The assignment and loading transportation problem

The Assignment and Loading Transportation Problem

Gabriel Homsi[1], Jeremy Jordan[2], Silvano Martello[1], Michele Monaci[1]

[1]*DEI "Guglielmo Marconi", Alma Mater Studiorum Università di Bologna, Viale Risorgimento 2, I-40136 Bologna, Italy*
[2]*Department of Mathematics & Statistics, Air Force Institute of Technology, 2950 Hobson Way, Wright Patterson AFB, OH 45433, USA*
{gabriel.homsi@cirrelt.ca}, {silvano.martello, michele.monaci}@unibo.it, {Jeremy.Jordan@afit.edu}

https://doi.org/10.1016/j.ejor.2019.07.039

Abstract

We consider a generalization of the multiple knapsack problem that combines assignment and loading. The problem can arise in military and emergency situations in which one is required to refurnish a unit with a number of different goods available at different locations. We present a mathematical model and study Lagrangian and surrogate relaxations. We propose heuristic and metaheuristic approaches which we use to develop two overall approximation algorithms: a self-contained polynomial-time heuristic and a more time consuming matheuristic approach that makes use of a MILP solver. Solution times and accuracy of lower and upper bounds are computationally evaluated on a real military data set and on sets of both realistic and randomly generated instances.

Key words: multiple knapsack problem, assignment problem, loading constraints, emergency, integer linear programming formulations, heuristic algorithms, computational experiments.

1. Introduction

Motivated by a real-world military application, we introduce a generalization of the multiple knapsack assignment problem (see Kataoka and Yamada [10]) that can arise in emergency logistics. A unit asks to receive a certain number of items of different types, each characterized by a profit (the value it has for the requesting unit) and a weight. For each item type, the unit wants to receive a minimum and a maximum number of items. The items are available at different bases and a set of heterogeneous vehicles is available for delivering the items. Each vehicle has a maximum weight capacity and can only be used for picking items at one base and delivering them to the unit. For each vehicle and base, there is a cost associated with moving the vehicle from its current location to the base, and then from the base to the unit. The problem is to find a feasible assignment of requested items to vehicles that maximizes the overall profit minus the overall cost. This scenario resembles emergency response situations such as military problems where a major base, perhaps overseas, receives goods from supply bases spread throughout the world, as well as certain humanitarian contexts.

For instance, consider the surge in troops during the Iraq war in 2007, where U.S. military bases in the middle east experienced increased demand for goods due to the push for

additional troops and longer deployment durations. There was an urgent need to rapidly transport items such as food, clothing, medical supplies, ammunition, etc. in order to keep the bases properly stocked. Various other situations arise overseas when a conflict in a region causes unexpected increases in allied military resources. The cost of overseas missions certainly merits the employment of better optimization methods. The war with ISIS alone cost the U.S. over $11 billion and the U.K. nearly $2 billion, see McCarthy [16, 17]. Overseas war operations in total since 2001 is nearing $6 trillion for the U.S. alone, see Shane [20]. New optimization methods for the considered assignment and loading transportation problem could help to lower future costs.

Emergency transportation planning, in general, involves choosing the best way to ship goods or personnel from a set of departure points to a set of destination points. Humanitarian transportation planning is a complex problem that is receiving considerable attention in the operational research literature. We refer the interested reader to de la Torre, Dolinskaya, and Smilowitz [5] and to Gralla and Goentzel [8] for a general overview of transportation problems in humanitarian logistics and a comprehensive classification of the solution approaches. A recent special issue (see Besiou, Pedraza-Martinez, and Van Wassenhove [2]) covers a number of optimization techniques employed in various disaster response logistics.

While there is a large amount of literature on emergency transportation in humanitarian contexts, few military transportation problems have been studied so far. Akgun and Tansel [1] define the deployment planning problem to build optimal plans for the physical movement of personnel units across the world. Transportation assets are scheduled and routed using a branch-and-bound approach. Zheng [25] develops a generalized cut set algorithm to solve military transportation path optimization problems. Jordan and Weir [9] consider multiple objectives for planning the shortest path through a network, resulting in new average shortest path and average minimum cost flow formulations. Efficient algorithms and heuristics are then developed to solve the problem. Relatively speaking, there are very few military transportation studies.

However, the magnitude of the problem warrants further attention: transportation to military units is not only performed in emergency situations, but also in a huge number of day to day operations. In the United States military alone, the 2018 transportation costs exceed $19 billion (see United States Transportation Command [22]), and during an average week they conduct more than 1900 air missions (see Williams and Gay [23]). Efficient optimization techniques are thus necessary to reduce waste and optimize shipments. In this paper, we consider a new optimization approach that assigns a set of vehicles to departure bases and transports goods to a single destination base with minimal cost.

The paper is organized as follows. Section 2 defines the considered problem, that can be seen as a generalization of the multiple knapsack (assignment) problem. Section 3 introduces Lagrangian and surrogate relaxations that can be used to compute upper bounds on the optimal value and analyzes their theoretical properties. In Section 4 we propose a polynomial-time approach based on greedy heuristics and on iterated local search. In Section 5 we present a (non-polynomial) matheuristic approach. Finally, in Section 6 we give the outcome of computational experiments on a real military case study and on benchmarks of realistic and randomly generated instances. Conclusions are drawn in Section 7.

2. Problem Statement

We are given a set $T = \{1, 2, \ldots, n\}$ of item *types* (e.g., medical equipment, food supplies, explosives and like). Each *item* of type j has positive *profit* p_j and *weight* w_j ($j \in T$). There is a set $B = \{1, 2, \ldots b\}$ of (military or humanitarian) *bases* at which items are stored: each base k has a_{kj} items of type j available ($k \in B,\ j \in T$). A *unit* asks to receive at least ℓ_j and at most u_j items of each item type j (with ℓ_j and u_j possibly being zero). There is a set $V = \{1, 2, \ldots, m\}$ of different *vehicles* (aircrafts, trucks, ...), currently available at different locations, that can be used for transportation. Each vehicle has a *capacity* c_i ($i \in V$) and may be assigned to at most one base. The *cost* for moving vehicle i from its current location to base k, loading a subset of items, and delivering them to the unit is denoted by q_{ik} ($i \in V,\ k \in B$). We look for a loading of items into vehicles such that

- no vehicle is loaded with a set of items whose weight exceeds its capacity;
- the unit receives a satisfactory amount of the items it needs, and
- the difference between the total profit of the transported items and the total transportation cost is a maximum.

The resulting *Assignment and Loading Transportation Problem* (ALTP) can be formally defined as an *Integer Linear Program* (ILP) as follows. Let us introduce two sets of decision variables. For each vehicle $i \in V$, base $k \in B$, and item type $j \in T$, let x_{ikj} be an integer variable representing the number of items of type j picked up at base k by vehicle i. For each vehicle $i \in V$ and base $k \in B$, let y_{ik} be a binary variable taking the value one if and only if vehicle i is assigned to base k. A possible ILP model is then:

$$\max \sum_{i \in V} \sum_{k \in B} \sum_{j \in T} p_j x_{ikj} \;-\; \sum_{i \in V} \sum_{k \in B} q_{ik} y_{ik} \tag{1}$$

$$\sum_{k \in B} y_{ik} \;\leq\; 1 \qquad (i \in V) \tag{2}$$

$$\sum_{i \in V} x_{ikj} \;\leq\; a_{kj} \qquad (k \in B, j \in T) \tag{3}$$

$$\sum_{j \in T} w_j x_{ikj} \;\leq\; c_i y_{ik} \qquad (i \in V, k \in B) \tag{4}$$

$$\ell_j \leq \sum_{i \in V} \sum_{k \in B} x_{ikj} \;\leq\; u_j \qquad (j \in T) \tag{5}$$

$$x_{ikj} \;\in\; \mathbb{Z}^* \qquad (i \in V, k \in B, j \in T) \tag{6}$$

$$y_{ik} \;\in\; \{0, 1\} \qquad (i \in V, k \in B), \tag{7}$$

where $\mathbb{Z}^*$ denotes the set of non-negative integers. The objective function (1) gives the difference between the total profit of the picked up items and the total cost for their transportation. Constraints (2) impose that each vehicle be assigned to at most one base. The availability of items at each base is expressed by (3), while the capacity limit of the vehicles at each base is forced by (4). Finally, constraints (5) specify the demand bounds for each item type, and constraints (6)-(7) define the domain of the variables.

We will assume in the following that all input data are non-negative integers.

2.1 Complexity and related problems

Problem ALTP is a generalization of the *Multiple Knapsack Assignment Problem* (MKAP), that has been recently studied by Kataoka and Yamada [10], Lalla-Ruiz and Voß [13], and Martello and Monaci [14]. In the MKAP one is given m knapsacks (the vehicles) and a set of n items, partitioned into a number of classes (the bases): a knapsack can only contain items of the same class, and the objective is to assign a set of knapsacks to each class in such a way that the total profit of the selected items is a maximum.

In the ALTP, the additional features are the transportation costs q_{ik} and the lower and upper bounds ℓ_j and u_j on the number of items of each type j to be selected. It follows that for our problem, differently from the case of the MKAP: (i) the optimal value can be negative; (ii) due to the lower bounding constraints (5), an instance can have no feasible solution.

The MKAP is in turn a generalization of the 0-1 *Multiple Knapsack Problem* (MKP), that arises when there is a single base. There is a huge literature on the MKP, for which we refer the reader to the specific chapters in the classical books by Martello and Toth [15] and Kellerer, Pferschy, and Pisinger [11]. The MKP has been shown to be strongly $\mathcal{NP}$-hard by transformation from 3-partition (see, e.g., Martello and Toth [15], Section 1.3). It follows that the ALTP is strongly $\mathcal{NP}$-hard. Note in addition that, due to the presence of the lower bound constraints in (5), even deciding if an instance of the ALTP has a feasible solution is a strongly $\mathcal{NP}$-complete problem, even in the case where there is a single base. This is easily shown by reduction from the recognition version of the bin packing problem: given n items with weights w_j, deciding whether they can be packed into m identical containers of capacity c corresponds to finding a feasible solution to an ALTP instance with n item types, each with $\ell_j = u_j = 1$, a single base, and m identical vehicles of capacity c.

Combinations of multiple knapsack problems with assignment/transportation constraints, arising in a variety of managerial and emergency situations, are attracting increasing interest from the literature. After the contributions of Dawande, Kalagnanam, Keskinocak, Salman, and Ravi [4] (inventory problems in the steel industry), and Dahl and Foldnes [3] (wireless telecommunications), in recent years new studies have been presented by Kataoka and Yamada [10] (MKAP), Laalaoui and M'Hallah [12] (machine scheduling), Dimitrov, Solow, Szmerekovsky, and Guo [7] (emergency relocation), Simon, Apte, and Regnier [21] (equipment selection for Marines), Zhen, Wang, Wang and Qu [24] (maritime shipping), and Diaz, Handl, and Xu [6] (production planning).

3. Upper Bounds

In this section, we discuss relaxations of the ALTP that can be used to compute upper bounds on the optimal value.

3.1 Lagrangian relaxation

Let us relax constraints (4) in a Lagrangian fashion through an array $\lambda = [\lambda_{ik}]$ of mb non-negative multipliers. The resulting objective function is

$$\max \sum_{i\in V}\sum_{k\in B}\sum_{j\in T} p_j x_{ikj} - \sum_{i\in V}\sum_{k\in B} q_{ik} y_{ik} + \sum_{i\in V}\sum_{k\in B} \lambda_{ik}(c_i y_{ik} - \sum_{j\in T} w_j x_{ikj}). \tag{8}$$

The Lagrangian relaxation can then be written as:

$$\max \sum_{i\in V}\sum_{k\in B}\sum_{j\in T} \widetilde{p}_{ikj} x_{ikj} + \sum_{i\in V}\sum_{k\in B} \widetilde{q}_{ik} y_{ik} \text{ subject to } (2), (3), (5) - (7), \tag{9}$$

where

$$\widetilde{p}_{ikj} = p_j - \lambda_{ik} w_j \qquad (i \in V, k \in B, j \in T), \tag{10}$$

$$\widetilde{q}_{ik} = \lambda_{ik} c_i - q_{ik} \qquad (i \in V, k \in B). \tag{11}$$

This relaxation has an interesting property, that will be relevant for obtaining the polynomial-time approach of Section 4

Property 1 *For any set of multipliers, the Lagrangian relaxation* (9) *of the ALTP is solvable in polynomial time.*

Proof. As the relaxed constraints (4) are the only ones involving both sets of variables, for each choice of multipliers the problem decomposes into two independent subproblems, one in the x variables, one in the y variables:

$$(LX) \qquad \max \sum_{i\in V}\sum_{k\in B}\sum_{j\in T} \widetilde{p}_{ikj} x_{ikj} \text{ subject to } (3), (5), (6); \tag{12}$$

$$(LY) \qquad \max \sum_{i\in V}\sum_{k\in B} \widetilde{q}_{ik} y_{ik} \text{ subject to } (2), (7). \tag{13}$$

Problem (LX) decomposes in turn into n independent subproblems (one per item type). For each item type j, we must load into a set of vehicles at least ℓ_j and at most u_j items, taken from a set of bases (without exceeding the corresponding availabilities). As the vehicles have no capacity limit, an optimal solution for item type j will consider the pairs (vehicle i, base k) according to non-increasing $\widetilde{p}_{ikj}$ values. Observe that such a sorting does not depend on j, as it is only induced by λ_{ik}, see (10). Accordingly, Procedure **LX**, shown in Algorithm 1, starts by sorting all pairs (i, k) by non-decreasing λ_{ik}, and then considers one item type j at a time. At the j-th iteration, s_j denotes the current number of selected items of type j, and $\overline{a}_{kj}$ ($k \in B$) the current availability of items of type j at base k. For each pair (i, k) in order, if the Lagrangian profit is non-negative (resp. negative), the maximum (resp. minimum) possible number of items is added to s_j.

Problem (LY) decomposes into m independent subproblems (one per vehicle). For each vehicle i, we must assign the vehicle to at most one base. An optimal choice is then to assign it to the base with maximum associated Lagrangian profit, provided such value is

Algorithm 1 Procedure **LX**(λ)

1: sort pairs (i,k) according to non-decreasing values of λ_{ik};
2: **for all** item types $j \in T$ **do**
3: $\quad s_j := 0$;
4: $\quad$ **for** $k := 1$ **to** b **do** $\overline{a}_{kj} := a_{kj}$;
5: $\quad$ **for all** pairs (i,k), in the order determined at Step 1, **do**
6: $\quad\quad \widetilde{p}_{ikj} := p_j - \lambda_{ik} w_j$;
7: $\quad\quad$ **if** $\widetilde{p}_{ikj} \geq 0$ **then** $x_{ikj} := \min\{\overline{a}_{kj}, u_j - s_j\}$ **else** $x_{ikj} := \min\{\overline{a}_{kj}, \max\{l_j - s_j, 0\}\}$;
8: $\quad\quad s_j := s_j + x_{ikj}$, $\overline{a}_{kj} := \overline{a}_{kj} - x_{ikj}$
9: $\quad$ **end for**
10: **end for**
11: **return** $\sum_{i \in V} \sum_{k \in B} \sum_{j \in T} \widetilde{p}_{ikj} x_{ikj}$

non-negative. The resulting solution is produced by the simple Procedure **LY**, shown in Algorithm 2.

The Lagrangian upper bound for a given set of multipliers (array λ) can then be computed as shown in Algorithm 3. Procedure **LX** takes $O(mb \max\{\log m, \log b\})$ time for the initial pair sorting. It then performs n iterations, each of which requires $O(mb)$ time, so the overall complexity of **LX** is $O(mb \max\{n, \log m, \log b\})$. Procedure **LY** takes linear time $O(mb)$. The overall time complexity needed to compute, for a given set of multipliers, the corresponding Lagrangian upper bound on (1)-(7) is thus $O(mb \max\{n, \log m, \log b\})$. □

Algorithm 2 Procedure **LY**($\widetilde{q}$)

1: **for all** vehicles $i \in V$ **do**
2: $\quad$ **for** $k := 1$ **to** b **do** $y_{ik} := 0$;
3: $\quad k_i = \arg\max_{k \in B}\{\widetilde{q}_{ik}\}$;
4: $\quad$ **if** $\widetilde{q}_{ik_i} \geq 0$ **then** $y_{ik_i} := 1$;
5: **end for**
6: **return** $\sum_{i \in V} \sum_{k \in B} \widetilde{q}_{ik} y_{ik}$

It should be observed that relaxation (9) has the *integrality property*, i.e., the solution of its linear programming relaxation is integer. It follows (see, e.g., Nemhauser and Wolsey [18], Chapter 6) that the *Lagrangian dual* (i.e., the problem of determining the multipliers providing the lowest upper bound) yields the same upper bound as the linear programming relaxation of (1)-(7). Subgradient optimization techniques could produce good approximations to the Lagrangian dual, but may be time-consuming. In order to quickly compute an

Algorithm 3 Procedure **Lagrangian**(λ)

1: $UX :=$ **LX**(λ);
2: **for all** $i \in V$, $k \in B$ **do** $\widetilde{q}_{ik} := c_i \lambda_{ik} - q_{ik}$;
3: $UY :=$ **LY**($\widetilde{q}$)
4: **return** $UX + UY$

upper bound we thus preferred to execute **Lagrangian**(λ) with a limited number of prefixed λ multipliers (see Section 6).

3.2 Surrogate relaxation

A surrogate relaxation of (1)-(7) can be obtained by replacing the mb capacity constraints (4) with b surrogate constraints (one per base), i.e., with

$$\sum_{i \in V} \pi_i \sum_{j \in T} w_j x_{ikj} \leq \sum_{i \in V} \pi_i c_i y_{ik} \qquad (k \in B), \tag{14}$$

where $\pi = [\pi_i]$ is an array of m non-negative multipliers. Intuitively we replace, for each base k, the set of vehicles that are assigned to k with a single vehicle having capacity equal to the weighted sum of their capacities.

For this relaxation it is possible to determine the optimal dual multipliers, generalizing to the ALTP a known property of the MKP (see Martello and Toth [15]).

Property 2 *Optimal surrogate multipliers for* (1) - (3), (14), (5) - (7) *are* $\pi_i = \vartheta$ *(ϑ any positive constant) for all* $i \in V$.

Proof. Let $\pi_h = \min_{i \in V}\{\pi_i\}$. Given a feasible solution x with $x_{ikj} > 0$ for a triplet (i, k, j) with $i \neq h$, we can obtain an equivalent solution x' in which $x'_{ikj} = 0$ and $x'_{hkj} = x_{hkj} + x_{ikj}$, i.e., we can move the current content of vehicle i to vehicle h. In this way the left-hand side of the k-th constraint (14) decreases (or is unchanged, if $\pi_i = \pi_h$) while those of the other constraints are unchanged, i.e., solution x' is feasible and has the same profit as x. By iterating, we get a solution in which all items are loaded into vehicle h and $x_{ikj} = 0$ for all $i \neq h, k \in B, j \in T$. It follows that the surrogate capacity constraints (14) can be written as

$$\pi_h \sum_{j \in T} w_j x_{hkj} \leq \sum_{i \in V} \pi_i c_i y_{ik} \qquad (k \in B). \tag{15}$$

Now observe that

(i) if $\pi_h = 0$, the left-hand side of each constraint (15) is zero, which allows the optimal choice $y_{ik} = 0$ for each $i \in V$ and $k \in B$. In this way, the solution of the surrogate relaxation loads all available items into vehicle h, i.e., it produces the trivial upper bound $\sum_{j \in T} p_j \overline{u}_j$, where $\overline{u}_j = \min\{u_j, \sum_{k \in B} a_{kj}\}$;

(ii) if $\pi_h > 0$, we can divide both terms of (15) by π_h obtaining

$$\sum_{j \in T} w_j x_{hkj} \leq \sum_{i \in V} \frac{\pi_i}{\pi_h} c_i y_{ik} \qquad (k \in B). \tag{16}$$

Since $\pi_i / \pi_h \geq 1$ for any i, the choice $\pi_i = \vartheta$ (any positive constant) for all $i \in V$ produces the minimum right-hand side, and hence the tightest (minimum) surrogate bound. □

Differently from the Lagrangian case, the optimal multipliers can be determined for this relaxation. However, the relaxation with such multipliers cannot be solved in polynomial time unless $\mathcal{P} = \mathcal{NP}$:

Property 3 *For the optimal surrogate multipliers, relaxation* (1) - (3)*,* (14)*,* (5) - (7) *is strongly $\mathcal{NP}$-hard.*

Proof. Given an MKP instance, consider the surrogate relaxation of an ALTP instance with

1. m bases and m vehicles, with assignment costs $q_{ii} = 0$ and $q_{ik} = +\infty$ if $i \neq k$;
2. availabilities $a_{kj} = 1$ for all k, j;
3. lower and upper bounds $\ell_j = 0$ and $u_j = 1$ for all j.

Due to **1.**, bases and vehicles coincide, and any optimal solution to the relaxed problem has $y_{ii} = 1$ and $y_{ik} = 0$ if $i \neq k$. It follows that: (i) the second term of (1) vanishes; (ii) constraints (2) and (7) are automatically satisfied; (iii) the right-hand side of each constraint (14) is $\pi_k c_k$, and hence (iv) y_{ik} variables can be eliminated.
Adopting optimal multipliers $\pi_i = 1$ for all i (see Property 2), we can define variables $\xi_{kj} = \sum_{i \in V} x_{ikj}$. The surrogate relaxation becomes then

$$\max \sum_{k \in B} \sum_{j \in T} p_j \xi_{kj} \tag{17}$$

$$\xi_{kj} \leq 1 \qquad (k \in B, j \in T) \tag{18}$$

$$\sum_{j \in T} w_j \xi_{kj} \leq c_k \qquad (k \in B) \tag{19}$$

$$0 \leq \sum_{k \in B} \xi_{kj} \leq 1 \qquad (j \in T) \tag{20}$$

$$\xi_{kj} \in \mathbb{Z}^* \qquad (k \in B, j \in T). \tag{21}$$

As constraints (18) are redundant, the resulting model describes an MKP, which is known to be strongly $\mathcal{NP}$-hard. □

4. Constructive and metaheuristic algorithms

In this section, we present two simple constructive heuristics and an iterated local search approach. These methods are purely combinatorial and can quickly produce a feasible solution with no need for a *Mixed-Integer Linear Programming* (MILP) solver. Combining them with the Lagrangian upper bound computation of Section 3.1, we obtain a viable, self-contained, polynomial-time approach to the ALTP that can be used when a solution has to be provided in a very short time like, e.g., in an emergency situation. A more time consuming (matheuristic) algorithm will be proposed in Section 5.

4.1 Constructive heuristics

We next introduce two constructive heuristics. Both start with an empty solution and follow a greedy strategy that assigns a vehicle at a time to a base and determines a set of items to be loaded into it. The possible assignments of a vehicle i to a base k are considered by non-decreasing cost q_{ik}. An assignment, say of i to k, is considered potentially feasible if

vehicle i is not yet assigned. In this case both algorithms consider the item types j according to non-increasing p_j/w_j ratios and load vehicle i according to its current residual capacity $\overline{c}_i$, the number s_j of already loaded items of type j, and the current residual availability $\overline{a}_{kj}$ of item type j at base k.

The first heuristic, **H1**, shown in Algorithm 4, loads the maximum possible number of items into the current vehicle.

The solutions provided by **H1** satisfy all capacity, availability, and upper bound constraints. The method, however, disregards the lower bound constraints in (5), and hence it can fail in producing a feasible solution. When this happens, a second constructive heuristic, **H2**, is executed, that explicitly takes such constraints into account and hence is more likely to determine a feasible solution. The algorithm, for which we do not give a detailed description, only differs from **H1** in two points:

- in the loading decision executed in the inner '**for all**' loop, the term $u_j - s_j$ is replaced by $\ell_j - s_j$, so as to avoid unnecessary item loading;
- the assignment test is modified to '**if** $\sum_{j \in T} x_{ikj} > 0$ **then** assign vehicle i to base k', so as to load necessary items even in case the resulting loading is unprofitable.

In addition, the solution produced by **H2** is improved by a post-processing procedure that considers one used vehicle at a time and loads additional items of each item type for which there is residual availability and the upper bound has not been reached.

Both **H1** and **H2** require $O(mb \log mb + n \log n + nb)$ for sorting and initialization. The outer for-each loop is executed mb times: at each iteration, the inner for-each loop takes $O(n)$ time. The overall complexity of both algorithms is thus $O(mb \log mb + n \log n + mnb)$.

4.2 Iterated local search

The best feasible solution obtained by the constructive heuristics (or the empty solution, if no feasible solution was obtained) can be improved through iterated local search as follows.

Algorithm 4 Procedure **H1**

```
sort the assignment pairs (i, k) by non-decreasing q_ik values;
sort the item types j by non-increasing p_j/w_j ratios;
set s_j := 0 (j ∈ T) and ā_kj := a_kj (k ∈ B, j ∈ T);
for all pairs (i, k) in order do
   if vehicle i is not yet assigned then
      c̄_i := c_i;
      for all item types j in order do
         x_ikj := max{0, min{⌊c̄_i/w_j⌋, u_j − s_j, ā_kj}};
         c̄_i := c̄_i − w_j x_ikj, s_j := s_j + x_ikj, and ā_kj := ā_kj − x_ikj
      end for;
      if Σ_{j∈T} x_ikj > 0 and Σ_{j∈T} p_j x_ikj − q_ik > 0 then assign vehicle i to base k
   end if
end for
```

For the sake of simplicity, let us denote by $\sigma = (x, y)$ the current solution, where $x = [x_{ikj}]$ is the item loading and $y = [y_{ik}]$ is the vehicle assignment, and by $z(\sigma)$ the corresponding profit (see (1)). The procedure can accept solutions in which some constraints are violated, but in this case it penalizes the corresponding objective function value. Specifically, the constraints on availability of items at the bases, vehicle capacity, and item demand lower and upper bounds can be relaxed. The corresponding violations,

$$\textit{availability: } e_a(x) = \sum_{k\in B}\sum_{j\in T} \max\{0, \sum_{i\in V} x_{ikj} - a_{kj}\}; \tag{22}$$

$$\textit{capacity: } e_c(x) = \sum_{i\in V} \max\{0, \sum_{k\in B}\sum_{j\in T} w_j x_{ikj} - c_i\}; \tag{23}$$

$$\textit{demand lower bound: } e_\ell(x) = \sum_{j\in T} \max\{0, \ell_j - \sum_{i\in V}\sum_{k\in B} x_{ikj}\}; \tag{24}$$

$$\textit{demand upper bound: } e_u(x) = \sum_{j\in T} \max\{0, \sum_{i\in V}\sum_{k\in B} x_{ikj} - u_j\}, \tag{25}$$

are then used to obtain a penalized solution value

$$\widetilde{z}(\sigma) = z(\sigma) - \mu_a e_a(x) - \mu_c e_c(x) - \mu_\ell e_\ell(x) - \mu_u e_u(x) \tag{26}$$

where μ_a, μ_c, μ_ℓ, and μ_u are appropriate *penalty parameters.*

Procedure **ILS**, shown in Algorithm 5, receives in input the incumbent solution σ produced by **H1** (and **H2**), and two parameters, I and F, that specify the number of iterations and the frequency of penalty adjustments, respectively. At each iteration, it randomly changes vehicle assignments (**Shake**) and improves the resulting solution by modifying the loading decisions (**Reload**). The procedure operates on a solution $\widetilde{\sigma}$ that stores the (possibly infeasible) best solution found for the penalized objective function (26). In addition, it stores in σ^* the best feasible solution found by the whole process, if any.

Shake is a simple procedure (for which we do not provide the explicit statement) that receives a solution $\widetilde{\sigma}$ and randomly selects a vehicle i and a base k ($k \in B \cup \{0\}$). Vehicle i is de-assigned ($k = 0$) with probability 0.5 (value settled on the basis of computational

Algorithm 5 Procedure **ILS**(σ,I,F,R)

1: $\widetilde{\sigma} := \sigma$;
2: **if** σ is feasible **then** $\sigma^* := \sigma$ **else** $\sigma^* := (\underline{\mathbf{0}}, \underline{\mathbf{0}})$, $z(\sigma^*) := -\infty$;
3: **for** $r := 1$ **to** I **do**
4: $\quad \sigma :=$ **Shake**$(\widetilde{\sigma})$;
5: $\quad \sigma :=$ **Reload**(σ);
6: $\quad$ **if** $\widetilde{z}(\sigma) > \widetilde{z}(\widetilde{\sigma})$ **then** $\widetilde{\sigma} := \sigma$;
7: $\quad$ **if** σ is feasible **and** $z(\sigma) > z(\sigma^*)$ **then** $\sigma^* := \sigma$;
8: $\quad$ **if** $r \bmod F = 0$ **then** update the penalty parameters
9: **end for**
10: **return** σ^*

experiments): in this case its current contents are unloaded. All other bases $k \in B$ are selected with probability $0.5/|B|$: in this case vehicle i is assigned to base k without modifying its loading. Note that the new solution σ may be feasible or infeasible.

Similarly, **Reload** consists of a local search that operates on a simple neighborhood defined by all pairs (vehicle i, item type j), and explores it in random order. For each selected pair (i, j), the procedure evaluates two moves, consisting of loading (resp. unloading) one item of type j into (resp. from) vehicle i. (But moves that increase the availability violation $e_a(x)$ (see (22)) are not allowed). As soon as a solution with better $\widetilde{z}(\sigma)$ is found, σ is updated and the search is iterated, with a prefixed limit R on the number of iterations.

The penalty parameters are dynamically adjusted every F iterations, according to the number of times each violation (22)-(25) occurred in the last solutions generated by **Reload**: If a constraint was violated in more than 10% of the solutions, the corresponding penalty parameter is increased, and otherwise it is decreased.

Procedure **Shake** takes $O(n)$ time to either unload the contents of the selected vehicle or to compute the updated penalty $e_a(x)$. Procedure **Reload** performs a constant number R of iterations, each taking $O(nm)$ time. As both procedures are executed at most I times, the overall time complexity of **ILS** is $O(nm)$.

4.3 Overall combinatorial algorithm

Our overall polynomial-time heuristic, **H-Poly**, shown in Algorithm 6, executes the constructive heuristics and computes the best Lagrangian upper bound relative to three vectors of Lagrangian multipliers: $\lambda_{ik} = 0$, or $\lambda_{ik} = 1$, or $\lambda_{ik} = q_{ik}/c_i$ $(i \in V, k \in B)$. Finally, it possibly performs the iterated local search (with parameters I and F).

As the time complexities of **Lagrangian** and **ILS** are dominated by that of **H1** (and **H2**), this polynomial-time heuristic has complexity $O(mb \log mb + n \log n + mnb)$.

5. A matheuristic approach

The many conflicting constraints of the ALTP may cause **ILS** (Algorithm 5) to quickly stagnate. In this section we present a non-polynomial approach that defines a neighborhood of exponential size and explores it through a MILP solver. The approach is based on

Algorithm 6 Procedure **H-Poly**(I, F, R)

1: execute **H1**;
2: **if** no feasible solution is returned by **H1 then** execute **H2**;
3: let σ be the best solution found, if any, and let $z(\sigma)$ be its profit;
4: **for all** $i \in V$, $k \in B$ **do** $\lambda^r_{ik} := q_{ik}/c_i$;
5: $U_P := \min\{$**Lagrangian(0)**, **Lagrangian(1)**, **Lagrangian**$(\lambda^r)\}$;
6: **if** $z(\sigma) < U_P$ **then** execute **ILS**(σ, I, F, R)
7: **return** σ, U_P

mathematical model (1)-(7). Given the incumbent solution $\sigma = (x, y)$, let

$$Y^0(\sigma) = \{(i,k) : i \in V, k \in B, y_{ik} = 0\}; \tag{27}$$

$$Y^1(\sigma) = \{(i,k) : i \in V, k \in B, y_{ik} = 1\}. \tag{28}$$

Our neighborhood is obtained by randomly selecting a subset $\overline{Y}^0 \subset Y^0(\sigma)$ of unused (vehicle, base) assignments, and defining a restricted problem instance in which all such assignments are forbidden. In our implementation, all (i,k) pairs of $Y^0(\sigma)$ have the same probability $\mathbb{P}$ of being selected for inclusion in $\overline{Y}^0$. We correspondingly run a MILP solver, with a prefixed time limit τ_1, on model (1)-(7) with the additional constraints:

$$y_{ik} = 0 \qquad (i,k) \in \overline{Y}^0. \tag{29}$$

Note that all non-zero assignments are preserved, so the (possible) feasibility of σ is maintained, and hence the solver can use σ as a starting solution. A large neighborhood of the current solution is thus described by the restricted model, that can be effectively solved (either exactly or heuristically), taking advantage of the wide arsenal of tools (cuts, heuristics, propagations, etc.) available in state-of-the-art MILP solvers.

When the solver does not improve the incumbent solution σ, parameters τ_1 and $\mathbb{P}$ are temporarily updated according to prefixed parameters δ_τ and $\delta_{\mathbb{P}}$, to increase the neighborhood size and the time allowed for the exploration.

If instead σ was improved, parameters τ_1 and $\mathbb{P}$ are re-stored to their initial values. In addition, if the solver was unable to prove optimality of the solution found, a second run is executed (with time limit $\tau_2 < \tau_1$) on the model obtained from (1)-(7) by fixing *all* variables y_{ik} to their current value.

As the computational effort required to solve the latter formulation is considerably smaller, the process is typically able to achieve near-optimal loading decisions.

Our overall matheuristic, **M-Exp**, is summarized in Algorithm 7, where `TL` denotes the overall CPU time allowed, and τ_1 and τ_2 are prefixed fractions of `TL`. We denote by

Algorithm 7 Procedure **M-Exp**(I,F,R)

execute **H-Poly(I,F,R)** and let σ and U be the returned solution and upper bound;
possibly improve U through the surrogate relaxation of Section 3.1;
$\tau_1' = \tau_1; \mathbb{P}' = \mathbb{P}$;
while time limit `TL` not exceeded **and** $z(\sigma) < U$ **do**
  define $Y^0(\sigma)$ (see (27)) and extract $\overline{Y}^0$ using probability $\mathbb{P}$;
  $\sigma :=$ **MILP**($\overline{Y}^0$, τ_1);
  if σ was not improved **then**
    $\tau_1 = \tau_1 + \delta_\tau$; $\mathbb{P} = \max\{0, \mathbb{P} - \delta_{\mathbb{P}}\}$
  else
    $\tau_1 = \tau_1'; \mathbb{P} = \mathbb{P}'$;
    if the solver was terminated by time limit **then** $\sigma :=$ **MILP**($Y^0(\sigma) \cup Y^1(\sigma)$, τ_2)
  end if
end while
return σ

MILP(S, τ) a black box procedure that runs the solver, with time limit τ, on model (1)-(7) with all variables of S fixed at their current value.

6. Computational experiments

In this section we evaluate the computational performance of our overall approach and that of a MILP solver directly executed on mathematical model (1)-(7). All procedures were implemented in C++, using the Gurobi Optimizer 8.0 as MILP solver. All experiments were executed, in single thread mode, on a computer with a 3.30 GHz Intel Core i7-3960X processor and 64 GB of RAM.

The proposed algorithms should be used in a number of different contexts: day to day operations, standard periodic schedules, and emergency situations. For example, in the United States military alone, the average number of missions per week, only considering air shipments, is higher than 1900, i.e., one every five minutes assuming 24/7 activities (see Williams and Gay [23]). For this reason, we are interested in having the possibility of solving instances both very quickly and in situations where a large amount of time is available. The experiments in this section evaluate thus the performance of our approaches for several different time limits TL, ranging from very small to large values. All time limits are expressed in CPU seconds.

On the basis of preliminary experiments, the following values are adopted for the parameters needed by the various procedures:

- penalty parameters: $\mu_a = \mu_c = \mu_u = 10$, and $\mu_\ell = 10^5$;
- maximum number of **ILS** iterations:
  – for **H-Poly** stand-alone: $I = 10^6$ if TL ≤ 50 and $I = 2 \cdot 10^6$ otherwise;
  – within **M-Exp**: $I = 2 \cdot 10^4$ if TL ≤ 50 and $I = 3 \cdot 10^4$ otherwise;
- maximum number of **Reload** iterations: $R = 100$;
- penalty parameters updating: $F = 100$;
- probabilities of fixing y variables to zero: $\mathbb{P} = 0.6$, $\delta_{\mathbb{P}} = 0.1$;
- parameters for MILP executions: $\tau_1 = 0.1\,$TL, $\delta_\tau = 0.1\,$TL, $\tau_2 = 0.05\,$TL.

The surrogate relaxation used by **M-Exp** was always computed with multipliers $\pi_i = 1$, for all i (see Property 2).

6.1 Real-world case study

We obtained a real-world ALTP instance from the United States Department of Defense. The case study involves 12 bases, 11 item types, and 8496 vehicles. The characteristics of the instance are summarized in Table 1. The item types are those typically used by the U.S. Defense Contingency: food, clothing, fuel, construction materials, ammunition, personal demand items, medical material, repair parts, etc. (See [19] for the detailed list). The

	Item types					Vehicles	
	ℓ_j	u_j	p_j	w_j	a_{kj}	c_i	q_{ik}
min	50	140	103	2330	0	44,000	0
max	200	250	973	23,669	40	240,000	1,363,945
average	130.00	201.82	556.27	10,837.64	17.04	134,970.02	193,690.94

Table 1: Characteristics of the real-world instance.

locations of the bases and the details of the vehicles are sensitive, and hence we did not receive them.

For the real-world instance, the ILP model of Section 2 results to be as follows:

- number of constraints = 110,602;
- number of integer variables $[x_{ikj}]$ = 1,121,472;
- number of binary variables $[y_{ik}]$ = 101,952;
- number of non-zero constraint matrix elements = 4,689,792.

In Table 2 we compare the results obtained, for different time limits (expressed in seconds), by the ILP model of Section 2 (solved using Gurobi), the polynomial-time heuristic of Section 4, and the matheuristic algorithm of Section 5. The first column gives the time limit. For each algorithm, we report the lower and upper bounds, LB and UB, computed within the time limit and the percentage gap between the lower bound and the best available upper bound (say, U^*, underlined in the table), computed as $100(U^* - \text{LB})/U^*$. In addition, for **H-Poly** we give the actual CPU time (in seconds) used. (CPU time and time limit coincide for **ILP** and **M-Exp**.) For each time limit, the best solution found appears in bold. The

	ILP			**H-Poly**				**M-Exp**		
TL	LB	UB	%Gap	LB	UB	%Gap	time	LB	UB	%Gap
25	–	1022352	–	**982347**	1216542	3.89	25.01	970834	1216542	5.02
50	–	1022352	–	**982347**	1216542	3.89	50.01	970834	1216542	5.02
100	–	1022352	–	**982347**	1216542	3.89	100.01	970834	1216542	5.02
200	–	1022352	–	991297	1216542	3.02	200.01	**1018752**	1216542	0.33
300	1018303	1022352	0.37	991297	1216542	3.02	300.01	**1020099**	1216542	0.20
400	1019083	1022352	0.30	991297	1216542	3.02	400.00	**1020178**	1216542	0.19
500	1019083	1022352	0.30	991297	1216542	3.02	500.00	**1020178**	1216542	0.19
600	1019083	1022352	0.30	991297	1216542	3.02	542.00	**1019368**	1216542	0.27
700	1019083	1022352	0.30	991297	1216542	3.02	539.89	**1019368**	1216542	0.27
800	1019083	1022352	0.30	991297	1216542	3.02	538.70	**1019526**	1216542	0.25
900	1019083	1022120	0.30	991297	1216542	3.02	542.11	**1019526**	1216542	0.25
1000	1019083	1022120	0.30	991297	1216542	3.02	539.14	**1019526**	1216542	0.25

Table 2: Results for the real-world instance.

initial upper bound (1,022,352) was computed in 11.36 seconds by dropping integrality constraints (6)-(7) (*continuous relaxation*). Note that Gurobi is unable to improve such value in the next 800 seconds.

The table shows that:

- for small time limits the solver is unable to even find a feasible solution to model (1)-(7), and **H-Poly** turns out to be the best approach, clearly dominating **M-Exp**;
- the picture changes for large time limits, for which **H-Poly** is dominated by the other approaches. **M-Exp** produces the best solutions while the best upper bound, U^*, is obtained by running the solver on (1)-(7). The tiny percentage gaps indicate that **M-Exp** obtains a solution very close to the optimum.

Observe that **M-Exp** does not have a monotonic behavior: it may happen that larger time limits produce slightly worse solution values. This is due to the use of inner time limits (τ_1 and τ_2) that are fractions of the overall time limit `TL`, which can produce a different neighborhood exploration.

The best solution found, underlined in the table, was obtained by **M-Exp** for `TL` = 400 and for `TL` = 500, and has the following characteristics:

- objective function value 1020178, with overall profit 1158190 and overall cost 138012;
- number of items delivered 1977 (89.05% of the sum of upper bounds, 138.25% of the sum of lower bounds);
- number of vehicles used 192 (2.26% of the total);
- average percentage load of the used vehicles 75.90%;
- average transportation cost per vehicle 719;
- minimum and maximum net profit of a vehicle −8896 and 29035 (profit of the transported items minus movement cost);
- number of bases visited 11 (91.67% of the total);
- number of pairs (base, item type) for which all available items have been picked up 111 (84.09% of the total).

From an operational point of view, the results appear very satisfactory. The base receives a sufficient number of all requested item types, not very far from their maximum request. The solution shows a very efficient use of the vehicles: a limited fraction of them is used and their load is high, which results in transportation costs considerably smaller than the average cost (see Table 1). It is worth noting that some vehicles have a negative net profit, clearly needed to satisfy all minimum requests. All bases but one are visited and, in most cases, all available items of some type are picked up.

The solution improves upon those produced through existing manual methods. However, details on manual solutions are not available due to the sensitive nature of the information.

6.2 Realistic instances

Starting from the real-world instance of the previous section, we produced an additional benchmark of 30 *realistic* instances, obtained by emulating the original input data through slight random modification. The benchmark consists of six sets of five instances each, produced as follows:

- REA-c: capacity of vehicle i uniformly random in $[0.9\,c_i,\ 1.1\,c_i]$ ($i \in V$);
- REA-p: profit of item type j uniformly random in $[0.9\,p_j,\ 1.1\,p_j]$ ($j \in T$);
- REA-w: weight of item type j uniformly random in $[0.9\,w_j,\ 1.1\,w_j]$ ($j \in T$);
- REA-b: lower and upper bounds for item type j uniformly random in $[0.9\,\ell_j,\ 1.1\,\ell_j]$ and $[0.9\,u_j,\ 1.1\,u_j]$ ($j \in T$);
- REA-a: availability of item type j at base k uniformly random in $[0.9\,a_{kj},\ 1.1\,a_{kj}]$ ($k \in B$, $j \in T$);
- REA-q: cost of pair (vehicle i base k) uniformly random in $[0.9\,q_{ik},\ 1.1\,q_{ik}]$ ($i \in V$, $k \in B$).

All generated values are rounded to the closest integer. These instances, as well as those discussed in the next session, are available at `http://or.dei.unibo.it/library/`.

The outcome of the experiments is presented in Table 3. For each approach and time limit, the table reports the average percentage gap with respect to the best upper bound. The best values are given in bold. The results confirm those obtained for the original instance for what concerns small time limits: the solver is inadequate and **H-Poly** is the best approach. For larger time limits, there is some variation with respect to the previous picture: **ILP** and **M-Exp** are the best approaches, with gaps one order of magnitude smaller than those of **H-Poly**, and **ILP** is almost always the winner. Overall, **H-Poly** should be preferred for day to day operations and in cases where almost on-line solutions are needed, while **M-Exp** and **ILP** are the most robust approaches when a solver is available and larger solution times are allowed. The final upper bound value U^* is typically identical (or very close) to the value of the continuous relaxation (not reported in the table), that requires, on average, about 11 seconds.

6.3 Random instances

In order to further test the various algorithms, we use an additional benchmark of 50 random instances, generated with the same number of item types, bases, and vehicles as the original instance. All input data (ℓ_j, u_j, p_j, w_j, a_{kj}, c_i, and q_{ik} for $j \in T$, $k \in B$, and $i \in V$) are generated as integer values uniformly random in the intervals given by the corresponding minimum and maximum values in the real-world instance (see Table 1).

Table 4 reports the average percentage gaps for the three algorithms, with different time limits. The best values (in bold) confirm the previous indications. For small time limits, **H-Poly** is the best approach and **ILP** is unable to find a feasible solution. In addition, **ILP** can fail for $\texttt{TL} = 300$ as well: we thus report the average percentage gap over the successful cases

	REA-c			REA-p			REA-w		
TL	**ILP**	**H-Poly**	**M-Exp**	**ILP**	**H-Poly**	**M-Exp**	**ILP**	**H-Poly**	**M-Exp**
25	–	**4.88**	5.50	–	**4.71**	5.35	–	**4.93**	5.33
50	–	**4.49**	5.50	–	**4.48**	5.35	–	**4.73**	5.33
100	–	**4.02**	5.50	–	**3.49**	5.02	–	**4.05**	5.28
200	0.31	3.76	**0.24**	0.39	3.09	**0.30**	**0.36**	3.51	0.38
300	**0.17**	3.42	0.21	0.36	3.03	**0.29**	0.33	3.29	**0.26**
400	**0.14**	3.28	0.20	0.33	3.03	**0.28**	0.30	2.97	**0.25**
500	**0.13**	3.26	0.16	**0.28**	2.96	**0.28**	0.27	2.71	**0.24**
600	**0.13**	3.26	0.17	0.28	2.96	**0.27**	0.25	2.71	**0.24**
700	**0.13**	3.26	0.15	0.28	2.96	**0.24**	**0.23**	2.71	0.25
800	**0.13**	3.26	0.15	0.28	2.96	**0.22**	**0.23**	2.71	0.24
900	**0.13**	3.26	0.15	0.27	2.96	**0.22**	**0.23**	2.71	0.24
1000	**0.13**	3.26	0.16	0.27	2.96	**0.22**	**0.23**	2.71	0.25
	REA-b			REA-a			REA-q		
TL	**ILP**	**H-Poly**	**M-Exp**	**ILP**	**H-Poly**	**M-Exp**	**ILP**	**H-Poly**	**M-Exp**
25	–	**5.34**	7.76	–	**4.44**	**4.44**	–	**3.43**	5.10
50	–	**4.24**	7.76	–	**4.44**	**4.44**	–	**3.03**	5.10
100	–	**4.20**	6.67	–	**4.09**	4.44	–	**2.72**	4.77
200	0.41	3.52	**0.27**	–	3.88	**0.31**	–	2.72	**0.29**
300	0.34	3.28	**0.29**	0.29	3.52	**0.27**	0.35	2.72	**0.22**
400	0.29	2.65	**0.26**	0.26	3.52	**0.25**	0.26	2.65	**0.23**
500	**0.29**	2.59	0.30	**0.26**	3.52	**0.26**	0.22	2.65	**0.21**
600	**0.26**	2.59	0.28	0.25	3.52	**0.23**	**0.21**	2.65	**0.21**
700	**0.24**	2.59	0.29	0.25	3.52	**0.24**	**0.21**	2.65	0.22
800	**0.23**	2.59	0.28	**0.25**	3.52	0.26	**0.21**	2.65	0.22
900	**0.22**	2.59	0.27	**0.24**	3.52	0.26	**0.21**	2.65	0.22
1000	**0.22**	2.59	0.28	**0.24**	3.52	0.25	**0.21**	2.65	0.23

Table 3: Percentage gaps for the realistic instances. Average values over 5 instances.

and the corresponding number in parentheses. For the two largest time limits, **ILP** obtains the smallest gaps. Worth is noting that random instances appear to be more challenging than real and realistic cases, but the proposed heuristics always succeeds in finding a feasible solution. In this case, the solution of the continuous relaxation requires about 12.5 seconds on average.

7. Conclusions

We introduced the assignment and loading transportation problem, a generalization of the multiple knapsack problem that can arise in military and emergency contexts. We formally described it as an Integer Linear Program with a polynomial number of variables and constraints. The problem is strongly $\mathcal{NP}$-hard and difficult to solve in practice to proven optimality for real-size instances. We introduced Lagrangian and surrogate relaxations, a self-contained polynomial-time heuristic and a non-polynomial matheuristic approach that makes use of a MILP solver. We used a real-world military instance and a number of realistic and random instances to compare these approaches with the direct use of a MILP solver. The outcome of the experiments indicates that the polynomial-time algorithm is the

TL	ILP		H-Poly	M-Exp
25	–		**5.60**	6.36
50	–		**5.12**	6.36
100	–		**4.47**	6.21
200	–		3.81	**2.01**
300	–		3.56	**1.13**
400	1.16	(16)	3.40	**0.94**
500	0.82	(44)	3.37	**0.98**
600	0.66	(49)	3.37	**0.94**
700	**0.58**		3.37	0.83
800	**0.55**		3.37	0.86
900	**0.53**		3.37	0.87
1000	**0.50**		3.37	0.84

Table 4: Percentage gaps for the random instances. Average values over 50 instances (in parentheses, number of solutions found if less than 50).

best approach when the problem has to be solved very quickly, while the other methods are preferable when sufficient CPU time and a MILP solver are available. For all benchmarks, the proposed heuristics obtained, even within small time limits, feasible solutions of high quality (typically over 95% of the upper bound).

Acknowledgments

This research was supported by Air Force Office of Scientific Research (under award number FA9550-17-1-0067). We thank the United States Department of Defense for providing a real case study.

References

[1] İ. Akgün and B.Ç. Tansel. Optimization of transportation requirements in the deployment of military units. *Computers & Operations Research*, 34:1158–1176, 2007.

[2] M. Besiou, A.J. Pedraza-Martinez, and L.N. Van Wassenhove. OR applied to humanitarian operations. *European Journal of Operational Research*, 269:397–405, 2018.

[3] G. Dahl and N. Foldnes. LP based heuristics for the multiple knapsack problem with assignment restrictions. *Annals of Operations Research*, 146:91–104, 2006.

[4] M. Dawande, J. Kalagnanam, P. Keskinocak, F.S. Salman, and R. Ravi. Approximation algorithms for the multiple knapsack problem with assignment restrictions. *Journal of Combinatorial Optimization*, 4:171–186, 2000.

[5] L.E. de la Torre, I.S. Dolinskaya, and K.R. Smilowitz. Disaster relief routing: Integrating research and practice. *Socio-Economic Planning Sciences*, 46:88–97, 2012.

[6] J.E. Diaz, J. Handl, and D.-L. Xu. Integrating meta-heuristics, simulation and exact techniques for production planning of a failure-prone manufacturing system. *European Journal of Operational Research*, 266:976–989, 2018.

[7] N.B. Dimitrov, D. Solow, J. Szmerekovsky, and J. Guo. Emergency relocation of items using single trips: Special cases of the multiple knapsack assignment problem. *European Journal of Operational Research*, 258:938–942, 2017.

[8] E. Gralla and J. Goentzel. Humanitarian transportation planning: Evaluation of practice-based heuristics and recommendations for improvement. *European Journal of Operational Research*, 269:436–450, 2018.

[9] J.D. Jordan and J.D. Weir. Average longest path and maximum cost network flows with multiple-criteria weights. *Electronic Notes in Discrete Mathematics*, 69:181–188, 2018.

[10] S. Kataoka and T. Yamada. Upper and lower bounding procedures for the multiple knapsack assignment problem. *European Journal of Operational Research*, 237:440–447, 2014.

[11] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, 2004.

[12] Y. Laalaoui and R. M'Hallah. A binary multiple knapsack model for single machine scheduling with machine unavailability. *Computers & Operations Research*, 72:71–82, 2016.

[13] E. Lalla-Ruiz and S. Voß. *A Biased Random-Key Genetic Algorithm for the Multiple Knapsack Assignment Problem*, pages 218–222. Springer International Publishing, Cham, 2015.

[14] S. Martello and M. Monaci. Algorithmic approaches to the multiple knapsack assignment problem. *Omega*, 2018. (to appear).

[15] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, 1990.

[16] N. McCarthy. The cost of the air war against ISIS has reached $11 billion. *Forbes*, 2017. https://www.forbes.com/sites/niallmccarthy/2017/02/01/the-cost-of-the-air-war-against-isis-has-reached-11-billion-infographic/#4f2de1b5b120.

[17] N. McCarthy. The cost of the UK's air war in Syria & Iraq. *Statista*, 2018. https://www.statista.com/chart/13043/the-cost-of-the-uks-air-war-in-syria-iraq/.

[18] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience, Chichester, 1988.

[19] Office of Secretary of Defense. Classes of supply. https://www.acq.osd.mil/dpap/ccap/cc/jcchb/Files/Topical/Supplies_files/resources/table_2_supply_classes.pdf. Accessed: 2019-05-02.

[20] L. Shane, III. Report: Full cost of U.S. wars overseas approaching $6 trillion. *Military Times*, 2017. https://www.militarytimes.com/news/pentagon-congress/2017/11/08/report-full-cost-of-us-wars-overseas-approaching-6-trillion/.

[21] J. Simon, A. Apte, and E. Regnier. An application of the multiple knapsack problem: The self-sufficient marine. *European Journal of Operational Research*, 256:868–876, 2017.

[22] United States Transportation Command. Fiscal year 2018 financial report. https://comptroller.defense.gov/Portals/45/Documents/afr/fy2018/DoD_Components/2018_AFR_USTRANSCOM.pdf. Accessed: 2019-05-02.

[23] L.M. Williams and V.C. Gay. Defense primer: U.S. Transportation Command (TRANSCOM), report March 6, 2018. *CRS In Focus*, 2018. https://fas.org/sgp/crs/natsec/IF10840.pdf.

[24] L. Zhen, K. Wang, S. Wang, and X. Qu. Tug scheduling for hinterland barge transport: A branch-and-price approach. *European Journal of Operational Research*, 265:119 – 132, 2018.

[25] L. Zheng and J.L. Zhou. Optimization of military transportation path based on generalized cut set algorithm. *Computer Engineering*, 33:4–9, 2007.