Array 7 (2020) 100031

Contents lists available at ScienceDirect

Array

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

Algorithms for optimizing fleet staging of air ambulances

Joseph Tassone [a,*], Geoffrey Pond [b], Salimur Choudhury [a]

[a] Lakehead University, 955 Oliver Road, Thunder Bay, ON, P7B 5E1, Canada
[b] Royal Military College of Canada, 13 General Crerar Crescent, Kingston, ON, K7K 7B4, Canada

ARTICLE INFO

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

ABSTRACT

In a disaster situation, air ambulance rapid response will often be the determining factor in patient survival. Obstacles intensify this circumstance, with geographical remoteness and limitations in vehicle placement making it an arduous task. Considering these elements, the arrangement of responders is a critical decision of the utmost importance. Utilizing real mission data, this research structured an optimal coverage problem with integer linear programming. For an accurate comparison, the Gurobi optimizer was programmed with the developed model and timed for performance. A solution implementing base ranking followed by both local and Tabu search-based algorithms was created. The local search algorithm proved insufficient for maximizing coverage, while the Tabu search achieved near-optimal results. In the latter case, the total vehicle travel distance was minimized, and the runtime significantly outperformed the one generated by Gurobi. Furthermore, variations utilizing parallel CUDA processing further decreased the algorithmic runtime. These proved superior as the number of test missions increased, while also maintaining the same minimized distance.

1. Introduction

Rapid disaster response can be the difference in determining a patient's survival. In urban environments, ambulance retrieval is a standard procedure; however, the process becomes increasingly complicated with the remoteness of an incident and the dispersing of a population. As such, the placement of responders for optimal area coverage is an important and critical decision. Additionally, many air ambulance services contain a comparatively small fleet to handle vast areas [1]. Given such a small contingency, proper placement of these vehicles becomes even more crucial.

Multiple solutions have been proposed in previous works, with some implementing near-optimal metaheuristics [2] or concentrating specifically on scheduling [3]. For this research, the problem was formulated to maximize coverage, while minimizing the solution runtime. Real-mission data aided in developing a more realistic scenario rather than relying on synthetic generation. Each mission began at a potential base, performed a pickup, dropped off a patient, and then returned to the same base. The primary purpose was determining which bases the vehicles should be placed at to maximize the coverage. While exact methods are an option when time is not a factor, in emergencies there are instances where vehicles must be repositioned quickly to fill demands. In this case, algorithmic metaheuristics are much more necessary. Furthermore, many past works only considered sequential implementations, whereas the Compute Unified Device Architecture (CUDA) platform provided an opportunity for further improvement through parallelization. This research aimed to model the problem in terms of integer linear programming and then use custom algorithms to achieve a near-optimal solution.

Much of the current literature comments on the usage of exact techniques, which are less reliable when the scale increases and timing can mean the difference between life and death. These methodologies also become problematic in a shifting environment where positions must be constantly reconsidered [4].

This work aims to fill in the gap with near-optimal, fast algorithms that can be efficiently utilized over exact methods. The paper takes a generalized approach in modelling a system which can be further enhanced or customized to other environments. Additionally, there is not mentioned in the realm of parallelism, a technology that can dramatically improve the runtime of the algorithms.

The remainder of this paper is arranged as follows. Section 2 describes the related work, emphasizing previous or similar techniques for resolving the topic. Section 3 presents the problem domain and description, along with the constraints. Section 4 describes the base ranking, local search, and Tabu search-based solutions. Sections 5 and 6 explain the results and conclude the paper.

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

https://doi.org/10.1016/j.array.2020.100031
Received 9 January 2020; Received in revised form 5 May 2020; Accepted 12 May 2020
Available online 6 June 2020

2. Related work

The healthcare industry is only one candidate when considering the optimization of air assets. The present work explores the minimization of total distance for placement; however, others have examined cost in conjunction with distance. Fernandez-Cuesta et al. [5] looked at this problem from the perspective of the oil industry and suggested two heuristics for optimizing the position of a fleet of helicopters. Placement of vehicles is considered an NP-hard problem, making it unrealistic to achieve an optimal solution or use purely iterative techniques. Dong et al. [6] confronted this issue, taking a relaxed approach by solving for a subset of decision variables, and then locking the solved variables. An elementary approach was then utilized for resolving the remaining variables, minimizing the operation costs while encouraging maximized profits on fleet composition and service levels.

Regarding fleet management, an optimized solution must provide coverage with a minimized retrieval distance and potentially a minimal resource cost. Using approximate dynamic programming, Schmid [7] solved a dynamic ambulance reallocation problem. The approach resolved two of the previously mentioned criteria (coverage and cost) by relocating among a fixed set of stations. This had the added benefit of reducing the cost of subsequent ambulance requests. Maleki et al. [8] attacked a similar relocation problem; however, additionally attempted to minimize the total transit time by ambulances on succeeding calls. In Ref. [9], the authors confronted the added constraint relating to time-windows. There was a similar objective of providing minimum coverage at a reduced cost, yet the approach instead used a hybrid metaheuristic. Based on mixed-integer programming formulations, a hybrid evolutionary search algorithm (HESA) was developed. The algorithm shared similarities with genetic algorithms, while also using an embedded local search operator for improving offspring generated from the crossover operation.

Empirical data allows a model to make use of past trends for present solutions. Utilizing information like travel time, dispatch delay, and pickup time; McCormack et al. [10] developed a simulation for land-based ambulances. For actual optimization, the simulation relied on a genetic algorithm for fleet assignment. Similarly, the work of Zhen et al. [11] took a simulation approach with the use of a genetic algorithm for optimization. The work looked to maximize the expected survival probability across variable patient classes. In this approach, the authors utilized the tactic for developing a model for actual deployment and redeployment. As with previously mentioned works, Pond et al. [2] implemented a genetic algorithm, although directed the solution space towards air ambulance vehicle placement. The paper asserted that population density alone was not an accurate enough determinant for placement of vehicles, and instead relied on a large volume of past data for resolving. Other generalized solutions for these types of set-covering problems implemented fuzzy parameters, such as those in Ref. [12,13].

Local and Tabu search are well-documented methodologies for resolving optimization and set coverage problems. The literature on these algorithms is substantial and will only briefly be touched upon in this section. In local search, small localized changes are periodically made until a solution is approximately optimal. Tabu search is an extension of local search-based algorithms. It allows for the exclusion of recently explored areas within a search space and can allow moves that would not improve the objective. A list of previous states is held within a Tabu list, which prevents a search from reaching a local optimum. It only records recent moves and will not allow a solution that has been explored within a period. The list clears after a predetermined number of iterations, although the size of the list can vary depending on the problem [14]. In Ref. [15], Zimmermann exploited local search for resolving a mobile facility location problem, whereby clients were assigned to existing facilities so that the total movement and client travel costs were minimized. The model reduced the problem into smaller solvable subproblems and then implemented a modified local search for optimization. Gendreau et al. designed and modeled an ambulance location problem, resolving it with Tabu search [16]. The objective was to maximize the coverage using two ambulances, constrained by actual requirements imposed by EMS service laws. Real and randomly generated data points were used, approaching near-optimal results in a reasonable computing time compared to the CPLEX optimizer. Oberscheider and Hirsch investigated efficient transport for non-emergency patients utilizing real-patient data from the Red Cross of Lower Austria [17]. They generated all combinations of patient transports, then performed a set partitioning action upon the previous generation to gain an initial solution. They then inputted these combinations into a Tabu Search and optimized the routing.

Implementing parallelization to improve algorithms is not a new trend and has generally been used to speed up calculations using the graphics progressing unit (GPU) [18]. Reorganizing algorithms to take advantage of multiple simultaneous threads can dramatically enhance performance and see a huge improvement in the runtime of certain techniques. Hussai et al. altered the particle swarm optimization algorithm with the CUDA platform [19]. Through partially coalescing memory accesses, they were able to achieve a massive time improvement when applied to benchmarks. Fabris and Krholing utilized similar benchmarks to test the applications of the CUDA platform on a co-evolutionary differential evolution algorithm for solving minmax optimization problems [20]. Through this application, they found that the algorithm converged to a near-optimal fitness and scaled far better than nonparallel variations. Following review, there is currently little published research on applying GPU parallelization to ambulance problems. Similarly, Schulz et al. discussed in their survey that there is a comparatively small amount of literature on applying GPU parallelization to local or Tabu search [21]. Much of the current research has been directed towards swarm algorithms, though the survey suggests that it is still useful for local search-based methods.

As previously discussed in Ref. [2], multiple maximum coverage problems have relied on population density for the development of an optimized solution space. This is not feasible when considering a sizeable non-uniform density over a large region. Utilizing some traditional methodologies would mean that a significant area is ignored, risking patient survival through invalid placement. The problem can be treated as a coverage problem with the added addition of ensuring equal importance among those in the north. It should be noted that research on the effect of ambulance response time is still a wary topic [22–25]; however, from an economic standpoint, there is an interest in reducing travel distance.

Several recent algorithms have been suggested for implementation on related optimization problems, many of them based upon evolutionary or swarm-based techniques. While complex, these methodologies are potentially unreliable due to unknown optimal parameters [26]. For instance, differential evolution (DE) could be considered as it can solve in a changing environment. However, it requires certain choices to be made towards its parameters, which can significantly impact the result [27]. Swarm intelligence methods like particle swarm optimization (PSO) suffer similar issues where the choice in parameters will significantly impact the final optimization [28]. Parameter tuning is possible in cases where a fast response may not be required; however, is significantly problematic in an emergency environment. There is the possibility of a shifting ecosystem forcing the algorithms to change settings; something which may not be reasonable in a disaster situation. In the case of a critical system such as this, the algorithm utilized should be computationally efficient, while also not depending on parameter tuning to function.

Prior research has relied on evolutionary algorithms, specifically a genetic algorithm for achieving optimization [2], yet was not utilized in this paper as further constraints were implemented and tested upon. There is a significant lack of literature related to this problem, especially in recent years. This provides an opportunity to enhance with algorithmic approaches over exact techniques still being researched [29–31]. The methodology of this research was compared against optimized results, achieving near-optimal itself. Similar works have been completed regarding real provided data [3]; although this was more directly related to scheduling, they did not list substantial constraints and made use of a set-partitioning integer program. Given the organization of the data and prior works, this research will explore both local and Tabu search-based solutions, while at the same time assessing the usefulness of parallelizing both algorithms.

3. Model

Placement of ambulances for maximum coverage is a more nuanced problem that cannot rely solely on demographic data alone. Patients typically move to specialized facilities if the care required is more particular. Additionally, if a region's population is sparse then population density is a poor predictor for developing an optimized solution. As a result, historical mission data can instead be utilized for considering possible future demands. For this paper, two years of Ornge collected research data was employed, consisting of both hospital transfers and area pickups by rotary-wing aircraft.

A mission consists of a pickup, a delivery, and a return to base. To simplify the calculation, the distances between pickup and delivery points are ignored since excluding them does not affect the final coverage determination. As this is not a scheduling problem, the formulation considers that each base can only hold a single aircraft. In general, dispersing the vehicles will be more beneficial the more spread out missions become over a large area. In essence, this process would be completed before scheduling to determine the best placement of vehicles for servicing missions. Additionally, cost determination for travel is related primarily to scheduling and less to coverage. The objective function can be modified for cost if the need requires it; however, distance is more useful in determining the best regional coverage. Similarly, the vehicle speed is ignored for this problem as some areas are clustered with vehicle-specific missions (rotary-wing helicopters are required). In this case speed of the vehicle is irrelevant, as the missions cannot make use of the faster vehicle.

The aircraft fleet is made up of following two sets:

R: the set of all rotary-wing helicopters $r_i \in R \; \forall \; i = 1, \ldots, 8$.

F: the set of all fixed-wing planes $f_j \in F \; \forall \; j = 9, \ldots, 12$.

The potential bases consist of the following two sets:

A: the set of all aerodromes capable of supporting both rotary-wing helicopters and fixed-wing planes, with each being a 3-tuple of form

$a_k = < k \; \phi_k \; \psi_k >, a_k \in A \; \forall \; k = 1, \ldots, 274$

$k \equiv$ aerodrome ID

$\phi_k \equiv$ row coordinate of aerodrome k

$\psi_k \equiv$ column coordinate of aerodrome k.

H: the set of all heliports, only supporting rotary-wing helicopters, with each being a 3-tuple of form

$h_n = < n \; \phi_n \; \psi_n >, h_n \in H \; \forall \; n = 275, \ldots, 378$

$n \equiv$ heliport ID

$\phi_n \equiv$ row coordinate of heliport n location

$\psi_n \equiv$ column coordinate of heliport n location.

The set set of all missions consists of:

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

$m_z = < z \; \phi_p \; \psi_p \; \phi_d \; \psi_d \; \rho >, m_z \in M \; \forall \; z$

$z \equiv$ 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

$$\rho = \begin{cases} 1 \text{ if mission requires rotary} - \text{wing helicopter} \\ 0 \text{ otherwise} \end{cases}$$

Decision variables consist of the following:

$$v_{im} = \begin{cases} 1 \text{ if roatary} - \text{wing helicopter } i \text{ is assigned to mission } m \\ 0 \text{ otherwise} \end{cases}$$

$$w_{jm} = \begin{cases} 1 \text{ if fixed} - \text{wing plane } j \text{ is assigned to mission } m \\ 0 \text{ otherwise} \end{cases}$$

$$x_{jk} = \begin{cases} 1 \text{ if fixed} - \text{wing plane } j \text{ is assigned to aerodrome } k \\ 0 \text{ otherwise} \end{cases}$$

$$y_{ik} = \begin{cases} 1 \text{ rotary} - \text{wing helicopter } i \text{ is assigned to aerodrome } k \\ 0 \text{ otherwise} \end{cases}$$

$$z_{in} = \begin{cases} 1 \text{ if rotary} - \text{wing helicopter } i \text{ is assigned to helipad } n \\ 0 \text{ otherwise} \end{cases}$$

To perform optimal vehicle placement, distances needed to be calculated. D represented the distance between a potential aerodrome with the sum of each mission's patient pick-up and delivery location. Similarly, E described a measurement applied instead to helipads. For determining optimal distances, there are several distance formulas that may be substituted in the model. If the data was purely simulated then Euclidean distance would have been a valid option, though this is not practical when using real coordinate data based on latitude and longitude. The Earth is not a perfectly flat space and the curvature must be considered to garner an accurate measurement. In this case, Haversine distance is a far more reliable metric for the model and can be calculated with the following formula:

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

In the above formula ϕ represents row location values for ϕ_k or ϕ_n (aerodromes and helipads) respectively. Similarly, the same can be said for ψ applying to column location values ψ_k or ψ_n. The remaining distance matrices use the same formula; however, ϕ_p and ψ_p can be substituted for ϕ_d and ψ_d. The resulting matrices are both of dimension z (number of missions) by k or n (number of aerodromes or heliports) and referenced for achieving total distances for each mission. The optimization model takes the following form:

minimize

$$\sum_m \text{Unsupported} \left(\sum_i \text{Unsupported } v_{im} \left(\sum_k \text{Unsupported } D_{mk} + \sum_n \text{Unsupported } E_{mn} \right) + \sum_j \text{Unsupported } w_{jm} \left(\sum_k \text{Unsupported } D_{mk} \right) \right) \tag{2}$$

subject to

$$\sum_i \text{Unsupported } v_{im} + \sum_j \text{Unsupported } w_{jm} = 1; \forall \, m \tag{3}$$

$$\sum_k \text{Unsupported } y_{ik} + \sum_n \text{Unsupported } z_{in} = 1; \forall \, i \tag{4}$$

$$\sum_k \text{Unsupported } x_{jk} = 1; \forall \, j \tag{5}$$

$$\sum_i \text{Unsupported } z_{in} \leq 1; \forall \, n \tag{6}$$

$$\sum_j \text{Unsupported } x_{jk} + \sum_i \text{Unsupported } y_{ik} \leq 1; \forall \, k \tag{7}$$

$$\sum_m \text{Unsupported } v_{im} \leq \sum_n \text{Unsupported } z_{in} + \sum_k \text{Unsupported } y_{ik}; \forall \, i \tag{8}$$

$$\sum_m \text{Unsupported} \quad w_{jm} \leq \sum_k \text{Unsupported} \quad x_{jk}; \forall j \tag{9}$$

$$w_{jm} \leq 1 - \rho_m; \forall j \tag{10}$$

$$v_{im}, w_{jm}, x_{jk}, y_{ik}, z_{in} \in \{1, 0\}; \forall\, i, j, k, m, n \tag{11}$$

The objective function is described by Equation (2), where the total distance flown by each aircraft assigned to a mission is minimized. The binary decision variables applied are used with references to matrices D and E respectively.

These are used to sum the total distances for each assigned base. The equation is separated into three parts, with each separation indicating the possible assignments that can occur:

- Rotary-wing helicopter located at aerodrome assigned to a mission.
- Fixed-wing plane located at aerodrome assigned to a mission.
- Rotary-wing helicopter located at helipad assigned to a mission.

Furthermore, the objective function is constrained based on the rules set by Equations (3)–(11). Each mission can only support a single aircraft, constrained by Equation (3). Equations (4) and (5) limit each aircraft to a single base, while equations (6) and (7) ensure that for every helipad or aerodrome there is at most one vehicle. Enforced by prior constraints, Equation (8) guarantees that each rotary-wing assigned mission has an occupied base if the variable is set. In this particular case, both decision variables for a base assignment will not be set as a result of previous constraints. Similarly, Equation (8) does the same type of operation, except towards fixed-wing aircraft and aerodromes. As some missions are rotary-wing only, Equation (10) prevents a fixed-wing aircraft from being assigned to these missions. Lastly, Equation (11) fixes the decision variables to a binary format.

4. Algorithm

Solutions were designed with the previously discussed model while considering the prior constraints and minimization requirements. The description of this solution is described in Algorithm 1, Algorithm 2, and Algorithm 3. All versions had a sequential and parallel CUDA implementation, with minor changes in design. The primary differences related to the lack of outer loops in the CUDA versions, as these indices were acted upon simultaneously by individual threads. Additional differences are given following the description of each algorithm. Given that the differences between the parallel and sequential algorithms are only minor, the actual outlining of them are expressed using the sequential versions.

4.1. Base ranking

Algorithm 1 reduced the problem scope by ranking the most effective bases for covering every mission. The idea was to allow for a more reasonable starting position over the local search, acting on a randomly assigned set. Specifically, the algorithm looked at which bases had the lowest total Haversine distance and assigned a vehicle to each top-ranked base. The number of vehicles was predetermined based on the fleet occupied by Ornge and represented in lines 1 and 2. The input for the solution utilized coordinate data (longitude and latitude) for base locations, mission pickup, and mission destination. In lines 3–5, this information was respectively assigned to the *Destination*, *Pickup*, and *Base* matrices. In this case, the organization of the matrix indices corresponded to each mission (the exception being *Unused*). As an example: the first index of *Destination*, *Pickup*, and *Base* would be one complete mission from start to end.

Lines 8–12 of the algorithm summed the Haversine distance for each base relative to both its destination and mission pickup. Following this, lines 13–18 determined the top bases for each mission based on the minimum summed Haversine calculation. For this particular set, 12 bases were chosen as this corresponded to the number of vehicles available for assignment. Per the model's decision variables and Equations (3)–(10), a fixed-wing vehicle could not be assigned to a helipad. As such, lines 14–16 prevented the number of helipads chosen to exceed the number of rotary-wing vehicles. The ranking for each mission was then taken at line 19, where each distance was assessed based on the based on chosen bases in *Top_Vals* relative to *Distance*. Each vehicle was then assigned to a respective aerodrome or helipad at lines 20–21. Helipads were allocated first to ensure that fixed-wing aircraft had an available aerodrome available. From lines 22–29 the actual mission to base assignment took place. A constraint handled by the algorithm was that each base selected had to be able to accommodate a specific assigned vehicle. In this case 8 helicopters and 4 planes; meaning that the top bases were occasionally the second-best option instead of the first choice. Additionally, some missions specifically required the use of a rotary-wing aircraft, further constraining the rankings (lines 23–25). Line 24 and 27 finalized the algorithm, assigning the bases to each mission (placing them at the corresponding index in *Base*). This solution did not guarantee the best possible placement, only that the local search had a strong starting point. Swaps among unused bases still needed to be considered, as alternatives may have yielded better results. As such, line 30 assigned any unused bases to *Unused*.

The Base Ranking algorithm operates quite efficiently, although can be easily made parallel by the elimination of the outer loop at line 8. CUDA operates through simultaneous thread organization, so by separating i into individual threads, each can perform the inner loop j at the same time. As there are no race conditions for writing to *Distances*, each can write to a row i without issue. Another modification can be made at line 17, as it requires the summing of every row i in the *Distance* matrix to determine the total distance of each base to every mission. This would again apply a thread to each row i to determine the respective sums. These modifications remove the bottleneck associated with scaling for an increase in missions. The remainder of the algorithm did not require parallelism, as there were only small quick accesses using single loops.

Algorithm 1: Base Ranking

```
    Data: Base, mission pickup, and mission destination data
    Result: Assignment of vehicles to top aerodromes
1   heli-number← number of helicopters;
2   Plane-number ← number of airplanes;
3   Destination ← coordinates of destinations for respective missions;
4   Pickup ← coordinates of each mission;
5   Base ← coordinates of aerodromes;
6   Distances ← empty list sized to length of Pickup by length of Base;
7   Top-Vals 4← empty list sized to heli-number +plane-number;
8   for i 4r- 1 to number of bases do
9     for j 4← 1 to number of missions do
10      Add to Distances summed Haversine distance of Base i to each respective
        mission Pickup and Destination j;
11    End
12  End
13  while Top-Vals is not full do
14    if Number of helipads chosen = heli-number then
15      Choose an aerodrome instead to provide enough locations for plane-number;
16    End
17  Populate Top-Vals with indices of top aerodromes (minimum summed Haversine
    distance to each mission);
18  End
19  Rank chosen Top-Vals bases in Distances for each mission;
20  Assign rotary-wing vehicles to helipads;
21  Assign remaining rotary-wing and fixed-wing vehicles to aerodromes;
22  while all missions are not assigned a base do
23    if mission requires a fixed-wing vehicle then
24      Assign best available and compatible rotary-wing occupied base to the
        mission;
25    End
26    Else
27      Assign best available and compatible rotary-wing or fixed-wing occupied
        base to the mission;
28    End
20  End
30  Unused = remaining unassigned aerodromes and helipads:
```

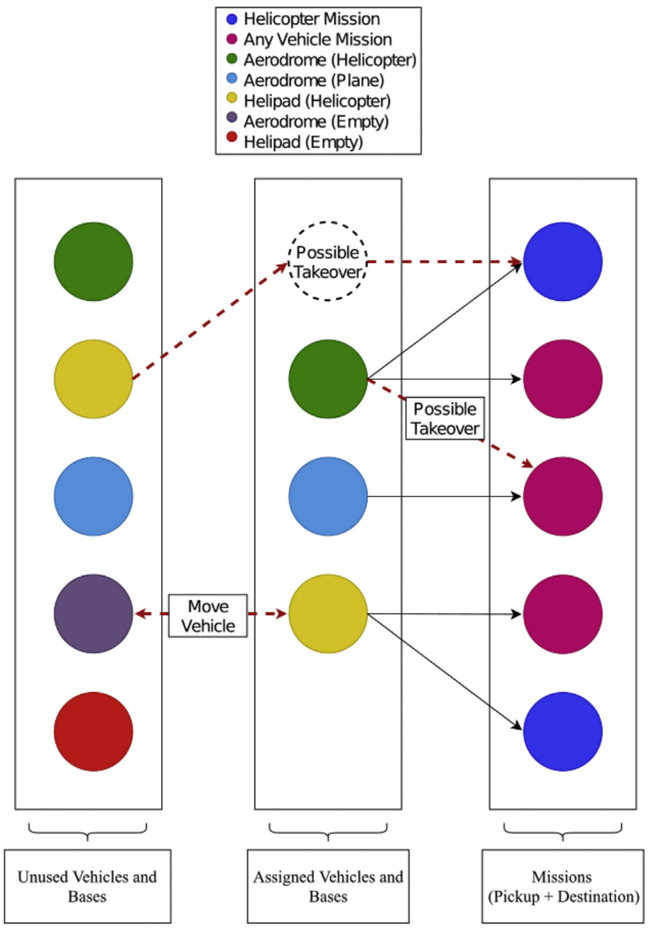

Fig. 1. Visualization of local search.

4.2. Local search fleet optimization

Algorithm 2 accepted the ranked data and unused bases assigned in Algorithm 1. As there would be no variation on successive runs (and the guarantee of optimal placement), two permutation matrices were generated corresponding to each mission index (lines 5 and 6). Essentially, this meant that swaps or take-overs would occur at different points each time, offering a distinction between results and allowing the exploration of varying neighbourhoods. Per line 9, the goal of the algorithm was to minimize the total Haversine distance across all missions. The entire set would iterate multiple times completely, stopping only after no further improvement could be found.

The algorithm ran through every mission at least once (line 11), while ensuring that each had a chance to swap with every corresponding option (line 12). Two sets of changes were possible, depending on whether a given unused base contained a vehicle. The choices from lines 13–22 were for the corresponding mission vehicle to be taken over by another vehicle assigned base or moved to a new compatible base (an empty base in *Unused*). This depended again on whether the base in *Unused* was occupied or not. In the case of the latter, all subsequent missions utilizing said vehicle needed to be updated. These changes only held if they lowered the total Haversine distance and then the previously used (or occupied) base was transferred into *Unused*. Once all missions were explored, the iterators were reset, and the algorithm repeated if there was further improvement found during the run.

The remaining changes occurred between lines 23–27 where the vehicle at $Permutation_b$ index i was compatible with the mission at $Permutation_b$ index j. The vehicles located at the respective indexed bases that would be assigned had to be of a compatible type (example: rotary-wing only as per Equation (10)) and were updated if they minimized the total Haversine distance. Additionally, if the replaced base was no longer assigned to any other mission, it was moved to *Unused*.

The parallelization of this algorithm was a bit more complex than the Base Ranking, as there were simultaneous accesses and dependencies. The outer loop at line 11 was eliminated and each index i of the $Permutation_a$ matrix was assigned a thread. This would allow the CUDA platform to do simultaneous checks for each inner loop j at once. In order to prevent race conditions a lock was placed between lines 13 and 14, lines 18 and 19, and lines 23 and 24. Once the checks were completed, a thread was allowed to perform the adjustment. Additional threads would only act once the thread released the lock. This added a level of sequential access to the algorithm, although checks were performed in parallel and the lock would only activate upon a change occurring. Since updates did not occur as often as checks, much of the bottleneck of the algorithm was eliminated.

A visualization of this algorithm can be seen in Fig. 1. Assigned vehicles were applied to each mission (summed to pickup and destination), meeting Equations (3)–(10). So long as the change was viable, another assigned base could attempt to take over the assignment of another. The new base would gain the mission, and the old base would be moved to *Unused* if it had no more assignments remaining. A similar change took place based on whether an unused base contained a vehicle. If it did then a similar takeover occurred with the now assigned based moving from *Unused*. However, if the base did not contain a vehicle, then a swap occurred with the vehicle moving to the new location and the prior assigned base moving to *Unused*. Changes only held if they reduced the total Haversine distance applied across all missions.

Algorithm 2: Local-Search Fleet Optimization

Data: Ranked and Unused Mission Data Front Algorithm 1
Result: Assignment of aircraft to missions and bases

1 *Destination* ← coordinates of destinations for respective missions;
2 *Pickup* ← coordinates of each mission;
3 *Base* ← coordinates of vehicle assigned to mission;
4 *Unused* ← coordinates of empty bases and unused vehicles;
5 $Permutation_a$ ← random permutation of indices equal to the number of missions;
6 $Permutation_b$ ← random permutation of indices equal to the number of missions;
7 i← first index of Permutation $_a$ vector;
8 j← first index of Permutation $_b$ vector;
0 k← Total Haversine Distance;
10 **while** improvement **do**
11 **while** i not equal to the last index value in $Permutation_a$ vector **do**
12 **while** j not equal to the last index value in $Permutation_b$ vector **do**
13 **if** unused vehicle i is compatible with mission j **then**
14 Replace vehicle j with unused vehicle i;
15 l← Total Haversine Distance;
16 Update if k is greater than l and move vehicle j to Unused if not allocated to a mission
17 **End**
18 **if** unused base i can host vehicle j and is compatible with mission j then
10 Move vehicle j to unused base i;
20 l ← Total Haversine Distance;
21 Update if k is greater than l and move new empty based j to Unused:
22 **End**
23 **if** vehicle i is compatible with mission j **then**
24 Replace vehicle j with vehicle i;
25 l← Total Haversine Distance;
26 Update if k is greater than l and move vehicle j to Unused if not allocated to a mission:
27 **End**
28 j = next index of $Permutation_b$ vector;
20 **end**
30 i = next index of $Permutation_a$ vector;
31 **End**
32 $Permutation_a$ — new random permutation of indices equal to the number of missions;
33 $Permutation_b$ — new random permutation of indices equal to the number of missions;
34 **End**

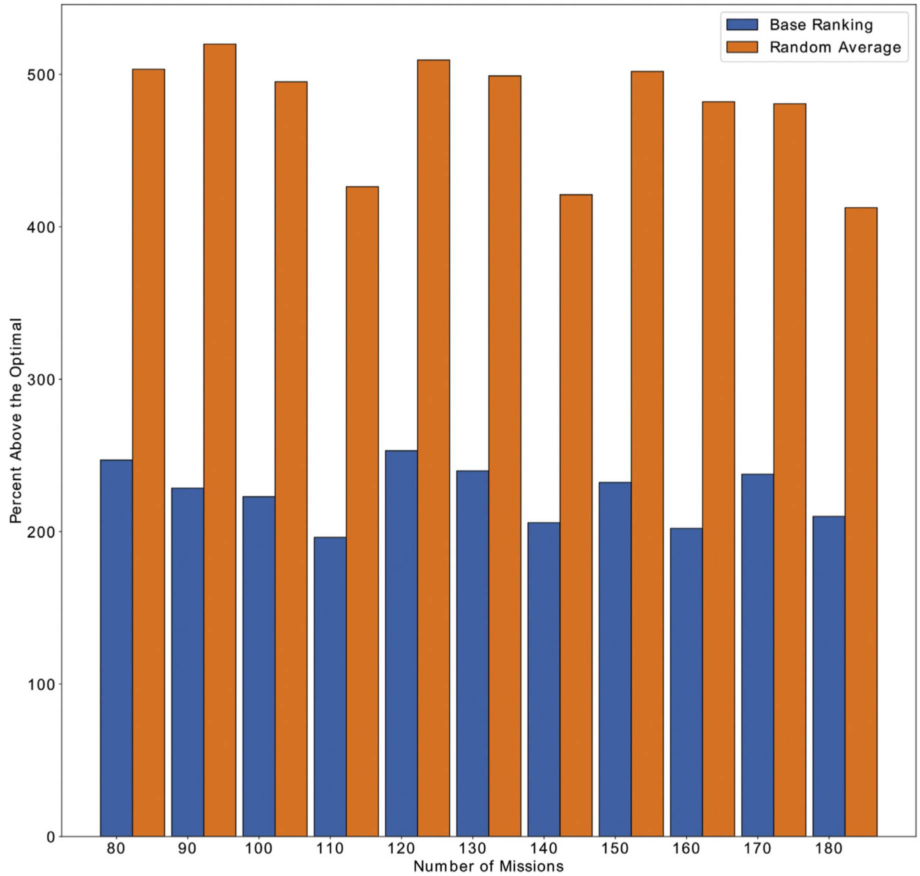

Fig. 2. Base Ranking versus random average starting over an optimal ground truth.

4.3. Tabu Search fleet optimization

The Tabu search algorithm is an extension of Algorithm 2. For the most part, the operations have remained the same and even the parallelization operates in the same fashion. It still uses Algorithm 1 as a starting point. The primary contribution of this the introduction of the *TabuList* at line 7. A Tabu search prevents recently explored neighbourhoods that improved the results from being explored again. The purpose of this is to give other possible changes a fair chance and prevent the algorithm from becoming stuck in a local optimum. Prior to each iteration j, a check is performed from lines 15–23. If the neighbourhood to be explored exists in the *TabuList*, then it is skipped depending on whether the selection is based on index i or j (lines 17–22). Selections are only added to the list if the improve the result as suggested by lines 28, 34, and 40. Each neighbourhood in the list has a counter, expressed by *TabuCounter* and decremented at either line 16 or 42. After so many iterations, the selection is removed from the *TabuList* and is allowed to be explored again.

It should be noted that a traditional Tabu search can potentially allow moves that will not improve the results. This was not done in the case of this algorithm, as it almost always resulted in a significantly poorer answer. One way to combat this problem was the introduction of more variability through the vectors $Permutation_a$ and $Permutation_b$. Originally both Algorithm 2 and Algorithm 3 did not utilize these which significantly impacted the performance. The two loops traversed sequentially through indices relative to the normal order the *Pickup* matrix. As already suggested, this caused the results to be the exact same each time as Algorithm 1 would never have a different result. Additionally, using one permutation vector did improve the result, though the introduction of two allowed for significant variability and the exploration of previously unexplored neighbourhoods by older versions.

In terms of parallel implementation, the algorithm performs almost identically to Algorithm 2. That being said, while the parallel local search results in the same answer as the sequential variation, the parallel Tabu search will not. The reasoning for this is due to the nature of the Tabu search itself. In this algorithm neighbourhoods will only be explored if they are not in the list, otherwise, they are skipped. Since multiple neighbourhoods are being explored simultaneously, the result of each can potentially be added to the list and the respective counters are reduced at different intervals. The forces it to explore differently that the non-parallel Tabu search, giving a comparable, yet different result.

Algorithm 3: Tabu Search Fleet Optimization
Data: Ranked and Unused Mission Data Front Algorithm 1
Result: Assignment of aircraft to missions and bases
1 *Destination*← coordinates of destinations for respective missions:
2 *Pickup* ← coordinates of cach mission:
3 *Base* ← coordinates of vehicle assigned to mission;
4 *Unused* ← coordinates of empty bases and unused vehicles;
5 $Permutation_a$ ← random permutation of indices equal to the number of missions:
6 $Permutation_b$ ← random permutation of indices equal to the number of missions:
7 *Tab List* ←list of recently explored neighbourhoods:
8 *Tab Counter* ← how long a neighbourhood can be held within a list;
9 i ← first index of Permutation a vector:
10 j ← first index of Permutation b vector:
11 k ← Total Haversine Distance:
12 **while** improvement **do**
13 **while** i not equal to the last index value in Permutation $_a$ vector **do**
14 **while** j not equal to the last index value tri Permutation b vector **do**
15 **if** Selection i or j for Un use. d or Vehicle arc in the *TabuList* **then**
16 Reduce the *TabuCounter* for each within the *TabuList*:
17 **if** Selection is of an index i **then**
18 Continue to next i index of $Permutation_a$ vector:
19 **end**
20 **else**
21 Continue to next j index of Permutationa vector:

(*continued on next page*)

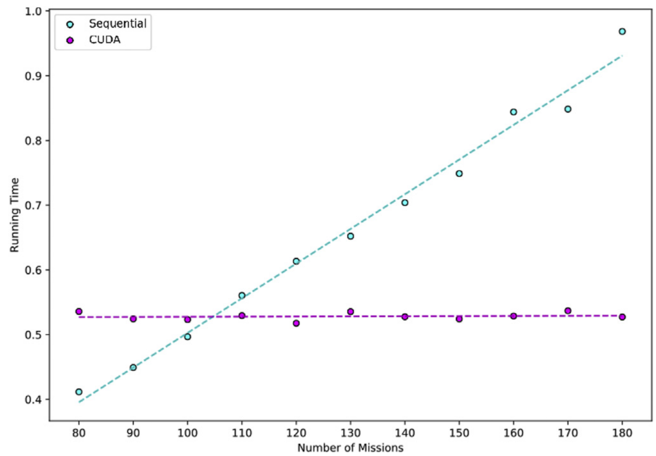

Fig. 3. Comparison of sequential and CUDA times for base ranking.

(*continued*)

end
End
if unused vehicle i is compatible with mission j **then**
Replace vehicle j with unused vehicle i;
$l \leftarrow$ Total Haversine Distance;
Update if k is greater than i and move vehicle j to Unused if not allocated to a mission:
Add selection to the *TabuList* and initiate *TabuCounter* for the selection:
end
if *unused base i can host vehicle j and is compatible with mission j* **then**
Move vehicle j to unused base i;
$l \leftarrow$ Total Haversine Distance;
Update if k is greater than l and move new empty based j to Unused;
Add selection to the TabuList and initiate TabuCounter for the selection:
end
if vehicle i is compatible with mission j then
Replace vehicle j with vehicle i;
$l \leftarrow$ Total Haversine Distance:
Update if k is greater than l and move vehicle j to Unused if not allocated to a mission:
Add selection to the *TabuList* and initiate *TabuCounter* for the selection:
end

(*continued on next column*)

(*continued*)

Reduce the *TabuCounter* for each within the *TabuList;*
j = next index of Permutationb vector:
End
i = next index of Permutation$_a$ vector;
End
Permutation$_a$ $\leftarrow$ new random permutation of indices equal to the number of missions:
Permutation$_b$ $\leftarrow$ new random permutation of indices equal to the number of missions:
End

5. Results

Eleven datasets were analyzed for testing the previously described algorithms. All ran for 10 separate attempts; the results of which are summarized in Figs. 2–5, Table 1, and Table 2. The datasets were generated as a randomized subset of missions chosen among 13,824 previously recorded in the real-mission dataset. 12 vehicles (8 rotary-wing and 4 fixed-wing) and 378 bases (274 aerodromes and 104

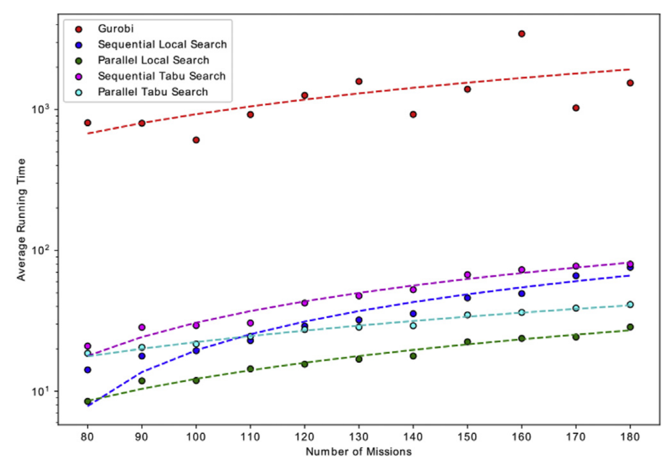

Fig. 5. Comparison of algorithmic runtimes (logarithmic scaling).

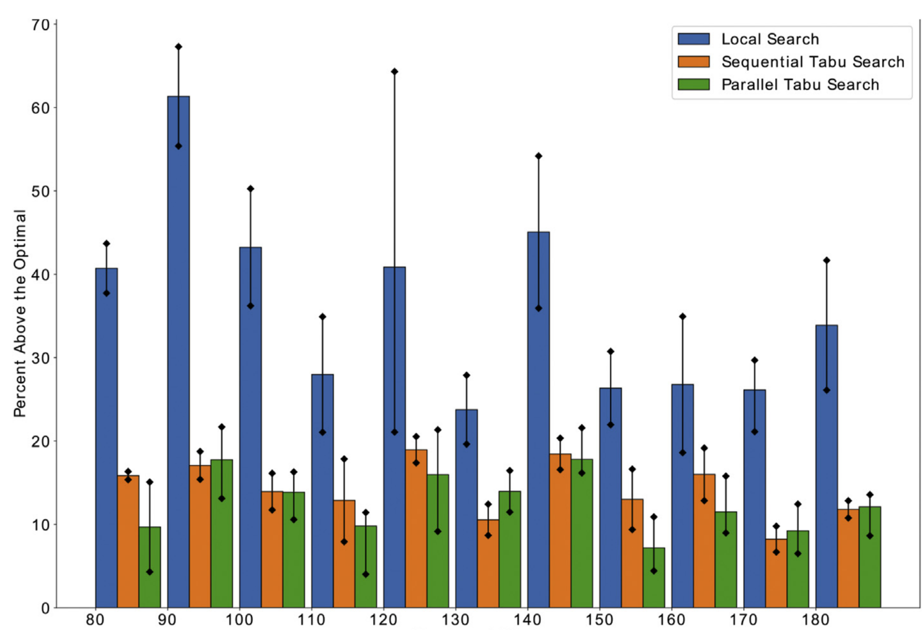

Fig. 4. Comparison of algorithms against the optimized solution.

Table 1
Summary of total haversine distance results.

Number of Missions	Optimal Distance	Local Search	Tabu Search	Parallel Tabu Search
80	19,665.9	U: 28,257.612 L: 27,087.806 A: 27,672.709	U: 22,879.656 L: 22,687.638 A: 22,783.647	U: 22,630.631 L: 20,512.258 A: 21,571.445
90	21,535.7	U: 36,028.446 L: 33,461.815 A: 34,745.130	U: 25,572.773 L: 24,853.126 A: 25,212.949	U: 26,203.804 L: 24,360.679 A: 25,372.478
100	23,578.5	U: 35,428.436 L: 32,116.723 A: 33,772.579	U: 27,382.108 L: 26,346.672 A: 26,864.390	U: 27,420.052 L: 26,073.513 A: 26,846.783
110	28,774.4	U: 38,819.082 L: 34,829.650 A: 36,824.366	U: 33,909.447 L: 31,054.683 A: 32,482.065	U: 32,061.554 L: 29,930.193 A: 31,595.873
120	27,338.4	U: 44,920.914 L: 33,098.822 A: 38,509.868	U: 32,948.576 L: 32,092.066 A: 32,520.321	U: 33,171.001 L: 29,843.666 A: 31,707.334
130	30,931.7	U: 39,554.642 L: 37,001.330 A: 38,277.991	U: 34,774.973 L: 33,617.954 A: 34,196.463	U: 36,019.660 L: 34,483.924 A: 35,251.797
140	37,530.2	U: 57,863.797 L: 51,014.197 A: 54,438.997	U: 45,159.912 L: 43,750.540 A: 44,455.226	U: 45,625.989 L: 43,599.954 A: 44,212.971
150	36,359.9	U: 47,536.625 L: 44,338.199 A: 45,937.412	U: 42,406.212 L: 39,771.094 A: 41,088.653	U: 40,321.420 L: 37,970.291 A: 38,972.549
160	40,578.4	U: 54,753.877 L: 48,131.950 A: 51,442.913	U: 48,351.222 L: 45,793.889 A: 47,072.556	U: 46,986.095 L: 44,223.194 A: 45,248.996
170	41,700.4	U: 54,078.253 L: 50,506.153 A: 52,592.203	U: 45,778.813 L: 44,871.607 A: 45,133.210	U: 46,886.254 L: 44,412.027 A: 45,549.141
180	49,758.6	U: 70,492.104 L: 62,744.421 A: 66,618.263	U: 56,146.200 L: 55,119.247 A: 55,632.724	U: 56,508.848 L: 54,055.691 A: 55,782.260

helipads) were used for an individual assignment. Datasets initially ran through an instance of base ranking to generate a strong starting point, and then adjustments were made with the local search algorithm. The latter was performed until no further improvement could be found, at which point the results were recorded. As previously mentioned, this occurred ten times for each set with the average and limits being documented upon completion. Two separate instances were executed for each: one being sequential and the other being a parallel CUDA implementation. For validation, the Gurobi optimizer was employed to measure the algorithmic against the optimized solution.

Table 2
Summary of runtime results (seconds).

Missions	Gurobi (s)	Local Search (s)	Parallel Local Search (s)	Tabu Search (s)	Parallel Tabu Search (s)
80	804.503	14.203	8.478	20.915	18.620
90	798.339	17.804	11.863	28.426	20.484
100	608.483	19.411	11.925	29.272	21.641
110	920.3110	22.879	14.424	30.491	24.502
120	1260.200	29.026	15.572	42.328	27.415
130	1583.530	32.071	16.909	47.558	28.503
140	922.661	35.524	17.800	52.625	29.169
150	1394.010	45.977	22.426	67.249	34.803
160	3454.810	49.351	23.705	72.937	36.289
170	1025.990	66.105	24.243	77.386	38.909
180	1543.920	75.794	28.569	79.921	41.246

The purpose of the base ranking algorithm was to allow an improved starting position over a randomized permuted assignment. The results of this algorithm are displayed in Figs. 2 and 3. As there was no randomization in the operation, the runtimes and values were the same for every specific sequence. Fig. 2 compared this result against the average random starting position generated by 100 sets. In all cases, the base ranking algorithm outperformed a random generation of bases applied to missions. While base ranking in conjunction with Tabu search allowed for a near-optimal result, it could not be used on its own for assignment. Despite being an improvement over randomization, Fig. 2 showed that base ranking was still substantially above the optimal ground truth. While the values were fairly uniform with an increase in mission size, they were not accepted without additional algorithms. The speed of the base ranking should be noted, as it was far faster than the local or Tabu search. Furthermore, Fig. 3 shows that the speed was nearly constant for the CUDA variation of the algorithm, while it increased linearly for the sequential version with the addition of missions. This result implies that for this scale the primary slow down point for the CUDA variation is the kernel call to the GPU itself.

Given the consistency over randomization and the speed being negligible using CUDA meant it was worth performing upon the increasing sets.

On its own, the local search algorithm proved unsuccessful in achieving a close to the optimal solution. Fig. 4 displays that even in the best scenarios, the increase was still just under 30% and went as high as over 60% for the average. The bounds were also not acceptable, with sets like 120 showing a very wide gap. These conclusions imply that it is getting stuck in a local optimum and the nature of the algorithm is preventing it from continuing. The Tabu search modification greatly improved the results and allowed it to achieve much closer to the Gurobi solutions. Per Fig. 4, all were consistently under 20% and contained much smaller bounding gaps. This suggests that each solution was achieving a similar one to each other on successive tests. Likewise, the parallel Tabu search was able to garner a similar result and even outperformed the sequential variation in some instances. There is no guarantee that the parallel version will always achieve a better metric to the sequential version, as they essentially use the same process with a difference mainly in runtime. However, they should always achieve a similar answer which is proven by Table 1. It not only shows similar averages (A), but also comparable upper (U) and lower (L) bounds. It should be noted that there is still the possibility of outliers occurring, meaning that for real-use the algorithm should be performed multiple times.

For the local search variation, the CUDA and sequential algorithms will always result in the same answer so long as matching permutations are used. As such, the only metric available for differentiating was time, which is summarized in Table 2. The CUDA running time was a significant improvement over the sequential algorithms, only increasing by a small amount given the number of missions. In Fig. 5 it can be seen that the trend line is much flatter than the sequential at a higher number of missions. This similarity is seen in the parallel Tabu search as well. Admittedly, it does display a comparable timing to the sequential variation at a lower mission count, though this changes with greater missions. It even manages to surpass the sequential local search after 110 missions. In all cases, the time for the sequential versions increased faster than the CUDA variations given an increase in the number. As previously speculated, this trend will continue to grow as the number of missions increases, giving validation to the CUDA implementation. Regardless, the timing in conjunction with near-optimal results justifies the use of parallelization for this purpose.

6. Conclusion

In a city-wide EMS system, ambulance placement is simplistic, however; in an air ambulance service, there are often far more bases than vehicles. Additionally, the coverage regions can be massive with most of these bases being heavily dispersed. Facility coordinate data suggests that demographic information is not reliable enough for the positioning of vehicles. Achieving an exact solution, while not completely unrealistic, can be time-consuming in the case of a disaster. Therefore, alternative algorithms were necessary for determining the placement of air ambulances for optimal coverage.

The results generated demonstrates the usefulness of the ranking and algorithmic solutions in both sequential and parallel forms. For the empirical data collected by Ornge, Gurobi achieved an optimized solution in a significantly increased time. Though this time may be acceptable in cases where missions are known, it may not be viable in an emergency where reorganization is required.

As such, a solution that delivers near optimization becomes all the more critical. All of the datasets reached a timeframe far exceeding traditional methodologies, while still being within an admissible target range. All algorithms approached an acceptable limit relative to the optimal, which was further enhanced, utilizing parallelization through the CUDA platform. On its own, the local search proved insufficient, although modifying the algorithm into a Tabu search greatly enhanced the result. It should be noted that this model is adaptable to possible future changes in the data and could be updated quickly. This further denotes the advantage of these techniques over other similar solutions.

Evolutionary techniques have already been tested upon in past research [2] and while swarm intelligence has its issues, it may be useful to test upon in future works. Similar research has proved promising, though these were in less critical systems and parameter tuning is still be considered an issue [32,33]. Additionally, the current iteration uses generalized constraints and it may be useful to further represent this closer to real systems [3]. This may pose a problem as most ambulance organizations keep their models confidential, though for expansion this is the next logical step. Lastly, there is significant research in static variations of these problems. While this research is quite significant, most active systems must consider dynamic changes in the actual implementation. This opens the possibility for further research into dynamic scheduling and possibly even location shifting for vehicles. Some research has suggested further building off the advantage of the parallelism seen in evolutionary algorithms, as these types of techniques function better in a dynamic shifting environment [34]. These methods can be modified to become self-adaptive, whereby a set of configurations with solution parameters are encoded into the individual solutions of the dynamic problem.

Declaration of competing interests

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.

CRediT authorship contribution statement

Joseph Tassone: Conceptualization, Software, Methodology, Writing - original draft. **Geoffrey Pond:** Conceptualization, Supervision, Resources. **Salimur Choudhury:** Validation, Writing - review & editing, Funding acquisition, Supervision.

References

[1] Ornge [Online, https://www.ornge.ca/home. [Accessed 10 November 2019].

[2] Pond GT, McQuat G. Optimizing fleet staging of air ambulances in the province of ontario, international conference on theory and practice of natural computing. 2018. p. 215–24. https://doi.org/10.1007/978-3-030-04070-3_17.

[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] Recent optimization models and trends in location, relocation, and dispatching of emergency medical vehicles. Eur J Oper Res 2019;272(1):1–23. https://doi.org/10.1016/j.ejor.2018.02.055. URL, http://www.sciencedirect.com/science/article/pii/S0377221718302054.

[5] Fernández-Cuesta E, Norddal IK, Andersson H, Fagerholt K. Base location and helicopter fleet composition in the oil industry. INFOR Inf Syst Oper Res 2017;55(2):71–92. https://doi.org/10.1080/03155986.2016.1262583. arXiv:https://doi.org/10.1080/03155986.2016.1262583.

[6] Dong Z, Chuhang Y, Lau HH. An integrated flight scheduling and fleet assignment method based on a discrete choice model. Comput Ind Eng 2016;98:195–210. https://doi.org/10. 1016/j.cie.2016.05.040.

[7] Schmid V. Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming. Eur J Oper Res 2012;219(3):611–21. https://doi.org/10.1016/j.ejor.2011.10.043. feature Clusters.

[8] Maleki M, Majlesinasab N, Sepehri MM. Two new models for redeployment of ambulances. Comput Ind Eng 2014;78:271–84. https://doi.org/10.1016/j.cie.2014.05.019.

[9] Koç Çağri, Bektaş T, Jabali O, Laporte G. The fleet size and mix location-routing problem with time windows: formulations and a heuristic algorithm. Eur J Oper Res 2016;248(1):33–51. https://doi.org/10.1016/j.ejor.2015.06.082.

[10] R. McCormack, G. Coates, A simulation model to enable the optimization of ambulance fleet allocation and base station location for increased patient survival, Eur J Oper Res 247. doi:10.1016/j.ejor.2015.05.040.

[11] Zhen L, Wang K, Hu H, Chang D. A simulation optimization framework for ambulance deployment and relocation problems. Comput Ind Eng 2014;72:12–23. https://doi.org/10.1016/j. cie.2014.03.008.

[12] Hwang M, Chiang C, Liu Y. Solving a fuzzy set-covering problem. Math Comput Model 2004;40(7):861–5. https://doi.org/10.1016/j.mcm.2004.10.015.

[13] Zimmermann K. Fuzzy set covering problem. Int J Gen Syst 1991;20(1):127–31. https://doi.org/10.1080/03081079108945020. arXiv: 10.1080/03081079108945020.

[14] 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.

[15] Halper R, Raghavan S, Sahin M. Local search heuristics for the mobile facility location problem. Comput Oper Res 2015;62:210–23. https://doi.org/10.1016/j.cor.2014.09.004.

[16] Gendreau M, Laporte G, Semet F. Solving an ambulance location model by tabu search. Locat Sci 1997;5(2):75–88. https://doi. org/10.1016/S0966-8349(97)00015-6.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Schulz C, Hasle G, Brodtkorb AR, Hagen TR. Gpu computing in discrete optimization. part ii: survey focused on routing problems. EURO Journal on Transportation and Logistics 2013;2(1):159–86. https://doi.org/10.1007/s13676-013-0026-0. 10. 1007/s13676-013-0026-0. URL.

[22] H Blackwell T, Kline J, Jeffrey Willis J, Monroe Hicks G. Lack of association between prehospital response times and patient outcomes, Prehospital emergency care. official journal of the National Association of EMS Physicians and the National Association of State EMS Directors 2009;13:444–50. https://doi.org/10.1080/10903120902935363.

[23] Cannon E, Shaw J, Fothergill R, Lindridge J. Ambulance response times and mortality in elderly fallers. Emerg Med J 2016;33. https://doi.org/10.1136/emermed-2016-206139.29. e9.1–e9.

[24] A. Bürger, J. Wnent, A. Bohn, T. Jantzen, S. Brenner, R. Lefering, S. Seewald, J.-T. Gräsner, M. Fischer, The effect of ambulance response time on survival following out-of-hospital cardiac arrest, Deutsches Aerzteblatt Online 115. doi:10.3238/arztebl.2018.0541.

[25] Perez M. Response time to the emergency department (ed) and its effect on patient flow and hospital outcomes. Chest 2015;148:481A. https://doi.org/10.1378/chest.2215810. 4, Supplement.

[26] Ab Wahab MN, Nefti-Meziani S, Atyabi A. A comprehensive review of swarm optimization algorithms. PloS One 2015;10(5):e0122827.

[27] Das S, Suganthan P. Differential evolution: a survey of the state-of-the art. IEEE Trans Evol Comput 2011;15:4–31.

[28] Mavrovouniotis M, Li C, Yang S. A survey of swarm intelligence for dynamic optimization: algorithms and applications. Swarm and Evolutionary Computation 2017;33:1–17. https://doi.org/10.1016/j.swevo.2016.12.005. URL, http://www.sciencedirect.com/science/article/pii/S2210650216302541.

[29] Almehdawe E, Jewkes B, He Q-M. Analysis and optimization of an ambulance offload delay and allocation problem. Omega 2016;65:148–58. https://doi.org/10.1016/j.omega.2016.01.006. URL, http://www.sciencedirect.com/science/article/pii/S0305048316000074.

[30] V. Bélanger, E. Lanzarone, V. Nicoletta, A. Ruiz, P. Soriano, A recursive simulation-optimization framework for the ambulance location and dispatching problem, Eur J Oper Res:https://doi.org/10.1016/j.ejor.2020.03.041. URL http://www.sciencedirect.com/science/article/pii/S0377221720302551.

[31] Abdullah L, Adawiyah C, Kamal C. A decision making method based on interval type-2 fuzzy sets: an approach for ambulance location preference. Applied Computing and Informatics 2018;14(1):65–72. https://doi.org/10.1016/j.aci.2017.04.003. URL, http://www.sciencedirect.com/science/article/pii/S2210832717300923.

[32] Khorjuvenkar PR, Singh A. A hybrid swarm intelligence approach for anti-covering location problem. Innovations in power and advanced computing technologies (i-PACT), vol. 1; 2019. p. 1–6. 2019.

[33] Lien L-C, Cheng M-Y. A hybrid swarm intelligence based particle-bee algorithm for construction site layout optimization. Expert Syst Appl 2012;39(10):9642–50. https://doi.org/10.1016/j.eswa.2012.02.134. URL, http://www.sciencedirect.com/science/article/pii/S0957417412003971.

[34] Sabar NR, Bhaskar A, Chung E, Turky A, Song A. A self-adaptive evolutionary algorithm for dynamic vehicle routing problems with traffic congestion. Swarm and Evolutionary Computation 2019;44:1018–27. https://doi.org/10.1016/j.swevo.2018.10.015. URL, http://www.sciencedirect.com/science/article/pii/S2210650218303407.