Array 8 (2020) 100047

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/journals/array/2590-0056/open-access-journal

Tabu search-based fleet scheduling of air ambulances for disaster response

Joseph Tassone *, Salimur Choudhury

Lakehead University, 955 Oliver Road, Thunder Bay, ON, P7B 5E1, Canada

ARTICLE INFO

Keywords:
Air transportation
CUDA
Fleet scheduling
Integer linear programming
Local search
Tabu search

ABSTRACT

Proper scheduling of air assets can be the difference between life and death for a patient. While poor scheduling can be incredibly problematic during hospital transfers, it can be potentially catastrophic in the case of a disaster. These issues are amplified in the case of an air emergency medical service (EMS) system where populations are dispersed, and resources are limited. There are exact methodologies existing for scheduling missions, although actual calculation times can be quite significant given a large enough problem space. For this research, known coordinates of air and health facilities were used in conjunction with a formulated integer linear programming model. This was the programmed through Gurobi so that performance could be compared against custom algorithmic solutions. Two methods were developed, one based on neighbourhood search and the other on Tabu search. While both were able to achieve results quite close to the Gurobi solution, the Tabu search outperformed the former algorithm. Additionally, it was able to do so in a greatly decreased time, with Gurobi being unable to resolve to optimal in larger examples. Parallel variations were also developed with the compute unified device architecture (CUDA), though did not improve the timing given the smaller sample size.

1. Introduction

Disasters can strike at any time and in these scenarios, it can be devastating if there is not an efficient emergency medical service (EMS) system. While this can be problematic in a city-based environment, it is catastrophic when considering a larger air ambulance system. These vehicles travel over much longer distances, through regions with very dispersed populations. While patient transfers may occur between populous areas, this is not guaranteed during a disaster. Additionally, many air ambulances have a rather small contingency [1] which must cover all missions within a certain time frame to be effective. It is critical to accurately schedule vehicles in such a way that these potentially vast distances can be managed, and patient lives can be secured.

This work is a continuation of prior research which sought to optimize the staging and positioning of air assets [2]. Therefore, despite it being an independent methodology, it is also part of a greater problem directed towards air ambulance management. This is not necessarily new, as there have been several exact methodologies suggested previously [3]. Though solving for optimal is not always unreasonable, since scheduling is NP-hard, it becomes much more time consuming with an increase in the solution space. The issue with this is compounded in a critical system like EMS where time can mean the difference between life and death. Near-optimal metaheuristics provide an effective alternative as they achieve acceptable solutions in a much more manageable window. In this research, known coordinates of air and health facilities were utilized and aided in simulating an actual scheduling scenario. A mission in this case always began at a vehicle occupied base and returned to the same base. Subsets of missions could be performed by each vehicle with a respective mission performing a pickup and delivery before continuing to the next point. Two algorithmic solutions were developed for this problem and each was structured using a sequential and parallel methodology. For the latter, the Compute Unified Device Architecture (CUDA) platform was employed and timed against the former for each algorithm. Overall, the goal was to develop a model with integer linear programming and then formulate algorithms which could attain a similar solution to the optimal.

This paper is arranged as follows. Section 2 describes the related work with a description of past techniques and solutions. Section 3 outlines the modelling of the problem, emphasizing constraints and the objective. Section 4 displays the algorithmic solutions, describing the neighbourhood and tabu search, as well as the initialization technique. Sections 5 and 6 respectively show the results and conclude the paper.

2. Related work

For scheduling problems, neighbourhood searches have been well-

* Corresponding author..
E-mail address: jtasson2@lakeheadu.ca (J. Tassone).

https://doi.org/10.1016/j.array.2020.100047
Received 21 July 2020; Received in revised form 15 October 2020; Accepted 27 October 2020
Available online 11 November 2020

documented in achieving successful results. Dağlayan and Karakaya suggested a solution for scheduling ambulances following a disaster and formulated it as a capacitated vehicle routing problem (CVRP) [4]. In their research, they minimized the routes and then reduced the average travel time to health facilities. For this process they developed both a genetic algorithm and nearest-neighbour heuristic. Both operated independently and were compared in three different test cases. These were based on capacity where light damage had 1 victim, medium damage could have 1 to 8 victims, and heavy damage had beyond 8. While the authors claimed that further study was required, when compared against CVRP benchmarks the genetic algorithm achieved shorter distances. This work does require additional testing, as there is a limitation in utilizing a single facility and ambulance.

Tabu search is another successful algorithm in scheduling, being an extension of the neighbourhood-based algorithms. In this type of solution recently explored neighbourhoods which improved the result are ignored, preventing the risk of achieving only a local optimum [5]. Oberscheider and Hirsch utilized this methodology in conjunction with real-patient data to ensure the efficient transport of patients for lower Austria's Red Cross [6]. They developed an algorithm based on the static multi-depot heterogeneous dial-a-ride problem and generated all possible combinations of patient deliveries considering a set of constraints to solve it. For each prior generation, set partitioning was performed to achieve an initial solution and then these were inputted into a Tabu search metaheuristic for optimization. The authors claim that the work is an improvement upon previous variations as it considers non-static service times that depended on the combination of patients, their transport mode, the vehicle type, and the pickup or delivery locations.

Another research presented by Repoussis et al. formulated a mixed integer linear programming model for the scheduling and routing of large casualty disasters [7]. They sought to use a minimized amount of resources while also being able to achieve a reduced response time. Their solution was developed as a hybrid multi-start local search, involving an exact construction heuristic followed by an optimization by iterated Tabu search. Initialization utilized a greedy technique for assignments and then the problem in its reduced state was optimized. A high-quality upper bound was achieved and these were inputted into the Tabu search, repeating until the stopping condition was met.

Most solutions involve sequential designs where a solution must iteratively solve until reaching an optimal. While not a new trend, parallelization has been utilized to achieve higher calculation speeds through the graphics processing unit [8]. The algorithms must be redeveloped to take advantage of the multiple simultaneous threads and loops are replaced by these to perform calculations. Additionally, threads are divided into blocks which can be an issue in larger problems. In this case thread communication occurs within blocks, but synchronization cannot happen between them. For these scenarios block level design must be considered along with thread design. Regardless, the primary bottleneck is related to the kernel call to GPU when the problem space is small enough. Race conditions are another potential issue, though CUDA provides both shared memory and locks to deal with this problem [9].

While prior parallel research has been performed, it is limited in terms of optimization problems of this type. Schulz et al. performed a survey revealing a small amount of literature on applying GPU parallelization to local or Tabu search [10]. The same can be said when applied to regional or air ambulance problems. Hussai et al. redeveloped the particle swarm optimization algorithm by using CUDA to perform partially coalescing memory accesses [11]. When compared against benchmarks the new algorithm was able to gain a significant improvement in time compared to a similar sequential variation. Similarly, Fabris and Krholing used benchmarks and applied CUDA to a co-evolutionary differential evolution algorithm for solving min-max optimization problems [12]. Their research showed that the algorithm was able to converge to a near optimal result, although in a much better time when scaled up.

3. Model

Scheduling of air ambulances follows predetermined positions, where ambulances are already placed at bases. Time is also a factor as scheduling relies on patients arriving at certain periods. This may be due to urgency from an incident or transfers for more specialized care. While there are potentially many bases in an EMS system, this scheduling problem only deals with the subset of bases that have vehicles assigned to them. In general, unless otherwise specified, the scheduling problem treats the terms vehicles and bases as synonymous.

In this case a mission starts at a base and can be made up of at least a single pickup and delivery. However, rather than a simple return to base, another mission may be assigned to that vehicle, where it will continue from the previous delivery to next pickup. Therefore, the schedule will consist of each base being assigned a set of missions, followed by a return to the original base. An example of this problem and all possible setups can be seen in Fig. 1. There are three bases, with four missions. So long as the constraints are met, each base can have either as many missions as possible assigned or none. Essentially, this means a base can be assigned one or more missions up to the maximum which require assignment.

The potential bases consist of the following two sets:

P: the set of all fixed-wing plane occupied bases with each being a 4-tuple of form
$a_k =< k\ s\ \phi_k\ \psi_k >, a_k \in P\ \forall\ k = 1, \ldots, 4$
$k \equiv$ base ID
$s \equiv$ vehicle speed
$\phi_k \equiv$ row coordinate of base k
$\psi_k \equiv$ column coordinate of base k

H: the set of rotary-wing helicopter occupied bases with each being a 4-tuple of form.

$h_k =< k\ s\ \phi_k\ \psi_k >, h_k \in H\ \forall\ k = 5, \ldots, 12$
$k \equiv$ base ID
$s \equiv$ vehicle speed
$\phi_k \equiv$ row coordinate of base k
$\psi_k \equiv$ column coordinate of base k

The set of all missions consists of:
M: the set of all missions, with each being a 6-tuple of form.

$m_n =< n\ \phi_p\ \psi_p\ \phi_d\ \psi_d\ \rho >, m_n \in M\ \forall\ n$
$n \equiv$ mission ID
$\phi_p \equiv$ row coordinate of patient pick-up location
$\psi_p \equiv$ column coordinate of patient pick-up location
$\phi_d \equiv$ row coordinate of patient delivery location
$\psi_d \equiv$ column coordinate of patient delivery location
$$\rho = \begin{cases} 1 & \text{if mission requires rotary} - \text{wing helicopter} \\ 0 & \text{otherwise} \end{cases}$$

Sets H and P are vehicle occupied bases, where both are constant and locked to their specific coordinates. No other bases can be utilized and following a set of missions there must be a return to the original base. Both sets are parts of the full set of bases which can be utilized, stated as $P \cup H = K$. Additionally, rather than total distance, the travel time from one position to another is considered. Helicopters and planes travel at different speeds, meaning that regardless of distances they will arrive at different times. Therefore, speed must be considered for each set to perform accurate scheduling. Missions have specific requirements where only a helicopter occupied base may perform certain missions.

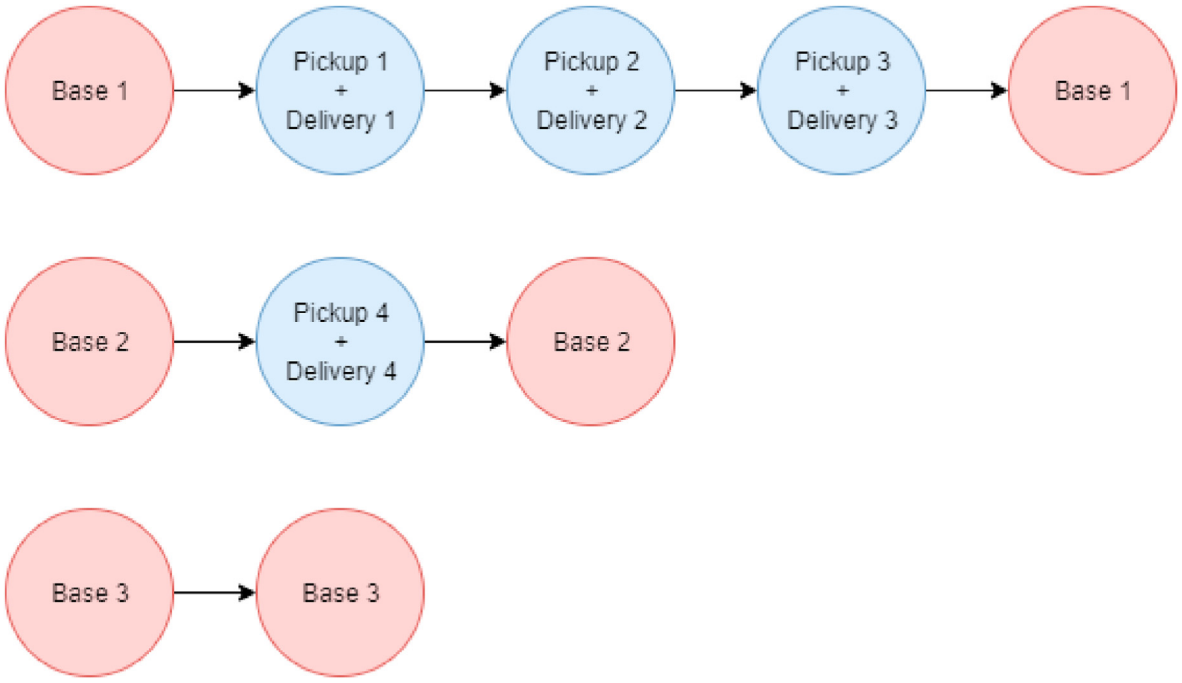

Fig. 1. Basic scheduling setup for missions and bases.

Table 1
Summary of scheduling model variables.

Variable	Description
$xijk$	A binary decision variable determining whether a mission i to j are connected, while starting and ending at base k
$dijl$	Time matrix where the distance from i to j is divided by the speed of the vehicle of type l
p	Constant stating the maximum flight time allowed by each vehicle per day
w_n	Constant stating the time by which a vehicle must arrive at mission n by
f_n	Constant stating the vehicle requirement for a mission n (0 for helicopter-only and 1 for any vehicle)
b_k	Constant stating the vehicle situated at a base k (0 for helicopter and 1 for airplane)
u_i	Integer decision variable for the exclusion of subtours, the value is a ranked order where a value $i = \{1, \ldots, n\}$ for each base k

There are a set of variables for the problem with a summary show in Table 1. x_{ijk} is binary, where an instance will be 1 if a mission i to j are connected and starting from base k. Additionally, the variable u_i is utilized for subtour elimination based on the Miller-Tucker-Zemlin constraint [13]. For the exclusion of subtours the value can be $i = \{1, \ldots, n\}$ for each base k. The model also considers a set of constants p, w_n, b_k, and f_n. Both p and w_n contribute to time constraints where p is the maximum flight time allowed by a vehicle in a day (10 h for the purposes of this research) and w_n is the time by which a vehicle must arrive at a base by. In the scheduling problem they are independent as once a vehicle arrives for a mission it must wait until its respective w_n to continue to the next point. The constant p only considers actual time in flight and does not recognize waiting. Both b_k and f_n are utilized for limiting certain bases to certain missions. b_k states the type of vehicle that is at an individual base (0 for helicopter and 1 for plane), while f_n describes the vehicle requirement that must be used for a mission (0 for helicopter-only and 1 for any vehicle).

In addition to distance calculations, the respective travel time is considered as determined by the speed of the vehicle. Based on known statistics about air ambulance vehicles, rotary-wing helicopters are assigned a speed of 300 km/h [14] and fixed-wing planes are assigned a speed of 500 km/h [15]. The matrix d is the result of the calculation, where the distance from i to j is determined and divided by the speed of the vehicle. The respective result is then placed into l where l_0 is for helicopters and l_1 is for planes. The distance is determined for a base as the sum of each mission's patient pick-up and delivery locations. Since this is not simulated data, Haversine distance is considered. The full calculation can be completed with the following:

$$D_{ij} = 2r\sin^{-1}\left(\sqrt{\sin^2\left(\frac{\varphi-\varphi_p}{2}\right)+\cos(\varphi)\cos(\varphi_p)\sin^2\left(\frac{\psi-\psi_p}{2}\right)}\right) \quad (1)$$

In the above formula ϕ represents row locations, while ψ applies to column location values. K_s is used to indicate a specific speed of a vehicle at a base, which is divided by the Haversine distance calculated. The resulting matrix is of dimension $(n+k) \times (n+k) \times 2$ and referenced for achieving total travel time for each mission. The optimization model takes the following form:

minimize

$$\sum_i \sum_j \sum_k x_{ijk} d_{ijl}; l = b_k \quad (2)$$

subject to

$$\sum_i \sum_k x_{ijk} =; \forall j \quad (3)$$

$$\sum_j \sum_k x_{ijk} =; \forall i \quad (4)$$

$$\sum_i x_{ink} = \sum_j x_{njk}; \forall k, n \quad (5)$$

$$\sum_i x_{ijk} \leq 1; \forall j,\ k \quad (6)$$

$$\sum_j x_{ijk} \leq 1; \forall i,\ k \quad (7)$$

$$\sum_i \sum_j x_{ijk} d_{ijl} \leq \mathrm{p}; \forall k, l = b_k \quad (8)$$

$$x_{ijk} d_{ijl} \leq x_{ijk} w_j; \forall\, j,k,l = b_k, i \leq n \tag{9}$$

$$x_{ijk}\left(d_{ijl} w_i\right) \leq x_{ijk} w_j; \forall\, j,k,l = b_k, i > n \tag{10}$$

$$x_{ijk}(b_k - f_i) \leq 0\,; \forall\, i,j,k \tag{11}$$

$$u_i - u_j + n\left(x_{ijk}\right) \leq n-1\,; 1 \leq i \neq j \leq n,; \forall\, k \tag{12}$$

$$x_{ijk} = 0\,; (i \neq j,k) \in K \tag{13}$$

$$x_{ijk} \in \{1,\ 0\}; \forall\, i,j,k \tag{14}$$

$$u_i \in \{i = 1, \ldots, n\} \tag{15}$$

Equation (2) describes the objective function, where the total time flown by each vehicle to its assigned missions are minimized. Each set's d_{ijl} is summed based on its corresponding x_{ijk} variables which have a 1. The actual distance is determined by the constant b_k, where the index of l equals the respective value. The objective function is also constrained based on the rules set by Equations (3)–(15).

Each mission can only be completed once, constrained by Equations (3) and (4). Equation (5) guarantees route continuity, where each mission must continue from one to the other along their assigned route. To ensure that missions leave and return to the same base, the model considers Equations (6) and (7). As previously stated, vehicles can only fly so long which determined by p. The constraint for this is upheld by Equation (8) for each base k set of missions. The remaining time constraints are determined by Equations (9) and (10). The first is for when a vehicle is travelling from a base to a mission. In this case the base does not have a time limit and begins at 0. The second considers either a mission to a mission, or a mission returning to a base. In the latter case the limit is usually assumed to be correspond to the number of hours in a day. Regardless of travel time the vehicle must wait until that mission's time limit to travel to the next one. The distance summed with the previous time limit must be less than the next mission's time limit.

To force each mission to be correctly assigned to a valid base, the model uses Equation (11). There is only one invalid permutation for this assignment, where a plane is assigned to a helicopter-only mission. This is prevented as all other possibilities are forced to be less than or equal to 0. The subtour elimination constraint is outlined by Equation (12) and uses the Miller-Tucker-Zemlin formulation. For this constraint, u_i is a variable associated with each mission and is used to eliminate sets that do not begin and end at a base. The variable is equal to the order in which each mission occurs for each base k, where the base begins at the first index [16]. To further limit the formulation, Equation (12) prevents bases from travelling to other bases other than themselves. Lastly, Equations (13) and (14) limit the values which can be assigned to each variable.

4. Algorithm

All the solutions within this section were designed with consideration of the constraints and variables in the scheduling model above. The algorithms each had a sequential and parallel CUDA implementation. As there were only minor changes within the algorithms, Algorithm 2 and 3 are outlined as the sequential versions. Any other alterations are given following a description of each one.

4.1. Scheduling initialization algorithm

Due to the constraints, there is no guarantee that a randomized initialization will be able to generate a valid solution. Since vehicles must fly under a certain period and within a particular limit there is an increased difficulty in discovering a starting state. Therefore, Algorithm 1 was developed to find an initialized organization for optimization. To clarify, the algorithm is not designed to find a near-optimal solution, only to determine the existence of any solution. Essentially, the method will attempt to assign missions to bases and upon failure will perform swaps. This will improve the current result and may increase the chances of finding a valid permutation. If after so many attempts, it does not find a solution it will report an error.

The algorithm accepts a set of occupied bases and a set of mission data as its input. Lines 1–4 show the constant data that must be abided by including the distance matrices for each respective vehicle type, the mission times determining when a vehicle must arrive by, and the daily flight limit for a vehicle. The *Solution* matrix declared at line 5 will hold each mission with a row starting and ending with a base, and compatible missions will then fill in between these indices. The remainder of the algorithm is broken up into a section for helicopter-only mission assignments (lines 6–42) and then a section for the remaining missions (lines 43–77). For both, the outer loops will only terminate once all are assigned.

The first step at line 8 is to determine the unassigned mission with the smallest time limit. There is an urgency for this to be completed first as the time limit constraint cannot be broken. As per line 9, every base will be checked, and the current best fit will be held in *CurrentMin*. The next set of lines follow the constraints set by the model, limiting which bases can accept a mission. If a base breaks any of the requirements, then the algorithm continues to the next possible base. Lines 10–12 ensure that only a helicopter occupied base may accept a helicopter-only mission. Lines 13–34 are the time limit constraints, ensuring that vehicle does not pass the arrival time set for a respective mission. This part is broken into two parts based on whether a base already has missions or has yet to have one assigned to it. As the mission is being placed between two indices it must consider the vehicle arriving at the new mission and the travel new mission to the next one. Additionally, once a vehicle arrives at a mission it must wait until that time limit to move on to the following. If it is starting at a base the limit is 0 and not considered, while if it from another mission it must utilize the distance to the next plus the previous time limit. If the vehicle is returning to a base then the time cannot pass a predetermined maximum limit, currently expressed as the number of hours in the day. Lines 29–31 performs a check on the current flight time of each set of missions. This constraint only considers actual flight time and not the waiting enforced by the previous constraints. Once these restrictions have been met, lines 32–34 check if the new assignment is an improvement over the prior. The result is saved if there has yet to be an update to *CurrentMin* or if the new value is smaller than the previous.

If *CurrentMin* has a solution, then it is added to the *Solution* matrix at line 40. There is the possibility that following the loop there will be no available positions to place the mission. If this occurs, then lines 36–37 attempt a reorganization to find a new valid permutation. These swaps are performed by Algorithm 2 and discussed below. The algorithm returns an error if no reorganization is possible, no missions have been assigned, or following reorganization the assignment fails.

Lines 43–77 perform nearly the exact same process, with the exception that there is no mission compatibility requirement. To validate the correct matrix to choose from, the algorithm includes a flag called *VehicleType* (line 47). Each remaining check performs almost identically to the last section; however, the flag is used to ensure the corresponding distance matrix matches the vehicle at each base.

The addition of parallelism to this algorithm was simplistic, as race conditions were only an issue for *CurrentMin*. CUDA operates through concurrent thread organization, so the loops at lines 7 and 44 could instead be replaced with their own threads. The continues are also replaced as each thread controls its own base. This allowed the *MinIndex* to be checked simultaneously for each base. A mutex check was performed at line 32 to add to *CurrentMin* along with the thread ID. At line 40, if the thread ID matched the *CurrentMin*, then it updated it with its solution. If there was currently no updated solution, a thread would be allocated to performing the swaps.

4.2. Neighbourhood search fleet scheduling optimization

Algorithm 2 accepts either the initialized or the partially complete data from Algorithm 1. The goal of the algorithm is to find an organization of missions and bases which minimizes the total travel time. As already discussed, the design of the previous algorithm was not to find an optimal solution, only a valid one. Conversely, Algorithm 2 was formulated to be performed through multiple instances to minimize the total time to near optimal. Though the initialization method uses this algorithm, it only performs a single iteration to attempt a new result. To ensure that Algorithm 2 has a diverse set of possibilities, the loops at lines 6 and 7 may be exchanged with random index permutation vectors.

The algorithm functions by moving missions to new bases, essentially having another vehicle take responsibility for that mission. It will only end once no further improvement is found. If utilizing a set of permutation vectors, this means no better result is found in the current permutation and provides the opportunity for another iteration. A similar set of constraints from the previous algorithm are used and updates are performed to the *Solution* matrix. There are four sets of loops with line 6 allowing for looping through the bases and line 7 for looping through the respective missions assigned to a base. The loops at lines 9 and 17 perform a similar function, though they are the corresponding positions being move to.

Algorithm 1: Scheduling Initialization Algorithm

```
   Data: Base, mission pickup, and mission destination data
   Result: Assignment of missions to bases
 1 HeliDistances ← distance matrix helicopter bases to each respective mission and each
     mission to each other mission;
 2 PlaneDistances ← distance matrix plane bases to each respective mission and each
     mission to each other mission;
 3 MissionTimes ← times by which a vehicle must arrive at a particular mission delivery
     point by;
 4 FlightLimit ← an integer stating the amount of allowed flight time per day;
 5 MaximumTimeLimit ← the final time where a vehicle must return to base by;
 6 Solution ← a matrix initialized with each base in the first and final index for each row;
 7 while all helicopter only missions are not assigned do
 8 |   for i ← 1 to number of bases do
 9 |   |   MinIndex ← current unassigned helicopter only mission with the smallest value
     |   |     in MissionTimes;
10 |   |   CurrentMin ← current best i base for MinIndex;
11 |   |   if base i does not contain a helicopter then
12 |   |   |   Continue to the next i base;
13 |   |   end
14 |   |   if base i does not have any missions then
15 |   |   |   if HeliDistances from base i to MinIndex > MissionTimes of
     |   |   |     MinIndex then
16 |   |   |   |   Continue to the next i base;
17 |   |   |   end
18 |   |   |   if MissionTimes of MinIndex + HeliDistances from MinIndex to base
     |   |   |     i > MaximumTimeLimit then
19 |   |   |   |   Continue to the next i base;
20 |   |   |   end
21 |   |   end
22 |   |   else if base i does have missions assigned to it then
23 |   |   |   if MissionTimes of previous assigned base i mission +HeliDistances
     |   |   |     from previous assigned base i mission to MinIndex > MissionTimes of
     |   |   |     MinIndex then
24 |   |   |   |   Continue to the next i base;
25 |   |   |   end
26 |   |   |   if MissionTimes of MinIndex + HeliDistances from MinIndex to base
     |   |   |     i > MaximumTimeLimit then
27 |   |   |   |   Continue to the next i base;
28 |   |   |   end
29 |   |   end
30 |   |   if assigning MinIndex mission to base i exceeds FlightLimit then
31 |   |   |   Continue to the next i base;
32 |   |   end
33 |   |   if CurrentMin is empty or the new i gives a better result then
34 |   |   |   Update CurrentMin;
35 |   |   end
36 |   end
37 |   if CurrentMin is empty then
38 |   |   Perform mission swaps from Algorithm 2 and retry for new arrangement
39 |   end
40 |   else
41 |   |   Update Solution with CurrentMin values;
42 |   end
43 end
```

```
43 while all remaining missions are not assigned do
44 |  for i ← 1 to number of bases do
45 |  |  MinIndex ← current unassigned mission with the smallest value in
   |  |    MissionTimes;
46 |  |  CurrentMin ← current best i base for MinIndex;
47 |  |  VehicleType ← flag determining whether to select from HeliDistances or
   |  |    PlaneDistances;
48 |  |  if base i does not have any missions then
49 |  |  |  if VehicleType type from base i to MinIndex > MissionTimes of
   |  |  |    MinIndex then
50 |  |  |  |  Continue to the next i base;
51 |  |  |  end
52 |  |  |  if MissionTimes of MinIndex + VehicleType type from MinIndex to
   |  |  |    base i > MaximumTimeLimit then
53 |  |  |  |  Continue to the next i base;
54 |  |  |  end
55 |  |  end
56 |  |  else if base i does have missions assigned to it then
57 |  |  |  if MissionTimes of previous assigned base i mission +VehicleType type
   |  |  |    from previous assigned base i mission to MinIndex > MissionTimes of
   |  |  |    MinIndex then
58 |  |  |  |  Continue to the next i base;
59 |  |  |  end
60 |  |  |  if MissionTimes of MinIndex + VehicleType type from MinIndex to
   |  |  |    base i > MaximumTimeLimit then
61 |  |  |  |  Continue to the next i base;
62 |  |  |  end
63 |  |  end
64 |  |  if assigning MinIndex mission to base i exceeds FlightLimit then
65 |  |  |  Continue to the next i base;
66 |  |  end
67 |  |  if CurrentMin is empty or the new i gives a better result then
68 |  |  |  Update CurrentMin;
69 |  |  end
70 |  end
71 |  if CurrentMin is empty then
72 |  |  Perform mission swaps from Algorithm 2 and retry for new arrangement;
73 |  end
74 |  else
75 |  |  Update Solution with CurrentMin values;
76 |  end
77 end
```

. (*continued*).

Like Algorithm 1, it includes a *CurrentMin* variable for keeping track of the best current position for swapping. Once every valid combination is checked, the *CurrentMin* updates the *Solution* at line 43. Swapping within a base is restricted by lines 10–12, as the current order is already

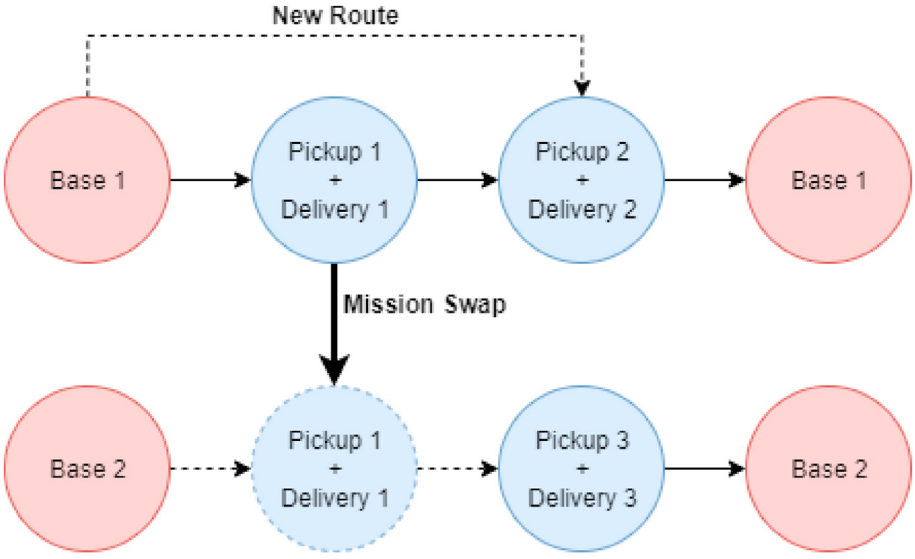

Fig. 2. The movement of one mission to another base.

constrained by the time. Additionally, certain planes cannot accept helicopter-only missions as stated in lines 13–15. There is a *VehicleType* flag for determining which matrix to select from for the remainder of the algorithm. This is based on the specific vehicle that is assigned.

An example of the process can be seen in Fig. 2, where the first mission connected to base 1 is moved to base 2. Once a swap occurs base 1 connects directly to mission 2, while the base 2 setup adds two new links between the new position of mission 1. During the movement, the new position of the mission cannot force the vehicle's flight limit to be exceed, expressed by lines 18–20.

Similarly, the new links must not exceed any of the time limits set by the lines 21–36. This function is identical to the design of the previous algorithm where the check depends on where the mission is being moved to. Again, a vehicle must wait until the current mission's time limit before proceeding, meaning each proceeding time limit must be considered in the check. If the movement passes these requirements and is the current best option, then it is added to *CurrentMin* at line 38.

Like the previous algorithm, this one can be made easily parallel by the removal of the loop at line 9. This allows every base to be simultaneously checked and for the *CurrentMin* to be updated with the best result quicker. The outer loop at line 6 could have instead been replaced, though this would have given a different value than the original, making accurate comparison difficult. Either design can be run multiple times and so long as an index ordering is done the solution may be improved.

Algorithm 2: Neighbourhood Search Fleet Scheduling

```
   Data: Initial Solution Data From Algorithm 1
   Result: Assignment of missions to bases
 1 HeliDistances ← distance matrix helicopter bases to each respective mission and each
     mission to each other mission;
 2 PlaneDistances ← distance matrix plane bases to each respective mission and each
     mission to each other mission;
 3 MissionTimes ← times by which a vehicle must arrive at a particular mission delivery
     point by;
 4 FlightLimit ← an integer stating the amount of allowed flight time per day;
 5 MaximumTimeLimit ← the final time where a vehicle must return to base by;
 6 while improvement do
 7 |   for i ← 1 to number of bases do
 8 |   |   for j ← 1 to number of i assigned missions do
 9 |   |   |   CurrentMin ← current best swap position;
10 |   |   |   for k ← 1 to number of bases do
11 |   |   |   |   if i = k then
12 |   |   |   |   |   Continue to next k;
13 |   |   |   |   end
14 |   |   |   |   if vehicle at base k is not compatible with mission j then
15 |   |   |   |   |   Continue to next k;
16 |   |   |   |   end
17 |   |   |   |   VehicleType ← flag determining whether to select from HeliDistances
     |   |   |   |     or PlaneDistances;
18 |   |   |   |   for l ← 1 to number of k assigned missions do
19 |   |   |   |   |   if assigning j mission to base k exceeds FlightLimit then
20 |   |   |   |   |   |   Continue to the next k base;
21 |   |   |   |   |   end
22 |   |   |   |   |   if base k does not have any missions then
23 |   |   |   |   |   |   if VehicleType type from base k to mission
     |   |   |   |   |   |     j > MissionTimes of j then
24 |   |   |   |   |   |   |   Continue to the next k base;
25 |   |   |   |   |   |   end
26 |   |   |   |   |   |   if MissionTimes of j + VehicleType type from j to base
     |   |   |   |   |   |     k > MaximumTimeLimit then
27 |   |   |   |   |   |   |   Continue to the next k base;
28 |   |   |   |   |   |   end
29 |   |   |   |   |   end
30 |   |   |   |   |   else if base k does have missions assigned to it then
31 |   |   |   |   |   |   if MissionTimes mission l + VehicleType type of l to
     |   |   |   |   |   |     j > MissionTimes of j then
32 |   |   |   |   |   |   |   Continue to the next k base;
33 |   |   |   |   |   |   end
34 |   |   |   |   |   |   if MissionTimes of j + VehicleType type from j to mission
     |   |   |   |   |   |     after l > MissionTimes of mission after l then
35 |   |   |   |   |   |   |   Continue to the next k base;
36 |   |   |   |   |   |   end
37 |   |   |   |   |   end
38 |   |   |   |   |   if CurrentMin is empty or the new assignment gives a better
     |   |   |   |   |     result then
39 |   |   |   |   |   |   Update CurrentMin;
40 |   |   |   |   |   end
41 |   |   |   |   end
42 |   |   |   end
43 |   |   |   if CurrentMin is not empty then
44 |   |   |   |   Update Solution with result in CurrentMin;
45 |   |   |   end
46 |   |   end
47 |   end
48 end
```

4.3. Tabu search fleet scheduling optimization

This algorithm functions as a modification of Algorithm 2, with a near identical design. The primary differences are the variables included at lines 5 and 6. The *TabuList* prevents recently explored neighbourhoods from being explored again. There is a chance that the algorithm may become locked into a local minimum, preventing optimum exploration of the space. Therefore, the *TabuList* helps to prevent this, keeping certain regions restricted for several iterations based on the *TabuCounter*.

During the iterative process if a region which has recently been explored is already in the list, then it is skipped as per lines 20–27. The respective counter for each value is then reduced at the end of the current iteration with line 56. Additionally, a value is added to *TabuList* only if improves the value determined by lines 52–55. If permutation vectors are used for the outter loops, then improvement may be discovered through multiple runs. In this case the *TabuList* variable is maintained through proceeding iterations.

The parallel implementation is identical to Algorithm 2. The inner loop at line 11 is replaced and continues are removed as each base receives its own thread. Also like the previous, Algorithm 3 will result in the exact same answer as the sequential variation. The updates are still handled by a single thread, which is determined by the best result in *CurrentMin*.

Algorithm 3: Tabu Search Fleet Scheduling

```
   Data: Initial Solution Data From Algorithm 1
   Result: Assignment of missions to bases
 1 HeliDistances ← distance matrix helicopter bases to each respective mission and each mission to each
     other mission;
 2 PlaneDistances ← distance matrix plane bases to each respective mission and each mission to each
     other mission;
 3 MissionTimes ← times by which a vehicle must arrive at a particular mission delivery point by;
 4 FlightLimit ← an integer stating the amount of allowed flight time per day;
 5 MaximumTimeLimit ← the final time where a vehicle must return to base by;
 6 TabuList ← list of recently explored neighbourhoods;
 7 TabuCounter ← how long a neighbourhood can be held within a list;
 8 while improvement do
 9     for i ← 1 to number of bases do
10         for j ← 1 to number of i assigned missions do
11             CurrentMin ← current best swap position;
12             for k ← 1 to number of bases do
13                 if i = k then
14                     Continue to next k;
15                 end
16                 if vehicle at base k is not compatible with mission j then
17                     Continue to next k;
18                 end
19                 VehicleType ← flag determining whether to select from HeliDistances or
                     PlaneDistances;
20                 for l ← 1 to number of k assigned missions do
21                     if Selection k or l are in the TabuList then
22                         if Selection is of an index k then
23                             Continue to next k index;
24                         end
25                         else
26                             Continue to next l index;
27                         end
28                     end
29                     if assigning j mission to base k exceeds FlightLimit then
30                         Continue to the next k base;
31                     end
32                     if base k does not have any missions then
33                         if VehicleType type from base k to mission j > MissionTimes of j then
34                             Continue to the next k base;
35                         end
36                         if MissionTimes of j + VehicleType type from j to base
                             k > MaximumTimeLimit then
37                             Continue to the next k base;
38                         end
39                     end
40                     else if base k does have missions assigned to it then
41                         if MissionTimes mission l + VehicleType type of l to j > MissionTimes
                             of j then
42                             Continue to the next k base;
43                         end
44                         if MissionTimes of j + VehicleType type from j to mission after
                             l > MissionTimes of mission after l then
45                             Continue to the next k base;
46                         end
47                     end
48                     if CurrentMin is empty or the new assignment gives a better result then
49                         Update CurrentMin;
50                     end
51                 end
52             end
53             if CurrentMin is not empty then
54                 Update Solution with result in CurrentMin;
55                 Add selection to the TabuList and initiate TabuCounter for the selection;
56             end
57             Reduce the TabuCounter for each within the TabuList;
58         end
59     end
60 end
```

5. Results

Eight datasets were employed for testing the algorithms, all being based on known coordinates of air and health facilities. The Numba library [17] was applied to speed up the programs through the JIT compiler. All programs, including Gurobi scripts, were tested with a system containing 128 GBs of RAM with a 24 core Intel Core i9-7920X X-series processor. CUDA versions made use of a GeForce GTX 1080 Ti graphics card as well as the prior technologies mentioned.

Each algorithm ran 10 times, with all results being summarized in Figs. 3 and 4 and Tables 2 and 3. A subset of each set of missions were chosen and divided from 12 to 33. 12 vehicles were used consisting of 8 rotary-wing helicopters and 4 fixed-wing planes. The vehicle's locations were locked to the bases and at the end of a set of missions each vehicle needed to return to the same base. Time limits were randomly generated for each, needing to be completed within a 24-h period. This factor in addition, to a 10-h flight limit, prevented sets from going over 33 missions. While under the right circumstances it is possible for vehicles to complete beyond 33 missions in a day, it is highly unlikely and difficult to generate a valid test scenario. Each method had a parallel and sequential variation, and both Algorithm 2 and 3 used Algorithm 1 for building a starting solution. For accurate comparison with the optimal, Gurobi was employed and tested upon the same data.

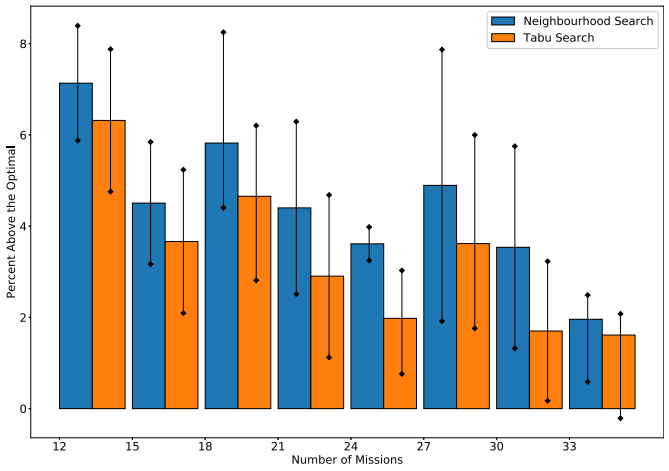

Fig. 3. Comparison of scheduling algorithms against the optimized solution.

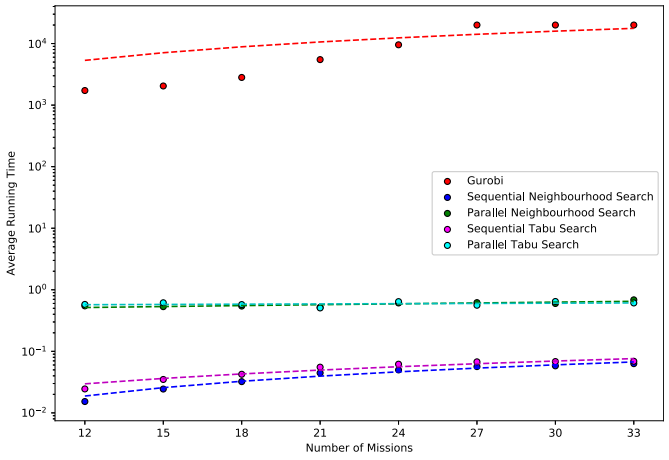

Fig. 4. Comparison of scheduling runtimes.

Based on the values shown in Fig. 3, the two optimization algorithms had similar results. All were consistently under 8% of the optimal, with some becoming quite close for the higher mission sets. The Tabu search did perform better than the Neighbourhood search in a very comparable time. The difference may not seem to be much based on the results in Table 3, though it should be noted that not a single case showed a better result for Neighbourhood search. Fig. 2 does display that it is technically faster, though at these speeds this is a negligible difference. Despite similar values, the consistency in the results show that the Tabu search is a better option. Admittedly, Fig. 2 does give the impression that upper and lower bounds are quite spread; however, the y-axis utilizes a very small incrementation. They are tight when further compared using Table 3. Additionally, total travel time is being compared instead of distance. This means the summed values are much smaller, explaining why it is easier to get within a better percentage of the optimal.

The Gurobi optimizer almost always managed to achieve an optimal solution in a much slower time compared to the custom algorithms. The problem with scheduling is that it does not scale well, with larger missions counts making the space exponentially more difficult. While it was able to solve it in an acceptable time for missions below 24, anything above this became stuck. At points beyond 27 the gap between the upper and lower bounds stopped shrinking and the optimal was no longer improving. The optimizer was set to stop running after 20,000 s which is why Fig. 4 shows the last three values as being the same. This problem shows the issue with Gurobi in a case where the optimal needs to be determined quickly, as there is not a guarantee it will find it. This is not an unusal case in a disaster where timing can mean the difference between life and death. In this scenario these custom algorithms may be more reasonable, since in the case of 33 missions the Tabu search achieved a better lower bound than the value Gurobi had solved at that point in time. While in some cases it may be more useful to utilize the optimizer for a smaller count, the difficulty of this problem and the fact that the custom algorithms have a much faster runtime, does at least offer some advantages to alternatives.

Base on the results in both Fig. 4 and Table 2, the CUDA variations does not offer a better option at this scale. Both sequential versions are significantly faster in all cases. The problem with CUDA at this level is that the kernel call itself takes a lot of time to perform. It is possible to bundle everything into a single call, though it still will not be as fast as the other algorithms with these mission sizes. The CUDA implementation is about the same for both algorithms and still significantly faster than Gurobi. It is possible that at a higher mission count the parallel implementation may begin to outperform the sequential algorithms. Although, as previously mentioned it is unlikely to have more than 33 missions in a day for this set of data, meaning the only option would be if it was for scheduling a much longer period. In any case, while not an unreasonable

Table 2
Summary of total missions times.

Number of Missions	Gurobi Runtime	Neighbourhood Search Runtime	Parallel Neighbourhood Search Runtime	Tabu Search Runtime	Parallel Tabu Search Runtime
12	1226	0.0153	0.548	0.0245	0.578
15	2045	0.0244	0.532	0.0349	0.615
18	2815	0.0322	0.544	0.0424	0.575
21	5486	0.0446	0.519	0.0553	0.507
24	9546	0.0499	0.611	0.0618	0.642
27	N/A	0.0565	0.618	0.0675	0.561
30	N/A	0.0581	0.599	0.0682	0.645
33	N/A	0.0631	0.686	0.0689	0.611

Table 3
Summary of total runtimes.

Number of Missions	Optimal Distance	Neighbourhood Search	Tabu Search
12	23.521	U: 25.495	U: 25.374
		L: 24.903	L: 24.640
		A: 25.199	A: 25.007
15	24.348	U: 25.771	U: 25.623
		L: 25.119	L: 24.858
		A: 25.445	A: 25.240
18	34.027	U: 36.834	U: 36.138
		L: 35.526	L: 34.984
		A: 36.008	A: 35.611
21	46.853	U: 49.801	U: 49.047
		L: 48.029	L: 47.380
		A: 48.915	A: 48.214
24	59.977	U: 62.365	U: 61.793
		L: 61.924	L: 60.434
		A: 62.144	A: 61.164
27	57.841	U: 62.393	U: 61.310
		L: 58.950	L: 58.859
		A: 60.672	A: 59.934
30	60.428	U: 63.904	U: 62.379
		L: 61.226	L: 60.532
		A: 62.565	A: 61.456
33	66.564	U: 68.221	U: 67.947
		L: 66.956	L: 66.425
		A: 67.868	A: 67.638

result, at this scale the sequential Tabu search is still the best option.

6. Conclusion

The results demonstrate the convenience of these custom algorithmic solutions. In all cases, Gurobi achieved an optimal in an increased time. Though not the universal, this is not acceptable in a disaster situation where time is not a luxury and fleet reorganization is required. In these scenarios, algorithms approaching the optimal become more valuable. Both algorithms were quite close to the Gurobi solution, though the Tabu slightly outperformed in all cases. In the latter simulations the Tabu even outperformed the Gurobi which could not calculate an effective permutation before the time termination expired. While parallel optimization may be an option in a larger scheduling scenario, it was not an improvement in this case. The kernel call resulted in a bottleneck which hindered any speedup gain. This should not eliminate the possibility of using this technique, though the situation should be considered. Overall, these algorithms can achieve acceptable results in an excellent time. They are adaptable to quick changes and can be run for multiple instances, displaying the effectiveness of this for the domain.

Credit author statement

Joseph Tassone: Conceptualization, Software, Methodology, Writing - Original Draft. Salimur Choudhury: Validation, Writing - Review & Editing, Funding acquisition, Supervision, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research is supported by the respective NSERC Discovery Grants from Dr.

Salimur Choudhury. Additionally, the work was funded by a Vector Institute Scholarships in AI from Joseph Tassone. Lastly, funding was also provided by a NSERC CGSM award through Joseph Tassone.

References

[1] Ornge. Online, https://www.ornge.ca/home. [Accessed 10 November 2019].
[2] Tassone J, Pond G, Choudhury S. Algorithms for optimize. fleet staging of air ambulances, Array 2020;7:100031. http://www.sciencedirect.com/science/article/pii/S2590005620300163ing.
[3] Carnes TA, Henderson SG, Shmoys DB, Ahghari M, MacDonald RD. Mathematical programming guides air-ambulance routing at ornge. Interfaces 2013;43(3):232–9.
[4] Dağlayan H, Karakaya M. An optimized ambulance dispatching solution for rescuing injures after disaster. Univ J Eng Sci 2016;4(3):50–7.
[5] Edelkamp S, Schrödl S. Chapter 14 - selective search. In: Edelkamp S, Schrödl S, editors. Heuristic search. San Francisco: Morgan Kaufmann; 2012. p. 633–69. https://doi.org/10.1016/B978-0-12-372512-7.00014-6.
[6] M. Oberscheider, P. Hirsch, Analysis of the impact of different service levels on the workload of an ambulance service provider, BMC Health Serv Res 16:487. doi: 10.1186/s12913-016-1727-5.
[7] Repoussis PP, Paraskevopoulos DC, Vazacopoulos A, Hupert N. Optimizing emergency preparedness and resource utilization in mass-casualty incidents. Eur J Oper Res 2016;255(2):531–44. https://doi.org/10.1016/j.ejor.2016.05.047.
[8] Lee D, Dinov I, Dong B, Gutman B, Yanovsky I, Toga AW. Cuda optimization strategies for compute- and memory-bound neuroimaging algorithms. Comput Methods Progr Biomed 2012;106(3):175–87. https://doi.org/10.1016/j.cmpb.2010.10.013.
[9] Cuda toolkit documentation v10.2.89. Nov 2019. https://docs.nvidia.com/cuda/.
[10] Schulz C, Hasle G, Brodtkorb AR, Hagen TR. Gpu computing in discrete optimization. part ii: survey focused on routing problems. EURO J Transp Logist 2013;2(1):159–86. https://doi.org/10.1007/s13676-013-0026-0.
[11] Hussain MM, Hattori H, Fujimoto N. A cuda implementation of the standard particle swarm optimization. In: 2016 18th international symposium on symbolic and numeric algorithms for scientific computing (SYNASC); 2016. p. 219–26. https://doi.org/10.1109/SYNASC.2016.043.
[12] Fabris F, Krohling RA. A co-evolutionary differential evolution algorithm for solving min–max optimization problems implemented on gpu using ccuda. Expert Syst Appl 2012;39(12):10324–33. https://doi.org/10.1016/j.eswa.2011.10.015.
[13] Desrochers M, Laporte G. Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. Oper Res Lett 1991;10(1):27–36. https://doi.org/10.1016/0167-6377(91)90083-2.
[14] The world's best turboprop better than ever: pilatus reveals the pc-12 ngx, Pilatus Aircraft Ltd.
[15] Aw139. https://www.leonardocompany.com/en/products/aw139.
[16] Ramos TRP, Gomes MI, Póvoa APB. Multi-depot vehicle routing problem: a comparative study of alternative formulations. Int J Logist Res Appl 2019:1–18. https://doi.org/10.1080/13675567.2019.1630374. 0 (0).
[17] A high performance python compiler. http://numba.pydata.org/; 2018.