

# Loading aircrafts for military operations

Christelle Guéret<sup>1,2</sup>, Narendra Jussien<sup>1</sup>, Olivier Lhomme<sup>1,3</sup>,  
Claire Pavageau<sup>1</sup> and Christian Prins<sup>4</sup>

<sup>1</sup> École des Mines de Nantes – BP 220722  
F-44307 Nantes Cedex 3, France  
{gueret, jussien, cpava}@emn.fr

<sup>2</sup> also at IRCCyN  
(Institut de Recherche en Communications et Cybernétique de Nantes)

<sup>3</sup> now at Ilog, SA  
1681 route des Dolines  
F-06560 Valbonne, France  
olhomme@ilog.fr

<sup>4</sup> Université de Technologie de Troyes – BP 2060  
F-10010 Troyes Cedex, France  
prins@univ-troyes.fr

## 1 Introduction

In military context, **projection of forces** means fast and massive transportation of military equipment (vehicles, troops, ammunitions, ...) from a set of bases to a set of destinations, with the objective of minimizing both the total transportation time and the number of required vectors (aircrafts, boats, trains, ...). The most time-critical projections use air transportation.

Force projections (either at wartime or for humanitarian operations) are usually scheduled by military logisticians. For prospective studies, the French military agency in charge of prospective studies (the *Délégation Générale pour l'Armement* – DGA) may need to simulate projections at any time: to evaluate a new aircraft, to find compatible dimensions for a new vehicle, ...

DGA is currently developing automated tools to tackle those complex and combinatorial operations. In order to reduce the complexity of the task, they decided to cut the problem into two sub-problems: an aircraft loading problem (assigning items to be shipped by aircrafts) and a flight scheduling and routing problem.

We present here the loading software that we developed for DGA. The aim is to speed up the tedious computation of loading patterns (typically one day by hand for an average size problem). The program should be fast enough to allow a human expert to test various scenarios within a very short period of time.

In this paper, we detail our method which quickly computes an initial heuristic solution before applying local search heuristics. Results obtained on real data sets are presented.

## 2 Problem description

### 2.1 Data

#### 2.1.1 Aircrafts and items

The basic information for this problem is composed of two sets: a set of aircraft types and a set of items to be shipped. An aircraft type is characterized by its rectangular bay (depth, width and height) and its maximal payload. An item is characterized by its dimensions (depth, width, height and weight) and its route (departure air base and destination).

#### 2.1.2 Troops

Troops are considered as special *items* because they cannot be located anywhere in the aircraft bay. For safety reasons, they must be seated on fixed seats in the cabin or on removable benches in the bay. Some aircrafts do not contain fixed seats nor removable benches. A bench is characterized by its position and the number of pax<sup>1</sup> it can handle. Its capacity depends on the type of transported troops (*e.g.* a paratrooper needs more room than a tank driver because of its backpack).

#### 2.1.3 Air bases

Air bases have their own fleet which is not shared with other departure bases. Therefore, each air base and its related items can be tackled separately. Note that in our problem, neither fuel restriction nor crew regulations have to be taken into account. Indeed, in war time, crew regulations are ignored and fuel *will* be available (so say operational officers).

### 2.2 Constraints and objective function

The aim of the problem is to minimize the total number of sorties for all aircrafts, subject to the following constraints:

---

<sup>1</sup>pax means people in air transportation.

- **Aircraft-related constraints**

- An aircraft delivers only one destination at a time. Thus items to different destinations need to fly on separate planes.
- The number of sorties of each aircraft is limited by lower and upper bounds. This constraint is used to perform prospective studies on the composition of the fleet of each base.
- An aircraft can only carry items which are compatible with it. For example, international regulations (IATA) forbid transporting ammunitions on civilian aircrafts<sup>2</sup>.

- **Item-related constraints**

- Items cannot be stacked up in the aircraft<sup>3</sup>.
- Some items must be loaded in the same plane (*e.g.* a tank driver and its tank, otherwise nobody will be able to get the tank out of the plane in the destination base).
- Some items cannot be loaded in the same aircraft (*e.g.* troops and ammunitions).
- Some items may be rotated (*e.g.* pallets), others have a unique orientation (*e.g.* consider turning a tank inside a bay !).

This problem is characterized by a small number of item types (but there are a lot of each item). It can be viewed as a kind of bi-dimensional (recall that items cannot be stacked up in a bay) bin-packing with heterogeneous bins (the aircrafts) and many side constraints. It is clearly an NP-hard problem.

### 3 An example

Here is a real size example. For confidentiality reasons, the data have been modified.

Consider the problem in which the items given in Table 1 must be transported from a given base to three destinations A, B and C. Items are described in Table 2. Note that items I2, I3, and I6 must be transported with one soldier. In this example, there are two types of soldiers (*S1* and *S2*), and items *I1* and soldiers (*S1* and *S2*) cannot be transported in the same aircraft.

The fleet associated to the selected base is composed of 2 aircrafts of type A1, 4 of type A2 and 44 of type A3. The characteristics of these aircrafts are given in Table 3. A2 aircrafts can only transport items I1 and/or items I5. A1 aircrafts contain 185 fixed seats in the cabin and no removable benches. A3 aircrafts contain three removable benches:

---

<sup>2</sup>Armies often charter civil aircrafts for punctual use.

<sup>3</sup>The problem is bi-dimensional. The heights of items and bays are only given to know if an item will fit in a bay.

	Destination A	Destination B	Destination C
Soldiers S1	122	201	371
Soldiers S2	100	380	0
Item I1	14	27	3
Item I2	14	26	-
Item I3	51	100	-
Item I4	2	-	-
Item I5	32	-	-
Item I6	9	20	-
Item I7	5	-	-

Table 1: List of items to transport

Items	Length	Width	Height	Weight	Spec. Req.
S1	-	-	-	0.135	-
S2	-	-	-	0.250	-
I1	2.64	2.74	2.43	4.00	-
I2	4.50	1.72	2.00	2.50	1 S1
I3	6.45	2.07	2.00	4.50	1 S1
I4	16.10	3.04	2.70	5.00	-
I5	10.00	3.50	2.50	25.00	-
I6	7.80	2.40	2.00	8.00	1 S1
I7	14.60	3.00	3.76	6.00	-

Table 2: Item characteristics

- two of length 23.10 and width 41.00 are placed on each side of the bay. The capacity is 24 soldiers  $S1$  and 15 soldiers  $S2$  for both.
- one of length 23.10 and width 1.16 is placed in the middle of the bay. Its capacity is 72 soldiers  $S1$  and 50 soldiers  $S2$ .

Finally, Table 4 gives the lower and upper bounds of the number of sorties for each aircraft. Note that the global bounds are not necessarily the sum of the destination bounds: other constraints may apply (this is the case for aircraft A3 in the example). In Table 4, when no figure is given, no aircraft of the considered type can transport items to the considered destination.

## 4 Fast computation of a first solution

The first step of our method consists in quickly computing a feasible solution. We developed two sets of heuristics for that purpose: a list-based heuristic and a loading pattern generation method.

### 4.1 List-based heuristic

The list heuristic sorts all the items according to a given criterion, and takes them one at a time to place them in a suitable aircraft. Such a heuristic is therefore composed of three phases:

1. sorting the items;
2. selecting for each item a suitable aircraft;
3. placing the items into the aircraft's bay.

#### 4.1.1 Sorting the items

Several sorting criteria come to mind for sorting the items. Like in the classical FFD (First Fit Decreasing) or BFD (Best Fit Decreasing) bin-packing heuristics, it seems wise to load cumbersome items first. Thus, we tried several orders: decreasing weights, decreasing areas, ...

Computational experiments showed that the best results are obtained when sorting the items in decreasing order of their *weight*  $\times$  *area* value.

The item-related constraint stating that some items need to be transported with others is handled in this phase by using *macro-items*: a *fititious* item that groups together all the items to be shipped in the same aircraft. This macro-item has a weight equal to the sum of the weights of the grouped items. When its dimensions have to be used, the macro-item is automatically expanded into the real items.

### 4.1.2 Selecting an aircraft for each item

After the sorting phase, the list heuristic selects each item in the ordered list and affects it to an aircraft. It is possible to either load one aircraft at a time or several ones at the same time (equivalent to *sequential* and *parallel* methods in Vehicle Routing Problems). These two options were tested but the latter gave much better results. The algorithm works as follows:

- The initial set of empty aircrafts is given by the lower bounds on the number of sorties.

In our example (see Section 3), the initial set of empty aircrafts only contains 3 aircrafts of type A1 to destination C, respectively 38 and 39 aircrafts of type A3 to destinations A and B. We add to this set 3 aircrafts of type A3 without preassigned destination (because the global lower bound on the number of aircrafts of this type is 80 whereas the sum of lower bounds for each destination is only  $38+39=77$ ).

- Once these aircrafts are full (or more precisely once the current item in the list does not fit in any of the aircrafts), another set of aircrafts is added. This set contains  $\min(UB_i - n_i, F_i)$  aircrafts of type  $i$  where  $UB_i$  is the global upper bound for aircrafts of type  $A_i$ ,  $n_i$  the number of aircrafts of type  $A_i$  selected so far, and  $F_i$  the number of aircraft of this type in the fleet.

In our example, the second set of aircrafts considered is composed of  $\min(4 - 3, 2) = 1$  aircrafts of type A1 to destination C,  $\min(8 - 0, 4) = 4$  aircrafts of type A2 to destination A, and  $\min(93 - 80, 44) = 13$  aircrafts of type A3 which will be shared between destinations A and B.

- That generation is repeated until all items are loaded or until upper bounds are reached for all aircraft types.

Each item is loaded into one aircraft of the current set in which it can fit. The selected aircraft is chosen according to a given criterion (*e.g.* the most loaded aircraft, the aircraft with the most number of items, ...).

### 4.1.3 Placing items into an aircraft

In order to make sure that an item fits into its aircraft, we developed a *checking* procedure. This is where *exotic* constraints are handled: incompatibility constraints, macro-items, ... Clearly, fitting a set of  $n$  items into an aircraft bay is a difficult problem. We thus chose to focus on effective heuristics to solve that problem. We first tried a bottom-left strategy [2] but this heuristic does not relocate the items already in the bay. Thus it can deny some insertions which would be possible by relocating items.

Thus, we chose to develop a constraint-based approach for solving that placement problem. Constraint-based programming uses variables with an assigned domain (set of possible values for the variables) upon which constraints are declared [4]. Solving such a set of constraints is done through an enumeration process helped with constraint filtering (setting aside parts of the search space proved to lead to non solutions).

Once *exotic* constraints are verified, an actual positioning of the items needs to be computed. Our variables will be the position of each item (*i.e.* the coordinates of its bottom-left corner in the bay). The aim is to find a position for each item such that none of them overlap in the bay.

Without loss of generality, we can only consider positions in which items are bottom-left stacked when considering the bay as an oriented plane. Obviously, no all single point in the bay will be an admissible position of items. One can only consider position that are sums of subsets of item dimensions. Therefore, two *subset-sum problems* are solved (one for each dimension of the bay: length and width).

A subset-sum problem is a particular case of the 0-1 knapsack problem: given a set  $S = \{a_1, a_2, \dots, a_n\}$  of  $n$  integers and another integer  $B$  (the knapsack capacity), find a subset  $Y$  of  $S$  whose cumulative sum is maximal without exceeding  $B$ .

This problem is binary NP-Hard, but can be solved efficiently in practice by a dynamic programming method in  $O(n.B)$  [3]. This method has an interesting property: for the same complexity, it obtains as by-products all sum values which can be achieved by subsets of  $S$ .

For example, consider the subset-sum problem composed of the  $n$  lengths of the items (without forgetting the width of the items that can turn), with capacity  $L$  where  $L$  is the maximum length of the bays. Solving this subset-sum problem will provide all the possible x-coordinates.

Possible positions for each item are deduced from those two subset-sum problems by taking an x-coordinate and a y-coordinate in the two answers. After removing non compatibles ones, the remaining set of positions becomes the *domain* of an item.

The search process is quite simple, variables are dynamically ordered regarding the current size of their domain (choosing first highly constrained variables: the well known *first-fail principle*) and enumerated this way. After each decision, the remaining domains are cleaned by removing all the positions that are covered by placing the related item to its newly assigned position (*forward checking*).

Due to the huge number of symetries in such a problem, the resolution is halted after a given number of backtracks if no solution is found. In such a case (after 100 backtracks), the answer of the procedure is **no**.

That is clearly not a complete method, but our experiments compared to the bottom-left heuristic show that, our technique:

- answers **yes** whenever bottom-left says **yes**
- and answers much more often **yes** when bottom-left falsely says **no**

#### 4.1.4 Handling troops

Troops are relatively easy to handle. They need a small area compared to other items and, as already mentioned, have little possibilities for placement; they need to be seated on seats or benches. In practice, we load them after all the other items: first on fixed seats in the cabins, then on removable benches in the bays. In the first case, we do not take their area into account but only their weight. Indeed, if they are placed on fixed seats in the cabin, the only limitations are the number of seats (an information given for each type of aircraft) and the maximum payload of the aircraft. If they are seated on removable benches, in addition to their weights, we take the area of the bench into account.

## 4.2 A loading pattern generation method

This method consists of two phases:

1. generating a great number of feasible and *attractive* loadings;
2. selecting a subset from those loadings to cover all the items to be shipped with the final objective of minimizing the number of sorties.

### 4.2.1 Generating loadings

The first phase randomly generates a given number of loading patterns for each aircraft type and each destination: items to a destination are randomly selected and inserted in the aircraft bay with the loading algorithm described above until completion of that bay.

For each type of aircraft, the number of generated loadings for each pair (origin, destination) is proportionnal to the upper bound of that type. Experiments show that  $20 \times UB_{id}$  (where  $UB_{id}$  is the upper bound of aircraft type  $A_i$  for destination  $d$ ) is enough for good results. Actually, as the number of different items in our problem is quite small, generating more loading patterns leads to several occurrences of the same one.

Like in the list heuristic previously described, troops are loaded after all other items. They are not considered through the generation process.

### 4.2.2 Selecting a subset of loadings

Once enough loading patterns have been generated, we have to determine a subset of all those patterns *covering* the items to be shipped. We developed for this purpose a greedy algorithm based on Chvatal's heuristic for the set covering problem [1].

In the set covering problem, the data consist of a set of objects  $V$  and a collection  $E$  of sets of these objects. To each set  $E_i$  of  $E$  is associated a cost  $c_i$ . A subset  $J$  of  $E$  is called a *cover* if each object of  $V$  is covered by (*i.e.* belongs to) at least one of the subsets in  $J$ . The cost of the cover is  $\sum(c_i : E_i \in J)$ . The problem is to find a cover of minimum cost.

Chvatal’s heuristic consists at each iteration in selecting the set  $E_i$  having the maximum ratio  $\frac{|E_i|}{c_i}$ . Then, each object selected in  $E_i$  is removed from all the other sets of  $E$ . The procedure is repeated until all the objects are covered (*i.e.*, all the remaining sets of  $E$  are empty).

In the unit cost version (our problem), this heuristic picks up, at each iteration, the set (*i.e.*, the loading) that contains the greatest number of items.

That heuristic was meant for problems in which each item is unique. Our problem is a little bit different since several items of the same type have to be transported. Furthermore, there exist lower and upper bounds on the number of aircrafts of each type. Thus, we adapted Chvatal’s heuristic to tackle these additional constraints:

At the beginning, our heuristic selects at each iteration a loading not yet chosen, among the aircrafts for which the lower bound is not yet reached. Once enough aircrafts are selected to reach all lower bounds, the heuristics chooses a loading among the aircrafts for which the upper bound has not been reached.

Ties among candidate aircrafts are broken using a different criterion from Chvatal’s heuristic. Indeed, with Chvatal’s original criterion, aircrafts that contain very small items would be preferred to the ones that contain only one big item. Our experiments showed that early loading of cumbersome items is much more beneficial for the search. Therefore, we chose to select in priority the aircrafts for which the sum of *weight*  $\times$  *area* value is maximal.

Furthermore, an object type is removed from all remaining loadings only when all the objects of this type are covered, unlike Chvatal’s version which performs this removal at each iteration.

Our heuristic is very fast and thus can be run several times by randomly selecting a loading among equivalent candidates at each iteration. Troops are handled at the end of the process like in the list heuristic.

Note that once the selection is done, it may happen that the set of selected aircrafts carries too many items. A last phase consists in removing from the bay all the supplementary items.

## 5 Local searches

Our two heuristics (the list heuristic from section 4.1 and the pattern generation technique from section 4.2) provide a solution very quickly. But, due to the upper bounds on the number of aircrafts, it is sometimes impossible to load all the items. We thus apply several procedures (or local searches) to improve this first solution in order to load more items if not all, and to reduce the number of sorties. All these improvement techniques are applied in turn until no further improvement is obtained.

## 5.1 Loading all the remaining items

It is important to note that, in some cases, some items cannot be loaded whereas the upper bound  $UB_i$  of an aircraft type  $A_i$  is not reached. It is the case when the remaining items are not compatible with this aircraft type. We developed two methods:

1. A first improvement method consists in applying the list heuristic a fixed number of times, recording the best result. Between each iteration, the list of items is modified by moving forward in the list all the items that could not be loaded in the previous iteration. Note that the cost of successive solutions may not strictly decrease.
2. The other modification made to the solution seeks room in an aircraft for items that could not be loaded. The idea is to create a non saturated aircraft (*i.e.*, its upper bound has not yet been reached) and move items from other aircrafts to it. We try to load the remaining items in aircrafts in which room has been made.

## 5.2 Reducing the number of sorties

We designed a local search to reduce the number of sorties needed in the final projection. Its neighborhood considers each aircraft in turn and attempts to empty it by transferring all its items to other aircrafts. Even when no aircrafts can be saved, this local search is useful to increase the empty space in aircrafts and to insert unloaded items.

# 6 Experiments

We tested our methods on real-world instances provided by DGA. They presented the following characteristics:

- average number of items types no greater than 20;
- number of items per destination up to 5 000 (about 3 destinations for each air base);
- 3 aircraft types in average;
- 10 to 60 aircrafts per air-base fleet.

Human experts from DGA solved those problems but their solutions usually did not verify all the constraints. For example, constraints on pax (handling the benches, ...) are not strictly enforced. Moreover, pallets are only handled through their weight leading to loadings carrying parts of pallets which is physically impossible. Our results enforce all the declared constraints and thus may lead to apparently worse solutions.

However, in general, the results of our two heuristics are very similar to those manually obtained by DGA. For some instances, there exists a significant

improvement. In a few cases, manually obtained solutions are better, but they never save more than one aircraft compared to the automated algorithms (which is negligible). Moreover, recall that human solutions do not always verify all the given constraints.

The two base heuristics (the list-based and the set covering-based) give similar results. Comparing results before and after the local search show that it is well worth using it. Usually, quite all the items are loaded while even lowering the number of sorties. Table 5 presents our results obtained on benchmarks provided by DGA. The results were obtained in a matter of seconds for each data set.

The main advantage of our set of algorithms is that a solution is available in a matter of seconds (the very worst case took approximatively half an hour to be solved). This is to be compared with the full day requirements of each instance by the human experts. DGA engineers can now perform many simulations by varying the parameters of the projections.

## 7 Conclusion

The two developed methods are both currently used at DGA. Two kinds of improvements are planned:

- In the current version of the projection software, loading the aircrafts is separated from actually scheduling the flights. It is time to integrate the two processes, at least to some extent, but solving simultaneously the two sub-problems seems very hard.
- The second improvement concerns allowing a more flexible routing, in which a plane from a base could pick items at other bases before flying to its destinations (as it likely happens in real operations).

## Acknowledgements

This work was partially supported by Délégation Générale pour l'Armement (DGA). We wish to thank Patrick Journée and Franck Ouary for their helpful comments in writing this paper.

## References

- [1] V. Chvatal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4:233–235, 1979.
- [2] D. Liu and H. Teng. An improved bl-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European Journal of Operational Research*, 112:413–420, 1999.
- [3] S. Martello and P. Toth. *Knapsack Problems*. Wiley, 1990.
- [4] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

Aircrafts	A1	A2	A3
Number in the fleet	2	4	44
Maximum payload	25.00	110.00	24.00
Length of the bay	11.20	15.00	23.10
Width of the bay	3.18	7.00	3.57
Height of the bay	2.44	2.50	3.85

Table 3: Aircraft characteristics

	dest. A	dest. B	dest. C	Global Bounds
A1 aircraft	-	-	3-4	3-4
A2 aircraft	0-8	-	-	0-8
A3 aircraft	38-46	39-47	-	80-93

Table 4: Lower and upper bounds of the number of sorties

Set	Items	Pax	LH		LPG		LH w/ LS		LPG w/ LS	
1	1082	0	257	-	250	(1/0)	252	-	248	-
2	1082	0	309	(83/0)	233	(118/0)	296	(28/0)	256	(11/0)
3	1085	4877	272	(72/0)	280	(76/314)	277	(40/0)	279	(59/314)
4	367	1772	69	(0/180)	66	(0/367)	65	-	64	(0/164)
5	368	1405	72	-	66	(1/0)	68	-	64	-
6	368	1405	115	(0/17)	109	(1/416)	110	-	114	(0/416)
7	1658	0	275	(1/0)	268	-	273	-	261	-
8	1700	0	174	-	177	(15/0)	169	-	171	-

Table 5: Results on benchmarks – **LH** (resp. **LH w/ LS**) and **LPG** (resp. **LPG w/ LS**) give results obtained with our list heuristic and our loading pattern generation method (resp. followed by a local search). Results in brackets represent the remaining items and pax.