.	.
Fok and Chun (2004)	.	★	.	.	.	.	.	★	.	.
Chan et al. (2006)	★	.	.	.	.	★	★	.	.	.
Kaluzny and Shaw (2009)	.	★	.	.	★	.	.	★	.	.
Verstichel et al. (2011)	.	★	.	.	.	.	.	★	.	.
Mesquita and Cunha (2011)	.	.	★	.	.	.	★	.	.	.
Limbourg et al. (2012)	.	★	.	.	.	.	.	★	.	.
Kaspi et al. (2019)	.	.	.	★	.	.	★	.	.	.
Roesener and Hall (2014)	★	★	.	.	.	★	.	★	.	.
Vancroonenburg et al. (2014)	★	★	.	.	.	.	.	★	.	.
Lurkin and Schyns (2015)	.	★	★	.	.	.	.	★	.	.
Roesener and Barnes (2016)	.	★	.	.	.	.	★	.	.	.
Paquay et al. (2016, 2018)	★	★	.	.	.	★	★	★	.	.
Chenguang et al. (2018)	.	★	.	.	★	.	★	.	.	.
Wong and Ling (2020)	★	★	.	.	.	.	.	★	.	.
Wong et al. (2021)	★	★	.	.	.	.	.	★	.	.
Zhao et al. (2021)	.	★	.	.	.	.	.	★	.	.
Mesquita and Sanches (2023)	★	★	★	★	.	.	★	★	.	.
This work	★	★	★	★	.	.	★	★	.	★

As can be seen, so far Lurkin and Schyns (2015) is the only work that simultaneously addresses an air cargo (WBP) and a flight itinerary (PDP) sub-problem. Although it is innovative, strong simplifications were imposed by the authors: in relation to loading, APP was ignored; with regard to routing, it is assumed a pre-defined flight plan restricted to two legs. It is important to note that these authors consider an aircraft with two doors, and the minimization of loading and unloading costs at the intermediate node was modelled through a container sequencing problem. Referring directly to this work, (Brandt and Nickel, 2019, p. 409) comment: *However, not even these sub-problems are acceptably solved for real-world problem sizes or the models omit some practically relevant constraints.*

There are real situations that are much more complex. In this work, we consider a practical case in Brazil, which is the largest economy in Latin America. Due to its dimensions, this country has the largest air market on the continent with $2,499$ registered airports, of which $1,911$ are private and 588 are public. Although it is an immense distribution network, airlift missions consider 3 to 5 nodes per flight plan. Throughout this work, we address routes with up to 7 nodes, as can be seen in Table 2 and Figure 1.

Table 2: Brazilian airports distances (km)

Node IATA*	l_0 GRU	l_1 GIG	l_2 SSA	l_3 CNF	l_4 CWB	l_5 BSB	l_6 REC
GRU	0	343	1,439	504	358	866	2,114
GIG	343	0	1,218	371	677	935	1,876
SSA	1,439	1,218	0	938	1,788	1,062	676
CNF	504	371	938	0	851	606	1,613
CWB	358	677	1,788	851	0	1,084	2,462
BSB	866	935	1,062	606	1,084	0	1,658
REC	2,114	1,876	676	1,613	2,462	1,658	0

*International Air Transport Association
Source: www.airportdistancecalculator.com

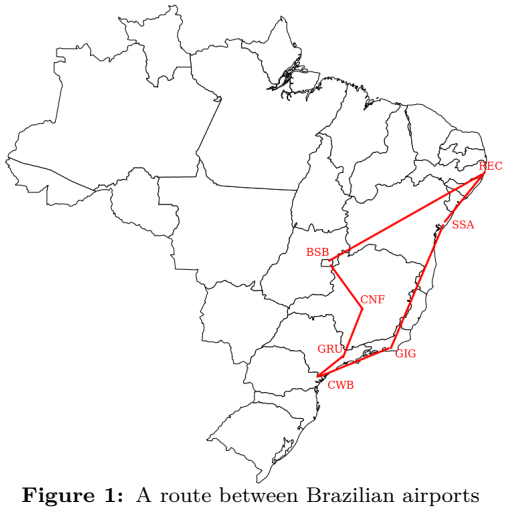

Figure 1: A route between Brazilian airports

In the *Brazilian Air Force* missions, hundreds of items can be carried at each node, where the objectives are to prioritize the transport of the most important items and minimize the cost of fuel along the route. As standardized pallets are used with predefined positions on the aircraft, it is possible to carry out loading and

unloading at each node in around two hours. However, there is no technological assistance that guarantees the achievement of these objectives.

Mesquita and Sanches (2023) proposed a method that attends these objectives: a pallet building and arrangement plan with a routing option that maximizes the benefit-cost ratio for the smooth execution of pickup and delivery transport missions. They develop a heuristic that can be executed on a simple handheld computer (like a laptop or a tablet) and that provides a solution quickly enough to keep this cargo handling time under an hour. This work seeks to improve their results in terms of quality and performance by the use of computer parallelism and by minimizing the distance between the next node destined pallets and the cargo door.

These improvements in their heuristics are meant to reduce the stress that transport planners are subjected to, because they have to deal with a lot of information in planning the aircraft route, assembling the pallets, and picking up and delivering at each node. To the best of our knowledge, that is the first time that an air cargo transport problem that simultaneously involves APP, WBP, PDP and TSP has been addressed, and the first time ACLP+RPDP is solved with computer parallelism.

This article is organized into six more sections. In Section 2, we give a brief review of the literature. In Section 3, we present the problem context and assumptions and, in Section 4, the mathematical model and how we dealt with its issues. In Section 5, we describe the elaborate algorithms, whose results are presented in Section 6. Finally, our conclusions are in Section 7.

2. Related literature

In this section, we briefly describe the characteristics of the main works related to air cargo transport, following the chronological order of Table 1.

Larsen and Mikkelsen (1980) developed an interactive procedure for loading 14 types of Boeing 747 into a two-leg flight plan. Seven types of items were considered to be allocated in 17 to 42 positions. With non-linear programming and heuristics, they present a solution that minimizes positioning changes in the intermediate node, optimizing the load balancing in the aircraft.

Brosh (1981) addressed the problem of planning the allocation of cargo on an aircraft. Considering volume, weight and structural constraints, the author finds the optimal load layout through a fractional programming problem.

Ng (1992) developed a multi-criteria optimization approach to load the *C-130* aircraft of the *Canadian Air Force*. Based on integer programming, this model provides timely planning and improves airlift support for combat operations, solving WBP with pallets in fixed positions, and considering 20 different items.

Heidelberg et al. (1998) developed a heuristic for 2D packing in air loading, comparing it with methods for solving the *Bin Packing Problem*. Authors conclude that the classical algorithms are inadequate in this context, because they ignore the aircraft balancing constraints.

Mongeau and Bes (2003) presented a method based on linear integer programming to solve the problem of choosing and positioning containers on the *Airbus 340-300*. Safety and stability constraints were considered, with the objective of minimizing fuel consumption.

Fok and Chun (2004) developed a web-based application to make efficient use of space and load balancing for an air cargo company. Based on an analysis of historical data, an operational load planning with mathematical optimization is obtained. This container load planning is usually done roughly 2 hours before departure, when all cargo details are in place.

Chan et al. (2006) carried out a case study with heterogeneous pallets. In order to minimise the total cost of shipping, they developed a 3D packing heuristic, with a loading plan for each pallet. Although the authors do not consider load balancing or positioning of pallets in the cargo hold, this method is relevant in commercial and industrial applications, where cargo items tend to be less dense.

Kaluzny and Shaw (2009) developed a mixed integer linear programming model to arrange a set of items in a military context that optimizes the load balance.

Verstichel et al. (2011) solved WBP by selecting the most profitable subset of containers to be loaded onto an aircraft using mixed-integer programming. Experimental results on real-life data showed significant improvements compared to those obtained manually by an experienced planner.

Mesquita and Cunha (2011) presented a heuristic for a real problem of the *Brazilian Air Force*, which consists of defining transport routes with simultaneous collection and delivery from a central distribution terminal.

Limbourg et al. (2012) developed a mixed-integer program for optimally rearranging a set of pallets into a compartmentalized cargo aircraft, specifically the *Boeing 747*.

Kaspi et al. (2019) define and solve a new extension of the TSP to maximize the financial contribution per invested time and present an optimal iterative solution procedure for the problem which converges after a limited number of iterations.

Roesener and Hall (2014) solved APP and WBP as an integer programming problem, which also allows items to be loaded into pallets according to a specific orientation (e.g., this side up).

Vancroonenburg et al. (2014) presented a mixed integer linear programming model that selects the most profitable pallets, satisfying safety and load balancing constraints on the *Boeing 747-400*. Using a solver, authors solved real problems in less than an hour.

As already mentioned, Lurkin and Schyns (2015) was the first work that simultaneously modeled WBP and SPDP in air cargo transport. The authors demonstrated that this problem is NP-hard and performed some experiments with real data, noting that their model offers better results than those obtained manually.

Roesener and Barnes (2016) proposed a heuristic to solve the *Dynamic Airlift Loading Problem* (DALP). Given a set of palletized cargo items that require transport between two nodes in a time frame, the objective of this problem is to select an efficient subset of aircraft, partition the pallets into aircraft loads and assign them to allowable positions on those aircraft.

Paquay et al. (2016) presented a mathematical modeling to optimize the loading of heterogeneous 3D boxes on pallets with a truncated parallelepipeds format. Its objective is to maximise the volume used in containers, considering load balancing constraints, the presence of fragile items and the possibility of rotating these boxes. Paquay et al. (2018) developed some heuristics to solve this problem.

Chenguang et al. (2018) modelled the air transport problem as a 2D packing problem, and presented a heuristic for its optimization in several aircraft, considering load balancing in order to minimise fuel consumption.

Wong and Ling (2020)A developed a mathematical model and a tool based on mixed integer programming for optimizing cargo in aircraft with different pallet configurations. Balance constraints and the presence of dangerous items were considered. Wong et al. (2021) integrated this tool to a digital simulation model, with a visualization and validation system, based on sensors that alert about load deviations.

Zhao et al. (2021) proposed a new modelling for WBP based on mixed integer programming. Instead of focusing on the center of gravity (CG) deviation, the authors consider the original CG envelope of the aircraft, with a linearization method for its non-linear constraints.

Mesquita and Sanches (2023) contributed with a complex model and elaborate heuristics for the ACLP-RPDP that simultaneously solve 4 intractable sub-problems: APP, WBP, SPDP and TSP. They also compare the performances of four well known heuristics: *Ant Colony Optimization*, the *Noising Method Optimization*, the *Greedy Randomized Adaptive Search Procedure*, and *Tabu Search*. They also create a new heuristic called *Shims* which is fast and may be run in a handheld computer. They did not use computer parallelism to improve performance, nor tryed to minimize the distances from the next nodes destined pallets to the cargo ramp door.

As can be seen, except for Mesquita and Sanches (2023), the remaining works do not address air cargo palletization and load balancing with route optimization in a multi-leg flight plan, and none of them employ parallelization in their algorithms. This is the objective of our work: to reshape Mesquita and Sanches (2023) solution process and algorithms to accommodate parallel features that can improve solution quality and performance, and also include a new feature that is to minimize the distances between the next node's destined pallets and the cargo ramp door.

3. Problem context and assumptions

In this section, we describe the context of the problem addressed in this work, as well as the assumptions considered.

3.1. Operational premises

As we are dealing with an extremely complex and diverse problem, we decided to establish some simplifying characteristics:

- At each node of the flight plan, the items to be allocated are characterized by weight, volume, scores, and previously known destinations, but do not have dimensions. We leave the consideration of 2D or 3D items for future work.

- We also disregarded *hazardous* items, which eventually could be treated as high score items and other specific constraints.
- We considered a unique pallet type: the *463L Master Pallet*, a common size platform for bundling and moving air cargo. It is the primary air cargo pallet for more than 70 Air Forces and many air transport companies. This pallet has a capacity of $4500kg$ and $14.8m^3$, is equipped for locking into cargo aircraft rail systems, and includes tie-down rings to secure nets and cargo loads, which in total weighs $140kg$. For more information, see `www.463LPallet.com`.
- All items allocated on a pallet must have the same destination. A pallet which has not yet reached its destination may receive more items, although it is known that these operations of removing restraining nets increase handling time and the risk of improper delivery. We do not consider oversized cargo in this work, but only cargoes that fit on these pallets.
- Pallets with destinations set to the next node should be put as near as possible to the cargo ramp door.
- Finally, as we are interested in minimizing fuel costs while keeping the CG in its operational range, we disregarded some lower ones as handling costs.

Throughout this text, we call a *consolidated item* a set of items of the same destination stacked on a pallet and covered with a restraining net. It is considered unique, having the same attributes of its components, whose values are the sum of individual scores, weights and volumes. See Figure 2. Consolidated items must stay on board until they reach their destination to maintain accuracy in pickup and delivery procedures.

Figure 2: Consolidated items on 463L pallets inside a *Boeing C-17*
Source: From Wikimedia Commons, the free media repository

3.2. Aircraft and load balancing

We consider real scenarios with a smaller or a larger aircraft with payloads of 26,000 kg or 75,000 kg respectively. Both layouts are represented in Figures 3 and 4, where the pallets are identified by p_i.

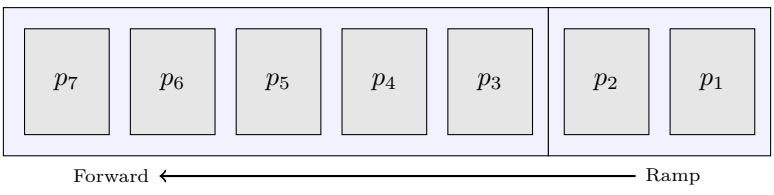

Figure 3: Smaller aircraft layout

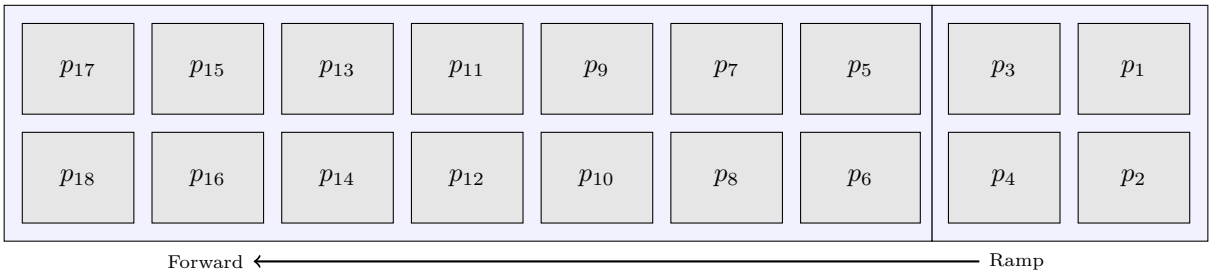

Figure 4: Larger aircraft layout

In both cases, the torque applied to the aircraft must keep its CG in the operational range, which corresponds to a percentage of the *Mean Aerodynamic Chord* [1]: $0.556m$ in the smaller aircraft and $1.17m$ in the larger one. See Figure 5.

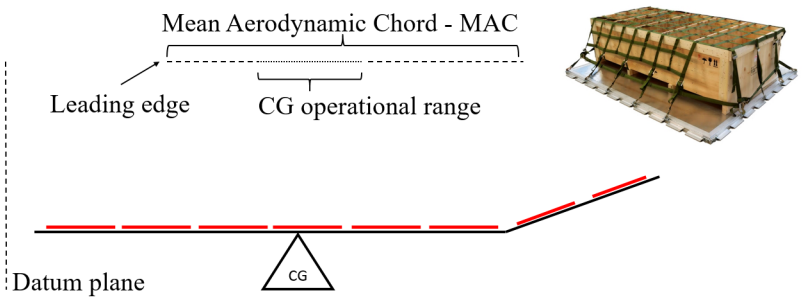

Figure 5: Aircraft longitudinal cut showing red lines as pallets

Tables 3 and 4 show the parameters in both cases. *CGx* and *CGy* refer to the relative distances of pallet centroids (in meters) in relation to the CG of aircraft along both axes. In both aircraft, as the ramps have an inclination of 25 degrees, we made the necessary corrections in *CGx*, *Weight* and *Volume limits* of the corresponding pallets. The monetary costs of both aircraft are also indicated: per unit of distance in flights between legs (c_d) and per deviation in the CG (c_g). It is important to consider that c_g tends to zero as the aircraft attitude tends to level.

Table 3: Smaller aircraft parameters

Limits	*Payload*: $26,000kg$				$limit^{CG}_{long}$: $0.556m$		
Pallets	p_7	p_6	p_5	p_4	p_3	p_2	p_1
CGx (m)	-5.10	-2.70	-0.30	2.10	4.50	6.25	8.39
Weight limits (kg)	4,500	4,500	4,500	4,500	4,500	4,000	3,500
Volume limits (m^3)	13.7	13.7	13.7	13.7	13.7	8.9	6.9
Costs	c_d: US$ 1.100/km				$c_g = 0.05$		

[1] Chord is the distance between the leading and trailing edges of the wing, measured parallel to the normal airflow over the wing (Houghton and Carpenter, 2003, p.18). The average length of the chord is known as the *Mean Aerodynamic Chord* (MAC).

Table 4: Larger aircraft parameters

Limits	*Payload*: $75,000kg$			$limit^{CG}_{long}$: $1.170m$			$limit^{CG}_{lat}$: $0.19m$		
Pallets	p_{17}	p_{15}	p_{13}	p_{11}	p_9	p_7	p_5	p_3	p_1
	p_{18}	p_{16}	p_{14}	p_{12}	p_{10}	p_8	p_6	p_4	p_2
CGx (m)	-17.57	-13.17	-8.77	-4.40	0	4.40	8.77	11.47	14.89
	-17.57	-13.17	-8.77	-4.40	0	4.40	8.77	11.47	14.89
CGy (m)	1.32	1.32	1.32	1.32	1.32	1.32	1.32	1.32	1.32
	-1.32	-1.32	-1.32	-1.32	-1.32	-1.32	-1.32	-1.32	-1.32
Weight limits (kg)	4,500	4,500	4,500	4,500	4,500	4,500	4,500	4,000	3,000
Volume limits (m^3)	14.8	14.8	14.8	14.8	14.8	14.8	14.8	10.0	7.0
Costs	c_d: US$ 4.900/km						$c_g = 0.05$		

We accept the same assumptions as stated by Mesquita and Sanches (2023):

- on each pallet, the items are distributed in such a way that their CG coincides with the centroid of the pallet;
- the CG of the payload must be at a maximum longitudinal distance of $limit^{CG}_{long}$ from the CG of the aircraft;
- in the larger aircraft, the CG of the payload must be at a maximum lateral distance of $limit^{CG}_{lat}$ from the CG of the aircraft;
- in the larger aircraft, pallets are distributed in two identical rows (with odd and even indices, respectively), and their centroids are at a distance d^{CG}_{pallet} from the center-line of the aircraft.

3.3. Problem Summary

Informally, ACLP+RPDP can be summarized as follows:

max	(items score sum) / (tour cost) of picked up and delivered items on a tour
s.t.	Along a tour, the set of unvisited nodes is updated.
	In each node, an item may be included in at most one pallet.
	In each node, consolidated items are composed of items with the same destination.
	Weight, volume and score of a consolidated item are the corresponding sum of their components.
	Consolidated items remain on board until their destinations.
	Consolidated items can only be included in the same pallet if their destinations are the same.
	Only items destined for the remaining nodes can be loaded.
	The lateral and longitudinal torques must be within the operational range of the aircraft.
	Weight and volume limitations of pallets must be respected.
	The total weight must be less than the aircraft payload or the total pallet capacity, whichever is the lowest.
	The pallets destined for the next node of the tour should be put as near as possible to the cargo ramp door (our inclusion).

4. The mathematical modelling

Given the assumptions, scenarios and parameters described in the previous section, we are ready to present the mathematical modelling of ACLP+RPDP, which is one of the contributions of our work.

Let $L = \{l_0, l_1, \ldots, l_K\}$ be the set of $K+1$ nodes (or destinations), where l_0 is the origin and end of a flight plan. Let $d(l_i, l_j)$ be the distance from l_i to l_j, where $0 \leq i, j \leq K$. By definition, $d(l_i, l_i) = 0$. Let L_k be the set of remaining nodes when the aircraft is in l_k, $0 \leq k \leq K$. Therefore, $L_0 = L$ and $L_K = \{l_0\}$.

Let $C = \{c_{ij}\}$ be the cost matrix of flights, where $c_{ij} = c_d * d(l_i, l_j), 0 \leq i, j \leq K$.

Let $S_K = \{s : \{1, \ldots, K\} \rightarrow \{1, \ldots, K\}\}$ be the set of $K!$ permutations, which correspond to all possible tours (or itineraries) that have l_0 as origin and end, passing through the others K nodes.

Let $M = \{p_1, p_2, \ldots, p_m\}$ the set of m pallets. Each pallet p_i, $1 \leq i \leq m$, has weight capacity $p_i.w$, volume capacity $p_i.v$, pallet destinations $p_i.to[k]$, $0 \leq k \leq K$, and distance to the CG of aircraft $p_i.d$. $p_i.to[k]$ denotes that pallet p_i may assume a different destination in each node k.

Let $N_k = \{t_1^k, t_2^k, \ldots, t_{n_k}^k\}$ be the set of n_k items to be loaded in node l_k, $0 \leq k \leq K$. Each item t_j^k, $1 \leq j \leq n_k$, has score $t_j^k.s$, weight $t_j^k.w$, volume $t_j^k.v$, and destination $t_j^k.to \in L_k$. Let $N = \bigcup_{0 \leq k \leq K} N_k$ be the set of items of all nodes along a tour.

Let $Q_k = \{a_1^k, a_2^k, \ldots, a_{m_k}^k\}$ be the set of consolidated items loaded in $m_k \leq m$ pallets when the aircraft arrives at node l_k, with $0 \leq k \leq K$. a_i^k, $1 \leq i \leq m_k$, is the group of picked-up items that were allocated on pallet p_i in some of the previous nodes. a_i^k has total weight $a_i^k.w$, total volume $a_i^k.v$, and destination $a_i^k.to \in L_k \cup \{l_k\}$. If $a_i^k.to = l_k$, then a_i^k is unloaded, and p_i will be available for reloading; otherwise, a_i^k remains on the aircraft, eventually in another pallet, and with items of N_k having the same destination.

Let X_{ij}^k and Y_{iq}^k be binary variables, where $0 \leq k \leq K$, $1 \leq j \leq n_k$, $1 \leq i \leq m$ and $1 \leq q \leq m_k$. $X_{ij}^k = 1$ if t_j^k is assigned to p_i in node l_k, and 0 otherwise. $Y_{iq}^k = 1$ if a_q^k is assigned to p_i in node l_k, and 0 otherwise. By definition, $Y_{iq}^0 = 0$. Allocations of items or consolidated items to pallets in node l_k can be seen as a bipartite graph $G_k(V_k, E_k)$, where $V_k = M \cup N_k \cup Q_k$, $E_k = E_k^N \cup E_k^Q$, $(p_i, t_j^k) \in E_k^N$ if $X_{ij}^k = 1$, and $(p_i, a_q^k) \in E_k^Q$ if $Y_{iq}^k = 1$.

The mathematical modeling of this problem is described in the equations below.

$$\max_{\pi \in S_K} f_\pi(\tilde{s}, \tilde{c}) \tag{1}$$

$$\tilde{s} = \sum_{k=0}^{K} \sum_{i=1}^{m} \sum_{j=1}^{n_k} X_{ij}^k \times t_j^k.s \tag{2}$$

$$\tilde{c} = c_{0,\pi(1)} \times (1 + c_g \times |\epsilon_0|) + \sum_{k=1}^{K-1} [c_{\pi(k),\pi(k+1)} \times (1 + c_g \times |\epsilon_k|)] + c_{\pi(K),0} \times (1 + c_g \times |\epsilon_K|) \tag{3}$$

$$maxW = \min(Payload, \sum_{i=1}^{m} p_i.w) \tag{4}$$

$$\tau_k = \sum_{i=1}^{m} [p_i.d \times (\sum_{j=1}^{n_k} X_{ij}^k \times t_j^k.w + \sum_{q=1}^{m} Y_{iq}^k \times a_q^k.w)];\ k \in \{0, 1, \ldots, K\} \tag{5}$$

$$\epsilon_k = \frac{\tau_k}{maxW \times limit_{long}^{CG}};\ k \in \{0, 1, \ldots, K\} \tag{6}$$

$$L_0 = L; L_k = L_{k-1} - \{l_{\pi(k)}\};\ k \in \{1, 2, \ldots, K\} \tag{7}$$

$$Y_{iq}^0 = 0; a_i^0.w = 0; a_i^0.v = 0; a_i^0.to = -1;\ i, q \in \{1, 2, \ldots, m\} \tag{8}$$

$$X_{ij}^k = 0 \text{ if } t_j^k.to \notin L_k;\ i \in \{1, 2, \ldots, m\};\ j \in \{1, 2, \ldots, n_k\};\ k \in \{1, 2, \ldots, K\} \tag{9}$$

$$Y_{iq}^k = 0 \text{ if } a_i^k.to \notin L_k;\ i \in \{1, 2, \ldots, m\};\ q \in \{1, 2, \ldots, m_k\};\ k \in \{1, 2, \ldots, K\} \tag{10}$$

$$a_i^{\pi(k+1)}.w = \sum_{j=1}^{n_k} X_{ij}^{\pi(k)} \times t_j^{\pi(k)}.w + \sum_{q=1}^{m_k} Y_{iq}^{\pi(k)} \times a_q^{\pi(k)}.w;\ i \in \{1, 2, \ldots, m_k\}; k \in \{0, 1, \ldots, K-1\} \tag{11}$$

$$a_i^{\pi(k+1)}.v = \sum_{j=1}^{n_k} X_{ij}^{\pi(k)} \times t_j^{\pi(k)}.v + \sum_{q=1}^{m_k} Y_{iq}^{\pi(k)} \times a_q^{\pi(k)}.v;\ i \in \{1, 2, \ldots, m_k\}; k \in \{0, 1, \ldots, K-1\} \tag{12}$$

$$a_i^{\pi(k+1)}.to = t_j^{\pi(k)}.to \text{ if } X_{ij}^{\pi(k)} = 1 \text{ and } t_j^{\pi(k)}.to \in L_{k+1};\ i \in \{1,2,\ldots,m_k\};\ j \in \{1,2,\ldots,n_k\};\ k \in \{0,1,\ldots,K-1\} \tag{13}$$

$$LatIt_k = \sum_{i=1}^{m}\sum_{j=1}^{n_k}(X_{ij}^k \times t_j^k.w \times (i\%2) - X_{ij}^k \times t_j^k.w \times (i+1)\%2);\ k \in \{0,1,\ldots,K\} \tag{14}$$

$$LatCons_k = \sum_{i=1}^{m}\sum_{q=1}^{m_k}(Y_{iq}^k \times a_q^k.w \times (i\%2) - Y_{iq}^k \times a_q^k.w \times (i+1)\%2);\ k \in \{0,1,\ldots,K\} \tag{15}$$

$$s.t.: d_{pallet}^{CG} \times |LatIt_k + LatCons_k| \leq \sum_{i=1}^{m} p_i.w \times limit_{lat}^{CG};\ k \in \{0,1,\ldots,K\} \tag{16}$$

$$s.t.: |\tau_k| \leq maxW \times limit_{long}^{CG};\ k \in \{0,1,\ldots,K\} \tag{17}$$

$$s.t.: \sum_{i=1}^{m}(\sum_{j=1}^{n_k} X_{ij}^k \times t_j^k.w + \sum_{q=1}^{m_k} Y_{iq}^k \times a_q^k.w) \leq maxW;\ k \in \{0,1,\ldots,K\} \tag{18}$$

$$s.t.: \sum_{j=1}^{n_k} X_{ij}^k \times t_j^k.w + \sum_{q=1}^{m_k} Y_{iq}^k \times a_q^k.w \leq p_i.w;\ i \in \{1,2,\ldots,m_k\};\ k \in \{0,1,\ldots,K\} \tag{19}$$

$$s.t.: \sum_{j=1}^{n_k} X_{ij}^k \times t_j^k.v + \sum_{q=1}^{m_k} Y_{iq}^k \times a_q^k.v \leq\ p_i.v;\ i \in \{1,2,\ldots,m_k\};\ k \in \{0,1,\ldots,K\} \tag{20}$$

$$s.t.: \sum_{i=1}^{m} X_{ij}^k \leq 1;\ j \in \{1,2,\ldots,n_k\};\ k \in \{0,1,\ldots,K\} \tag{21}$$

$$s.t.: Y_{iq}^k = 1 \text{ if } a_q^k.to \in L_k;\ q \in \{1,2,\ldots,m_k\};\ k \in \{0,1,\ldots,K\} \tag{22}$$

$$s.t.: p_i.to[k] = t_j^k.to \text{ if } X_{ij}^k = 1;\ i \in \{1,2,\ldots,m\};\ j \in \{1,2,\ldots,n_k\};\ k \in \{1,2,\ldots,K\} \tag{23}$$

$$s.t.: p_i.to[k] = a_q^k.to \text{ if } Y_{iq}^k = 1;\ i \in \{1,2,\ldots,m\};\ q \in \{1,2,\ldots,m_k\};\ k \in \{1,2,\ldots,K\} \tag{24}$$

The objective of this problem is to find a permutation $\pi \in S_K$ that maximizes the function $f_\pi(\tilde{s},\tilde{c})$ 1. In this way, the flight plan will be $l_0, l_{\pi(1)}, \ldots, l_{\pi(K)}, l_0$. $\tilde{s}$ is the total score of transported items 2 and $\tilde{c}$ is the total cost of fuel consumed 3. As can be seen, $\tilde{c}$ corresponds to the fuel consumption due to the flights carried out and the CG deviation of the transported cargo. Throughout this work, for simplicity, we use $f = \tilde{s}/\tilde{c}$.

The maximum load will be the minimum between the payload and the capacity supported by the pallets 4. Considering the maximum longitudinal distance allowed for the CG, all torques 5 and deviations 6 are calculated.

For each step of the flight plan, the set of unvisited nodes is updated 7. Although there are no items consolidated at the beginning of the flight plan, we defined these variables for ease of notation 8. Items destined outside the rest of the flight plan will not be loaded (9 and 10).

Consolidated items appear when there are items on the pallets that will not be unloaded on the next node. Its weights 11 and volumes 12 correspond to all the items that were on the pallet, since all these items have the same destination. On subsequent nodes, consolidated items can be allocated with other items of same destination 13.

Equations 14 and 15 respectively, are applied only to the larger aircraft, and calculate the lateral torques of items and consolidated items loaded in both rows of pallets, whose constraint is described in 16. Similarly, 17 is the longitudinal torque constraint, which is applied to both aircraft sizes.

The weight limitation of the aircraft must be respected 18. The sum of weights 19 and volumes 20 in each pallet must not exceed its capacity. Each item is associated with a pallet at most 21.

Consolidated items remain on board 22 until their destinations. At each node, an item (23) and a consolidated item (24) must only be allocated to a pallet if the destinations are the same.

5. Resolution strategy

Once the assumptions of this work and the mathematical modelling of the problem are presented, it is easy to see that ACLP+RPDP is NP-hard. In a similar way to (Lurkin and Schyns, 2015, p. 6), consider the simple case where $K = 1$ (one leg), $m = 2$ (two pallets around the aircraft CG), $2n$ sufficiently light items with same scores in l_0, and no items in l_1. Under these conditions, through polynomial reductions for the *Set-Partition Problem*, it is possible to demonstrate that the decision problem associated with ACLP+RPDP is NP-complete.

Real cases are more complex as they have hundreds of different items in each node and involve three intractable sub-problems: APP, WBP and SPDP. Through the mathematical modelling presented in the previous section, we verify that *Mixed-Integer Programming* (MIP) is not able to solve these cases in feasible time. Thus, it is necessary to adopt some strategy to find a viable solution, not necessarily optimal, that seeks to maximize the objective function f.

Mesquita and Sanches (2023) strategy is based on the fact that, in real cases, K is usually small. Specifically, they will consider $K \leq 6$ throughout their and our work, which is a higher value than usual in *Brazilian Air Force* missions. As a result, if we have fast node-by-node solutions that allow us to construct a complete tour, we will be able to test all possible $K!$ tours and thus select the one that provides the best value for the f function.

The tactic will be, at each shipping node, to predefine the destinations of the pallets at that node. In this way, we will reserve a number of pallets proportional to the volume demanded by 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. Once the destinations of the pallets are defined, we will use serial and Multi-processing heuristics to find the best possible node-by-node solutions. This strategy is summarized in Algorithm 1, retrieved from Mesquita and Sanches (2023).

Algorithm 1 Solves ACLP+RPDP for a scenario with certain volume surplus (1.2, 1.5, or 2.0)

1: **procedure** $ACLP + RPDP(scenario,\ surplus,\ numProcs,\ timeLim)$
2: Let L, M, C be according to $scenario$
3: $N \leftarrow ItemsGeneration(scenario,\ surplus)$
4: **for** each $method$
5: **for** each $\pi \in S_K$
6: $f_\pi \leftarrow SolveTour(\pi, L, M, C, N, method, numProcs,\ timeLim)$
7: $answer[scenario, surplus, method] \leftarrow \max f$
8: **return** $answer$

In this algorithm, there are six values for the *scenario* parameter, according to Table 5, which defines K, the sets of nodes, the aircraft, the pallets and the costs from Tables 3 or 4 that will be used (line 2).

The other parameter *volume* is a value greater than 1, which corresponds, at each node l_k, to the ratio between the sum of the volumes of the items ($\sum_{j=1}^{n_k} t_j^k.v$) and the load capacity of the pallets ($\sum_{i=1}^{m} p_i.v$). This parameter is passed to *ItemsGeneration* (line 3), responsible for creating the items to be shipped, which will be presented in the next section.

method corresponds to one of the heuristics that we will present in subsection 5.2. The loop of lines 5-6 goes through all permutations π, where the node-by-node resolutions are performed by *SolveTour*, whose result is stored in f_π. The best result among all $K!$ tours will be the answer for *scenario*, *volume* and *method* (line 7).

Table 5: Testing scenarios retrieved from Mesquita and Sanches (2023).

Scenario	K	L	**Aircraft**
1	2	$\{l_0, l_1, l_2\}$	smaller
2	2	$\{l_0, l_1, l_2\}$	larger
3	3	$\{l_0, l_1, l_2, l_3\}$	larger
4	4	$\{l_0, l_1, l_2, l_3, l_4\}$	larger
5	5	$\{l_0, l_1, l_2, l_3, l_4, l_5\}$	larger
6	6	$\{l_0, l_1, l_2, l_3, l_4, l_5, l_6\}$	larger

Next, we will present two subsections: in the first we explain how *SolveTour* is executed, while in the second we will present the heuristics developed for node-by-node resolutions.

5.1. SolveTour algorithm

In addition to the set of nodes, pallets, costs and items, *SolveTour*, described in Algorithm 2, receives the parameter *method*, which corresponds to a heuristic for solving the node-by-node problems, and the parameter π, which is a permutation that defines the order of visits in this tour.

As we mentioned in the previous section, all tours start and end at l_0 (lines 2-3). After initializing the score and cost values (lines 4-5), there is a loop for the $K+1$ flights (lines 6-20). Initially we set pallets destination as -1 (line 8). When the aircraft is at node l_0, the initial graph G_1 is empty because it has no consolidated items 11. Otherwise, the set L_k of remaining nodes is updated (line 13), and *UpdateConsolidated* (line 14) returns the set of consolidated items that have not yet reached their destination and remain on board, rearranging them on the pallets to minimize CG deviation. This allocation is stored in graph G_1 (line 15).

In the context of this work, we know that $m > K$, once the aircraft has 7 or 18 pallets and $K \leq 6$, allowing there to be at least one pallet for each node to be visited. *SetPalletsDestination* (line 16) presets the destination of each pallet based on the volume demands of the current node, without changing the pallets destination with consolidated items.

Finally, *SolveNode* includes the edges corresponding to the items shipped at the current node, returning the graph G_2 (line 17). The score and the CG deviation of this graph are calculated (line 18) and accumulated (lines 19-20), allowing the final result of this tour (line 21).

Algorithm 2 Solves the sequence of nodes in the tour indexed by π

1: **procedure** $SolveTour(\pi, L, M, C, N, method, numProcs,\ timeLim)$
2: $\quad\pi(0) \leftarrow 0$
3: $\quad\pi(K+1) \leftarrow 0$
4: $\quad score \leftarrow 0$
5: $\quad cost \leftarrow 0$
6: $\quad$**for** $k \leftarrow 0$ to K
7: $\quad\quad$**for** $i \leftarrow 1$ to m
8: $\quad\quad\quad p_i.to[k] \leftarrow -1$
9: $\quad\quad$**if** $k = 0$
10: $\quad\quad\quad L_0 \leftarrow L$
11: $\quad\quad\quad$Let $G_1(M \cup N_0, \varnothing)$
12: $\quad\quad$**else**
13: $\quad\quad\quad L_k \leftarrow L_k - \pi(k)$
14: $\quad\quad\quad Q_{\pi(k)}, E^Q_{\pi(k)}, M \leftarrow UpdateConsolidated(\pi(k))$
15: $\quad\quad\quad$Let $G_1(M \cup N_{\pi(k)} \cup Q_{\pi(k)}, E^Q_{\pi(k)})$
16: $\quad\quad M \leftarrow SetPalletsDestination(\pi(k))$
17: $\quad\quad G_2 \leftarrow SolveNode(method, \pi(k), G_1, numProcs,\ timeLim)$
18: $\quad\quad s, \epsilon \leftarrow ScoreAndDeviation(\pi(k), G_2)$
19: $\quad\quad score \leftarrow score + s$
20: $\quad\quad cost \leftarrow cost + c_{\pi(k),\pi(k+1)} * (1 + c_g * |\epsilon|)$
21: $\quad$**return** $score/cost$

UpdateConsolidated finds the best allocation for the consolidated items that remain on board. Initially, the set Q is created, with the consolidated items that did not reach their destination.

Then *MinCGDeviation* is run through a MIP solver to relocate the consolidated items on the pallets minimizing torque and ensuring that they all remain on board, one on each pallet. As there are few variables, the MIP solver returns an allocation E_k^Q very quickly.

SetPalletsDestination sets pallets destination not yet defined. The total volume of items and the number of pallets with consolidated items destined for each node. The destinations of each pallet are defined proportionally to the volume of items, regarding the pallets with consolidated items.

ScoreAndDeviation evaluates the allocation graph generated by *SolveNode*, returning the corresponding score and CG deviation. It consists of a loop that goes through all the pallets, accumulating the scores and the torques of the shipped items, allowing the final calculation of the CG deviation.

UpdateConsolidated, *MinCGDeviation*, *SetPalletsDestination*, and *ScoreAndDeviation* are thoroughly described by Mesquita and Sanches (2023).

5.2. Node-by-node solutions

In this subsection we present implementations of *SolveNode*(...), where *method* corresponds to a heuristic, k is the index of the current node and G is the allocation graph of the consolidated items that remain on board at node l_k.

An important issue on meta-heuristics is that they are known for the hard work in tuning parameters. For this, when we use the word *empirically* throughout the text, we want to say that we accomplished many tests with the hardest instances to fit these values. This does not guarantee best results for all instances, but, on average, overall results are acceptable. Another important observation is that our *ACO* implementations were designed based on traditional ideas from the literature Fidanova (2006) and on the experiences learned during tests, plus the special features to process-based parallelism.

To guarantee that performance comparisons are made correctly, our implementations use the same elements described below.

Data structure:	Items, pallets and edges are arrays of *Python* class objects, which are initialized with the same data loaded from text files.
Stopping criterion:	All methods have the same time limit of 0.7 seconds per node.
Common procedures:	All methods have the same procedures for: solution printing, pallets destinations and initial parameters setting, solution integrity checking, and the same procedures for selecting and inserting edges.

The selection of edges for E_k^N uses the *edge attractiveness* θ_{ij}^k, Equation 25, which can be understood as the tendency to allocate the item t_j^k to the pallet p_i. It is directly proportional to the score, and inversely proportional to the volume and the torque of each item. The 3000 value was empirically set to keep the attractiveness in the range [0, 1], to be in the same order of magnitude of the Ant Colony Optimization pheromone range.

$$\theta_{ij}^k = \frac{t_j^k.s}{t_j^k.v \times 3000} \times \left(1 - \frac{t_j^k.w \times |p_i.d|}{\max_s\{t_s^k.w\} \times \max_q\{|p_q.d|\}}\right);\ i \in \{1, 2, \ldots, m\},\ j \in \{1, 2, \ldots, n_k\} \tag{25}$$

We use four types of random selections:

- *RandomReal*(r_1, r_2): randomly selects a real number in $[r_1, r_2]$;
- *RandomInt*(i_1, i_2): randomly selects a integer number in $[i_1, i_2]$;
- *Roulette*(set) biased through ϕ: selects an element from *set*, where the probability of each element is proportional to the value of a given function ϕ defined on *set*.

To compare allocation graphs generated by the heuristics, we choose the one that offer a higher associated score, since the CG deviation is limited by the constraints.

In the pseudo-codes presented below, given an allocation graph G for the node l_k, $f_s(G)$ is equivalent to the score s returned by *ScoreAndDeviation*(k, G), described in Mesquita and Sanches (2023).

Below, we present 2 implementations for *SolveNode*: *Multi-process Ant Colony Optimization* (*mpACO*) and a new parallel heuristic that we propose called *Multi-process Shims (mpShims)*.

According to (Manfrin et al., 2006, p.226), there are five topologies for multi-process interchange of information: *fully-connected*, where the master process broadcasts to all remaining child processes; *replace-worst*, where the best-so-far solution process broadcasts only to the current worst solution process; *hypercube*, where processes are connected as a hypercube, and a vertex process broadcasts only to the connected vertices; *ring*, in which one process only sends a message to the next process connected to it; and *parallel independent runs (PIR)*, in which there are no communication costs and the best solution is chosen among all processes. They concluded that the PIR topology presented, in average, the smallest distance from the optimal solutions.

It is important to say that the parallel ACO (*pACO*) is a different implementation where employs threads instead of processes to solve each ant solution in parallel (for a detailed information on parallel ACO see Pedemonte et al. (2011)). We chose *mpACO* for our approach because we employed the *Fully-Connected* topology which requires to share the pheromone array between the multi-process ants and the facility to handle the possibility of race condition.

5.2.1. Multi-process Ant Colony Optimization (mpACO)

In ACO Dorigo (1992), a population of A ants performs independent search (sequential or in parallel), where each ant a finds its own allocation graph G^a. Each edge $e_{ij}^k = (p_i, t_j^k)$ has a general attractiveness $ga_{ij}^k = (\tau_{ij}^k)^\alpha + (\theta_{ij}^k)^\beta$, where τ_{ij}^k is the pheromone of this edge, and α and β are constants. If e_{ij}^k is included in G^a, ant a increases the pheromone τ_{ij}^k.

Algorithm 3 presents this heuristic in its parallel multi-process version as we have implemented.

$SolveNode(mpACO, ...)$ is the main process, while $AntSolve(...)$ is executed the child concurrent processes.

Algorithm 3 Solves a node with the *mpACO* method

1: **procedure** $SolveNode(mpACO, k, G, numProcs,\ timeLim)$
2: Let $G(V_k, E_k^Q)$ ▷ This is a partial solution with only consolidated on board
3: $G^*(V_k, E_k^{N*} \cup E_k^Q), \_ \leftarrow Greedy(k, G, 0.9)$ ▷ This partial solution is completed by a greedy algorithm
4: $G^+(V_k, E_k^{N+} \cup E_k^Q) \leftarrow G^*(V_k, E_k^{N*} \cup E_k^Q)$
5: $A \leftarrow numProcs$
6: $ga_{ij}^k \leftarrow 0,\ 1 \leq i \leq m,\ 1 \leq j \leq n_k$
7: $\tau_{ij}^k \leftarrow 1,\ 1 \leq i \leq m,\ 1 \leq j \leq n_k$
8: $\alpha \leftarrow 1.0$
9: $\beta \leftarrow 4.0$
10: $\rho \leftarrow 0.2$
11: $stagnant \leftarrow 0$
12: **while** $stagnant < 3$
13: $AntsQueue \leftarrow \{\ \}$
14: **for** 1 **to** A
15: $AntSolve(G^+,\ AntsQueue,\ ga_{ij}^k)$
16: $counter \leftarrow A$
17: **while** $counter > 0$
18: $G^a \leftarrow \varnothing$
19: **while** $runtime < timeLim$ ▷ Wait until there is a solution in the queue
20: $G^a \leftarrow AntsQueue.Pop()$
21: **if** $G^a \neq \varnothing$
22: **break**
23: **if** $f_s(G^a) > f_s(G^+)$
24: $G^+(V_k, E_k^{N+} \cup E_k^Q) \leftarrow G^a(V_k, E_k^{Na} \cup E_k^Q)$
25: $counter \leftarrow counter - 1$
26: **if** $f_s(G^+) > f_s(G^*)$
27: $stagnant \leftarrow 0$
28: $G^*(V_k, E_k^{N*} \cup E_k^Q) \leftarrow G^+(V_k, E_k^{N+} \cup E_k^Q)$
29: **else**
30: $stagnant \leftarrow stagnant + 1$
31: **for all** τ_{ij}^k
32: $\tau_{ij}^k \leftarrow (1 - \rho)\tau_{ij}^k$
33: $\tau_{ij}^k \leftarrow \tau_{ij}^k + \Delta\tau$
34: $ga_{ij}^k \leftarrow (\tau_{ij}^k)^\alpha + (\theta_{ij}^k)^\beta$
35: **return** $G^*(V_k,\ E_k^{N*} \cup E_k^Q)$

A greedy solution is obtained in line 3, being initially stored in G^* and G^+. The argument 0.9 generates an initial and poor solution for ACO to improve. The second return value denoted by underscore is not used by ACO.

The number A of ants was set to the number of parallel processes (line 5).

The $m \times n_k$ matrices τ_{ij}^k and ga_{ij}^k (lines 6-7) are used and updated by all ants. For further explanation, see Dorigo et al. (1996).

The number of iterations depends on the time limit or stagnation in improving results (line 12). At the beginning os each iteration (13) a queue of ants (*AntsQueue*) is created to hold ants parallel solutions generated in the child processes.

The local solution G^+ is sent to A child processes (14) almost simultaneously. While there are any ant working on an solution or the run-time is still less then maximum (17), it hangs in the *AntsQueue* until there is a solution available.

If there is a global improvement (26) the best so far solution ($G^*(V_k, E_k^{N*} \cup E_k^Q)$) is updated. At the end of each iteration pheromone levels (32) are evaporated, than updated with the new calculated $\Delta\tau$ (line 33), and according to Equation 26, and the general attractiveness (34) is also updated.

$$\Delta\tau = \frac{f_s(G^+) - f_s(G^*)}{A \times f_s(G^*)} \tag{26}$$

Many ACO applications update pheromone levels and the general attractiveness only when there is a solution improvement. Differently, this work updates in both cases, exploring also the negative learning approach proposed by Nurcahyadi and Blum (2021). We did not compare these performances because it is out of the scope of this work.

We also developed Algorithm 4, which generates a greedy allocation of the items available in node l_k, according to the non-ascending order of θ_{ij}^k, and considering the consolidated items already shipped (E_k^Q). Edges are included in this allocation as long as they respect feasibility constraints. Furthermore, the volume of each pallet p_i cannot exceed $p_i.v \times limit$, where $0 < limit \leq 1$ is a given parameter.

Algorithm 4 Mount a greedy solution until the volume limit for each pallet

```
1:  procedure Greedy(k, G, limit)
2:      Let G(V_k, E_k^Q)
3:      η_1 ← {0} × m
4:      volume ← {0} × m
5:      E_k^N ← ∅
6:      for q ← 1 to m
7:          if (p_i, a_q^k) ∈ E_k^Q
8:              volume[q] ← volume[q] + a_q^k.v
9:      for each e_ij^k in non-ascending order of θ_ij^k
10:         if (E_k^N ∪ {e_ij^k} is feasible) and (volume[i] ≤ p_i.v × limit)
11:             E_k^N ← E_k^N ∪ {e_ij^k}
12:             volume[i] ← volume[i] + t_j^k.v
13:             η_1[i] ← η_1[i] + 1
14:     return G(V_k, E_k^N ∪ E_k^Q), η_1        ▷ return the solution and the last item indexes array
```

In Algorithm 4, line 3 defines an array with the same size as pallets array to save the set of last edges inserted in the greedy solution. The set η_1 is returned together with the greedy solution.

Algorithm 5 Each ant solves a node in parallel

```
1:  procedure AntSolve(G^+, AntsQueue, ga_ij^k))
2:      G^a(V_k, E_k^{Na} ∪ E_k^Q) ← G^+(V_k, E_k^{N+} ∪ E_k^Q)
3:      E ← {(p_i, t_j^k) | (p_i, t_j^k) ∉ E_k^{Na}}
4:      while |E| > 0
5:          e ← Roulette(E) biased through ga_ij^k
6:          E ← E − {e}
7:          if E_k^{Na} ∪ {e} is feasible
8:              E_k^{Na} ← E_k^{Na} ∪ {e}
9:      AntsQueue ← AntsQueue ∪ G^a(V_k, E_k^{Na} ∪ E_k^Q)
```

Algorithm 5, which will be executed by a child parallel process, receives an initial and poor solution $G^+(V_k, E_k^{N+} \cup E_k^Q)$ generated greedily to be completed and improved by *mpACO*.

Line 3 expresses the creation of the neighborhood, the set of candidate edges.

While the neighborhood is not empty (line 4), choose an edge (line 5), delete it from the neighborhood and include it in the solution, if feasible (line 8).

Finally, put the solution in the queue (line 9) to be retrieved by the main process.

5.2.2. Multi-process Shims - mpShims

Finally, we present a new heuristic designed specifically for ACLP+RPDP, which we named *mpShims*. Like in mechanics, shims are collections of spacers to fill gaps, which may be composed of parts with different

thicknesses (see Figure 6). This strategy is based on a practical observation: usually, subsets of smaller and lighter items are saved for later adjustments to the remaining clearances.

Figure 6: Shims of various thicknesses
Source: www.mscdirect.com/product/details/70475967

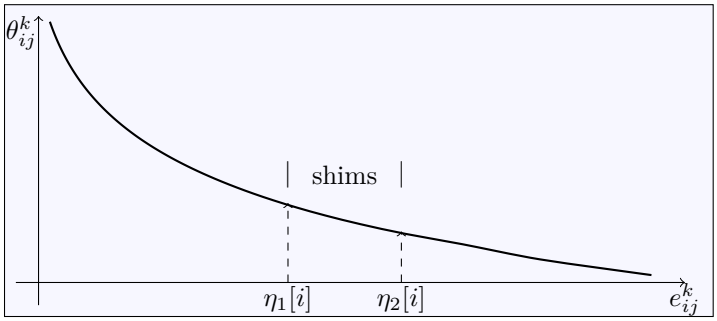

Figure 7: n_k edges e_{ij}^k of p_i sorted by θ_{ij}^k in non-ascending order

Figure 7 represents the n_k possible edges e_{ij}^k of p_i sorted by θ_{ij}^k. *mpShims* starts with a greedy solution, stopping at the edge with index in $\eta_1[i]$ close to the local optimum (first phase). Then, considering even the edge with the index $\eta_2[i]$, it elaborates different possible complements for this pallet (second phase) and selects the best of these complements (third phase).

Algorithm 6 Solve a node with *mpShims*

```
1:  procedure SolveNode(mpShims, k, G, numProcs, timeLim)
2:      G* ← G(V_k, E_k^Q)
3:      Sort M by |p_i.d| in non-descending order
4:      E_k^N ← {}
5:      Let E[1..m][1..n_k], where E[i] is an array with n_k edges (p_i, t_j^k) sorted by θ_ij^k in non-ascending order,
        1 ≤ i ≤ m
6:      limits ← getLimits(0.75, 0.98, numProcs)
7:      shimsQueue ← {}
8:      for lim ∈ limits
9:          shimsSolve(G(V_k, E_k^Q), M, shimsQueue, lim, E)
10:     counter ← |limits|
11:     while counter > 0
12:         G ← ∅
13:         while runtime < timeLim                    ▷ Wait until there is a solution in the queue
14:             G ← shimsQueue.Pop()
15:             if G ≠ ∅
16:                 break
17:         if f_s(G) > f_s(G*)
18:             G* ← G
19:         counter ← counter − 1
20:     return G*(V_k, E_k^N ∪ E_k^Q)
```

Algorithm 6 describes the *SolveNode* method when it receives *mpShims* as argument.

Line 2 defines the best solution so far which is formed by a semi-loaded aircraft where pallets have been unloaded and some remain on board (E_k^Q) with consolidated items.

The Shims solution starts with the set of pallets closer to the center of gravity (3).

A new empty set of edges is created in line 4.

Edges with items in node "k" to be loaded into not full pallets are defined in line 5.

Line 6 creates a list of unique limits with values between 0.75 and 0.98, that will be sent to different parallel processes to generate diverse solutions.

A queue of shims solutions to be used by the child processes to send solution to the main process is defined in line 7.

The iteration in 9 publishes or triggers parallel processes to solve $G(V_k, E_k^Q)$.

In line **??** the program hangs until each child process put a solution into the queue. Each solution is collected by the main process and compared with the best so far.

Finally, in line 20, the best solution found is returned.

Algorithm 7 Shims solving method to be run in parallel

1: **procedure** $shimsSolve(G,\ M,\ shimsQueue,\ limit,\ E)$
2: $G(V_k,\ E_k^Q \cup E_k^N),\ \eta_1 \leftarrow Greedy(k,\ G(V_k,\ E_k^Q),\ limit)$
3: **for** $i \leftarrow 1$ to m
4: $\eta_2[i] \leftarrow \eta_1[i]$
5: $volume \leftarrow 0$
6: **repeat**
7: $\eta_2[i] \leftarrow \eta_2[i] + 1$
8: $e_{ij}^k \leftarrow E[i][\eta_2[i]]$
9: $volume \leftarrow volume + t_j^k.v$
10: **until** $(\eta_2[i] = n_k)$ **or** $(volume \geq (2 - limit) * p_i.v))$
11: $slack[i] \leftarrow (1 - limit) \times p_i.v$
12: $E_k^N \leftarrow E_k^N \cup getBestShims(i,\ \eta_1,\ \eta_2,\ E,\ k,\ slack[i])$
13: $shimsQueue \leftarrow shimsQueue \cup \{G(V_k, E_k^N \cup E_k^Q)\}$

Algorithm 7, line 2 corresponds to Shims' first phase. Lines 6-10 correspond to the Shims' second phase. Line 12 corresponds to Shims' third phase. In the last line, 13, *mpShims* puts the solution into the queue to be retrieved by the main process.

Algorithm 8 Get the best shims of edges that fills each pallet gap

1: **procedure** $getBestShims(i,\ \eta_1,\ \eta_2,\ E,\ k,\ slack)$
2: $volume \leftarrow 0$
3: $b \leftarrow 1$
4: $shims[b] \leftarrow \{\}$
5: $Set \leftarrow Set \cup \{shims[b]\}$
6: **for** $x \leftarrow \eta_1[i]$ to $\eta_2[i]$
7: $NewShims \leftarrow$ **True**
8: $e_{ij}^k \leftarrow E[i][x]$
9: **for** $shims \in Set$
10: **if** $e_{ij}^k \notin (E_k^N \cup shims)$ **and** e_{ij}^k is feasible **and** $(t_j^k.v + volume) \leq slack[i]$
11: $shims \leftarrow shims \cup \{e_{ij}^k\}$
12: $volume \leftarrow volume + t_j^k.v$
13: $NewShims \leftarrow$ **False**
14: **break**
15: **if** $NewShims$
16: $volume \leftarrow 0$
17: $b \leftarrow b + 1$
18: $shims[b] \leftarrow \{\}$
19: $shims[b] \leftarrow shims[b] \cup \{e_{ij}^k\}$
20: $Set \leftarrow Set \cup \{shims[b]\}$
21: $sh_w \leftarrow shims$, where $shims \in Set$ and $\sum_{e_{ab}^k \in shims} t_b^k.w$ is maximum
22: $sh_v \leftarrow shims$, where $shims \in Set$ and $\sum_{e_{ab}^k \in shims} t_b^k.v$ is maximum
23: $sh_{best} \leftarrow shims$, where $shims \in \{sh_w, sh_v\}$ and $\sum_{e_{ab}^k \in x} t_b^k.s$ is maximum
24: **return** sh_{best}

Algorithm 8 describes the procedure do get the best Shims, the best subset of edges to fill each pallet gap. This algorithm follows the logic of the *First-Fit Decreasing* algorithm as described by Johnson and Garey (1985).

Line 4 creates the first empty shims. Line 5 creates the first set of shims from where the best shims will be chosen.

Line 6 iterate in the edges indexes range to find subsets of shims that fit into each pallet gap.

Initially, new shims creation is permitted (7), but it will only be created if the last edge could not be included in any shims from the set (15).

For each edge, iterate in all sets in a try to include it in any of the previous shims (9). As the last edge was included, forbid a new shims creation (13).

Finally, select the best weight shims (21), the best volume shims (22), and the best score shims (23) between these 2. The best score shims edges will be returned.

6. Implementation and results

As we are dealing with a new problem, which has been modeled by Mesquita and Sanches (2023), we use the same benchmarks of their work, which are based on the characteristics of real airlifts carried out by the *Brazilian Air Force*.

We test our methods with 126 different instances: six operational scenarios with 2 aircraft sizes and 3 to 7 nodes; and seven randomly generated item sets for three volume surpluses (1.2, 1.5, and 2.0 times aircraft volume capacities).

6.1. Results obtained

The final experiments are performed on a 64-bit, 128GiB, 3.6GHz, 48-core processor, 2 threads per core, Intel Xeon, with *Linux CentOS 8* as the operational system and *Python 3.8.2* as the programming language.

For parallelization, we used the *Python Multi-processing* package, which implements *process-based parallelism* or *parallel shared memory algorithm*. According to **?**, *Multi-processing* is a package that supports spawning processes using an API similar to the *Python Threading* package.

According to (Breshears, 2009, p.271), a *Process* is the operating system's spawned and controlled entity that encapsulates an executing application. A process has two main jobs: the first is to hold the application's resources, and the second is to carry out the application's instructions.

It is known that working concurrently opens up synchronization issues. But the *Multi-processing* package (mp) offers both local and remote concurrency, effectively side-stepping the global interpreter lock by using sub-processes instead of threads. Because of this, the *Multi-processing* module lets the programmer take full advantage of the fact that a machine has more than one processor, normally capable of 2 threads each.

By Amdahl's law, there is an optimal number of parallel executions for each problem in each environment. While working at IBM in 1967, Gene Amdahl developed the foundation for what became known as Amdahl's Law or Amdahl's Argument. Essentially, the law states that while a process can be decomposed into steps that may then be run in parallel, the time taken for the whole process will be significantly limited by the steps that remain serialized. So, we decided to run some tests to find the best number of parallel processes for each of this work's operational scenarios.

For each red bullet closest to the upper left corner, we call it *Utopia*, an impossible result with zero time duration and a possible best score. The blue bullets are experiments performed with the number of parallel processes indicated. The shortest distance from *Utopia* may reveal the most efficient parallel processes for the scenario (see Table 5).

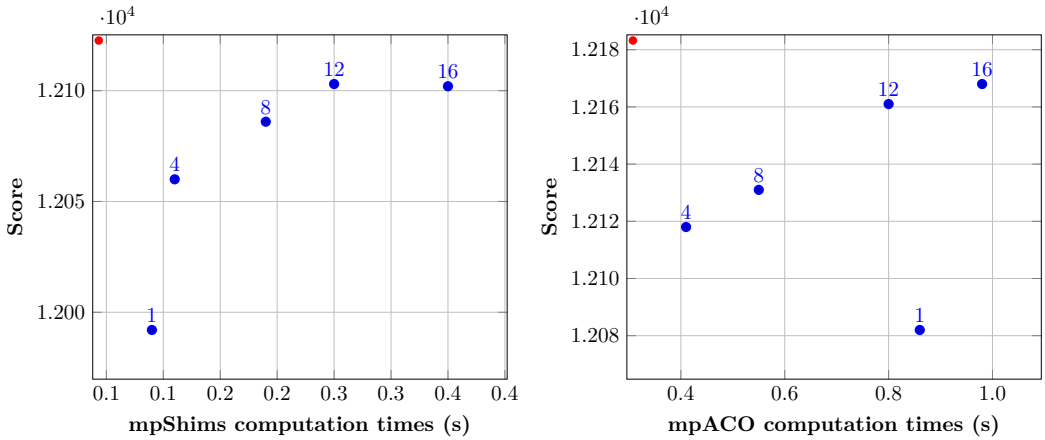

Figure 8: Scenario 1 performances with $1.2 \times volume$.

Observing Figure 8, it may be noticed that the best number of processes for *Scenario 1* solved with *mpShims* (the closest to the *Utopia*) is **6**. Solved with *mpACO*, the best number is **8**. Table 6 presents all methods, volumes percentages and scenarios numbers of best parallel processes to be run to solve all tours.

To build Table 6, it was necessary to determine the number of parallel processes that best solve each node. For this we created a procedure as in Algorithm 9.

Algorithm 9 Get the best number of parallel processes

procedure $getBestNPP(scores,\ runtimes,\ procs)$
  $maxS \leftarrow max(scores)$
  $minS \leftarrow min(scores)$
  $maxR \leftarrow max(runtimes)$
  $minR \leftarrow min(runtimes)$
  $bestNPP \leftarrow 0$, best number of parallel processes
  $minD \leftarrow 999999999.9$
  for $i \in |scores|$
    $y = (maxS - scores[i])\ /\ (maxS - minS)$ ▷ the Y axis leg
    $x = (runtimes[i] - minR)\ /\ (maxR - minR)$ ▷ the X axis leg
    $h = \sqrt{x^2 + y^2}$ ▷ distance from *Utopia*
    if $minD > h$
      $minD \leftarrow h$
      $bestNPP \leftarrow procs[i]$
  return $bestNPP$

Table 6: $bestNPP$ found for each scenario

		Scenarios					
method	*volume*	**1**	**2**	**3**	**4**	**5**	**6**
mpShims	1.2	6	4	4	4	2	2
	1.5	4	4	4	4	4	4
	2.0	4	4	4	2	2	4
mpACO	1.2	10	8	10	6	10	10
	1.5	10	6	12	8	10	8
	2.0	10	8	2	10	6	6

Table 6 presents the best numbers of parallel processes for each method, scenario, and volume surplus. As all numbers are less than or equal to 12, any hand-held computer with 8 cores can yield efficient solutions, as normally each core may be capable of 2 threads.

To find these values we submitted to test 1, 2, 4, 6, 8, 10, 12, 14, 16 parallel processes.

Table 7: Volume surpluses, methods and scenarios results

			Scenarios						Normalized
Surplus	*method*	**Results**	**1**	**2**	**3**	**4**	**5**	**6**	**Speed-up**
1.2	$mpShims^{0.7s}$	f	5.70	3.09	2.94	2.90	2.79	5.70	0.91
		run-time (s)	1	3	7	26	97	267	70
	$mpACO^{60s}$	f	6.42	3.78	3.19	3.17	2.97	5.87	1.00
		run-time (s)	16	282	314	1334	6945	18883	1
	$mpACO^{0.7s}$	f							
		run-time (s)							
1.5	$mpShims^{0.7s}$	f	8.15	4.22	4.40	4.01	3.51	7.08	
		run-time (s)	1	5	12	45	244	737	
	$mpACO^{60s}$	f	9.50	5.01	4.60	4.28	3.78		
		run-time (s)	9	344	787	1525	7525		
	$mpACO^{0.7s}$	f							
		run-time (s)							
2.0	$mpShims^{0.7s}$	f	11.31	6.29	5.54	5.83	5.05	10.63	
		run-time (s)	2	14	34	106	477	2417	
	$mpACO^{60s}$	f							
		run-time (s)							
	$mpACO^{0.7s}$	f							
		run-time (s)							

The numbers 0.7s and 60s as superscript on the methods names represent the time limit per node.

7. Conclusions

xxxxx

Acknowledgments

This research was partially supported by *São Paulo Research Foundation* (FAPESP, grant 2016/01860-1).

References

Brandt, F., Nickel, S., 2019. The air cargo load planning problem - a consolidated problem definition and literature review on related problems. European Journal of Operational Research 275, 399–410.

Breshears, C., 2009. Chapter 8: A Thread Monkey's Guide to Writing Parallel Applications. Kindle 1st Edition.

Brosh, I., 1981. Optimal cargo allocation on board a plane: a sequential linear programming approach. European Journal of Operational Research 8, 40–46.

Chan, F., Bhagwat, R., Kumar, N., Tiwari, M., Lam, P., 2006. Development of a decision support system for air-cargo pallets loading problem: A case study. Expert Systems with Applications 31, 472–485.

Chenguang, Y., Hu, L., Yuan, G., 2018. Load planning of transport aircraft based on hybrid genetic algorithm. MATEC Web of Conferences 179, 1–6.

Dorigo, M., 1992. Optimization, Learning and Natural Algorithms. Phd thesis. Politecnico di Milano. Milano, Italia.

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.

Fidanova, S., 2006. Ant Colony Optimization and Multiple Knapsack Problem. volume Chapter 33. J–Ph. Renard editor.

Fok, K., Chun, A., 2004. Optimizing air cargo load planning and analysis, in: Proceedings of the International Conference on Computing, Communications and Control Technologies.

Heidelberg, K.R., Parnell, G.S., Ames, J.E., 1998. Automated air load planning. Naval Research Logistics 45, 751–768.

Houghton, E.L., Carpenter, P.W., 2003. Aerodynamics for Engineering Students (5th ed.). Butterworth Heinmann.

Johnson, D.S., Garey, M.R., 1985. A 7160 theorem for bin packing. Journal of Complexity 1, 65–106. doi:10.1016/0885-064X(85)90022-6.

Kaluzny, B.L., Shaw, R.H.A.D., 2009. Optimal aircraft load balancing. International Transactions in Operational Research 16, 767–787.

Kaspi, M., Zofi, M., Teller, R., 2019. Maximizing the profit per unit time for the travelling salesman problem. Computers & Industrial Engineering 135, 702–710. URL: https://www.sciencedirect.com/science/article/pii/S0360835219303808, doi:https://doi.org/10.1016/j.cie.2019.06.050.

Larsen, O., Mikkelsen, G., 1980. An interactive system for the loading of cargo aircraft. European Journal of Operational Research 4, 367–373.

Limbourg, S., Schyns, M., Laporte, G., 2012. Automatic aircraft cargo load planning. Journal of the Operational Research Society 63, 1271–1283.

Lurkin, V., Schyns, M., 2015. The airline container loading problem with pickup and delivery. European Journal of Operational Research 244(3), 955–965.

Manfrin, M., Birattari, M., Stützle, T., Dorigo, M., 2006. Parallel Ant Colony for the Traveling Salesman Problem. Springer.

Mesquita, A.C.P., Cunha, C.B., 2011. An integrated heuristic based on the Scatter Search metaheuristic for vehicle routing problems with simultaneous delivery and pickup in the context of the Brazilian Air Force. Transportes 19, 33–42.

Mesquita, A.C.P., Sanches, C.A.A., 2023. Air cargo load and route planning in pickup and delivery operations. European Journal of Operations Research .

Mongeau, M., Bes, C., 2003. Optimization of aircraft container loading. IEEE Transaction on Aerospace and Electronic Systems 39, 140–150.

Ng, K.Y.K., 1992. A multicriteria optimization approach to aircraft loading. Operations Research 40, 1200–1205.

Nurcahyadi, T., Blum, C., 2021. Adding negative learning to ant colony optimization: A comprehensive study. Mathematics 9. URL: https://www.mdpi.com/2227-7390/9/4/361, doi:10.3390/math9040361.

Paquay, C., Limbourg, S., Schyns, M., Oliveira, J.F., 2018. MIP-based constructive heuristics for the three-dimensional Bin Packing Problem with transportation constraints. International Journal of Production Research 56, 1581–1592.

Paquay, C., Schyns, M., Limbourg, S., 2016. A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. International Transactions in Operational Research 23, 187–213.

Pedemonte, M., Nesmachnow, S., Cancela, H., 2011. A survey on parallel ant colony optimization. Applied soft computing 11, 5181–5197.

Python, 2022. Process-based parallelism. URL: https://docs.python.org/3/library/multiprocessing.html.

Roesener, A., Barnes, J., 2016. An advanced tabu search approach to the dynamic airlift loading problem. Logistics Research 9(1), 1–18.

Roesener, A., Hall, S., 2014. A nonlinear integer programming formulation for the airlift loading problem with insufficient aircraft. Journal of Nonlinear Analysis and Optimization: Theory and Applications 5, 125–141.

Vancroonenburg, W., Verstichel, J., Tavernier, K., Vanden Berghe, G., 2014. Automatic air cargo selection and weight balancing: A mixed integer programming approach. Transportation Research Part E 65, 70–83.

Verstichel, J., Vancroonenburg, W., Souffriau, W., Berghe, G.V., 2011. A mixed integer programming approach to the aircraft weight and balance problem. Procedia Social and Behavioral Sciences 20, 1051–1059.

Wong, E.Y., Ling, K.K.T., 2020. A mixed integer programming approach to air cargo load planning with multiple aircraft configurations and dangerous goods, in: 7th International Conference on Frontiers of Industrial Engineering (ICFIE), pp. 123–130. doi:10.1109/ICFIE50845.2020.9266727.

Wong, E.Y.C., Mo, D.Y., So, S., 2021. Closed-loop digital twin system for air cargo load planning operations. International Journal of Computer Integrated Manufacturing 34, 801–813. doi:10.1080/0951192X.2020.1775299.

Zhao, X., Yuan, Y., Dong, Y., Zhao, R., 2021. Optimization approach to the aircraft weight and balance problem with the centre of gravity envelope constraints. IET Intelligent Transport Systems 15, 1269–1286.