The results obtained with $tmax = 3600s$, which is the highest tested runtime limit, are in Tables 6, 7 and 8, with *surplus* values 1.2, 1.5 and 2.0, respectively. We indicate with an **x** the cases where *Gurobi* did not find a feasible solution within this runtime limit, or had to be aborted due to high RAM consumption.

Table 6: Solutions with $surplus = 1.2$ and $tmax = 3600s$

Tested tours	*method*	*scenario*	**1**	**2**	**3**	**4**	**5**	**Normalized Speed-up**
Shortest	*Gurobi*	f	8.52	11.70	13.07	13.54	12.44	1.00
		time (s)	17	17	16	17	26	1.0
	Shims	f	8.54	11.63	12.97	13.43	12.36	0.99
		time (s)	1	1	2	2	2	11.6
All $K!$	*Gurobi*	f	8.52	12.25	13.32	14.61	x	1.00
		time (s)	25	36	121	297	x	1.0
	Shims	f	8.54	12.26	13.23	14.53	13.42	0.99
		time (s)	1	2	6	17	106	18.4

Table 7: Solutions with $surplus = 1.5$ and $tmax = 3600s$

Tested tours	*method*	*scenario*	**1**	**2**	**3**	**4**	**5**	**Normalized Speed-up**
Shortest	*Gurobi*	f	11.80	16.74	18.04	18.86	16.92	1.00
		time (s)	38	30	27	29	81	1.00
	Shims	f	11.83	16.74	18.01	18.89	16.91	1.00
		time (s)	1	2	2	3	3	20.5
All $K!$	*Gurobi*	f	11.81	17.01	18.25	20.59	18.36	1.00
		time (s)	53	61	193	470	2241	1.0
	Shims	f	11.83	17.00	18.21	20.45	18.00	0.99
		time (s)	1	2	9	26	157	15.5

Table 8: Solutions with $surplus = 2.0$ and $tmax = 3600s$

Tested tours	*method*	*scenario*	**1**	**2**	**3**	**4**	**5**	**Normalized Speed-up**
Shortest	*Gurobi*	f	17.75	24.39	26.46	27.09	24.00	1.00
		time (s)	162	95	74	68	67	1.0
	Shims	f	17.78	24.40	26.36	27.01	23.94	0.99
		time (s)	2	3	3	4	5	27.4
All $K!$	*Gurobi*	f	17.75	25.24	26.46	29.11	x	1.00
		time (s)	151	123	319	689	x	1.0
	Shims	f	17.78	25.25	26.38	28.98	26.08	1.00
		time (s)	2	3	13	37	229	23.3

From these data, we can make some conclusions:

- The strategy of testing all $K!$ often gives a better quality solution.
- *Gurobi* fails in some cases, when $scenario = 5$ and the strategy is to check all $K!$ tours. This occurs because the runtime limit per node is smaller, and there tends to be more packed contents on the aircraft, reducing the space for allocating items and making the solution difficult.
- When *Gurobi* finishes, it finds the best solution, but the one obtained by *Shims* reaches 99% of that value.
- *Shims* always finds a solution, being 11 to 27 times faster.
- All runtimes are much lower than the limit because the solution on many nodes can be fast. Anyway, in all the tests performed, the maximum time spent by *Shims* did not reach 4 minutes, that is, it is practically instantaneous. On the other hand, when $scenario = 5$ and $surplus = 1.5$, Gurobi spent almost 40 minutes.

Table 9 shows the results obtained with the strategy of testing the $K!$ tours in all scenarios with different runtime limits. We can observe more cases where *Gurobi* fails, even in smaller scenarios. When *Gurobi* finishes, *Shims* finds quickly a solution of similar quality (98% or better). In all cases, *Shims* finds a good solution in less than 4 minutes.

Table 9: Solutions testing all $K!$ tours with different runtime limits

		surplus	1.2					1.5					2.0				
method	*tmax* **(s)**	*scenario*	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
Gurobi	240	f	8.53	12.24	13.34	x	x	11.81	17.02	18.31	x	x	17.75	25.24	x	x	x
		time (s)	41	43	123	x	x	50	62	181	x	x	166	140	x	x	x
	1200	f	8.52	12.25	13.31	14.67	x	11.81	17.02	18.25	20.64	x	17.75	25.24	26.49	27.97	x
		time (s)	39	41	129	217	x	52	61	190	304	x	161	139	384	579	x
	2400	f	8.53	12.25	13.30	14.59	13.59	11.80	17.02	18.25	20.60	x	17.75	25.24	26.46	29.10	x
		time (s)	33	39	127	313	1444	51	62	188	327	x	158	127	381	543	x
	3600	f	8.52	12.25	13.32	14.61	x	11.81	17.01	18.25	20.59	18.36	17.75	25.24	26.46	29.11	x
		time (s)	25	36	121	297	x	53	61	193	470	2241	151	123	319	689	x
Shims	240	f	8.54	12.26	13.23	14.53	13.42	11.83	17.00	18.21	20.45	18.00	17.78	25.25	26.38	28.98	26.08
		time (s)	1	2	6	17	109	1	2	8	26	160	2	3	13	37	229

Most of *Gurobi* executions consumed over 12 GB of RAM memory. Considering that the computer used in the experiments has a standard RAM consumption when idle of 3.5 GB (with *Python Interface Development Environment*, a PDF viewer, a system monitor, and a LaTeX IDE simultaneously open), the actual RAM consumption of *Gurobi* was over 8.5 GB. On the other hand, all *Shims* executions consumed at most 1.5 GB of RAM memory.

7. Conclusions

In this work, we modeled and solved a real air transport problem, named *Air Cargo Load Planning with Routing, Pickup and Delivery Problem* (ACLP+RPDP). For the first time in the literature, a *NP-hard* problem that involves *simultaneously* pallet assembly, load balancing and route planning is addressed, where the cost-effectiveness of transport is maximized. We adopted some simplifications that are not critical, but that allowed the unprecedented solution of this problem considering more than two nodes.

We consider that, in practical cases, the number K of nodes, excluding the base, is small ($K \leq 6$), each of them with hundreds of items to be shipped. Thus, considering a real aircraft, we have developed some node-by-node solutions, which are not necessarily optimal. This allows us to test two strategies: find a solution using the shortest tours, or enumerate all $K!$ tours and choose the best. The complete process can be executed quickly on a simple handheld computer, offering good results and reducing stress for the transport planners. As validation, we performed tests in several scenarios with real data from the *Brazilian Air Force*. At the moment, there is no commercial software available for this problem.

We have developed a solution process that, in less than 4 minutes, can establish a flight plan for a single aircraft with a good distribution of load on pallets to be put in the cargo bay, enforcing the loaded aircraft balance, maximizing the total score and minimizing fuel consumption. The output, which includes the tour plan and the pallet building and arrangement plan, is an essential part of airlift: it improves flight safety, makes ground operations more efficient, and makes sure that each item gets to its right destination. In this way, at each node of the tour, the time required is restricted exclusively to handling the airport equipment.

Our main contributions were the mathematical modeling of ACLP+RPDP that involves four sub-problems *NP-hard*, a complete process to solve it on a simple handheld computer, and a new heuristic that offers fast node solutions with good quality. Finally, by focusing on the node-to-node solution, the method of this work is not exclusive to aircraft and airports: it can be adapted, for example, to ships and ports, or vehicles and warehouses, or wagons and railways, provided that their practical cases are similar to those considered here. In these situations, it would be necessary to make some changes in the modeling: for example, modify the load balancing constraints, and consider the available space in vehicles or wagons instead of pallets.

As this is an ongoing research, we thought about some possible future improvements:

- consider a aircraft fleet, rather than a single one;
- model three-dimensional items;
- discard the highest cost tours, to reduce the runtime;
- implement parallel algorithms in some steps of the solution.