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 actually 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 which must cover all missions within a certain time frame in order 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 problem is not new, as there have been a number of exact methodologies suggested in past research. 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 an 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-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) [3]. 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 [4]. 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 [5]. 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 in order 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 [6]. 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 bounds 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 [7]. 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 [8].

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[9]. 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 [10]. 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 appleid CUDA to a co-evolutionary differential evolution algorithm for solving min-max optimization problems [11]. 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 a large number of 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 Figure 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 at all.

Figure 1: Basic scheduling setup for missions and bases.

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

\[ a_k = (k, s, \phi_k, \psi_k), a_k \in P \forall k = 1, ..., 4 \]

\( k \equiv \) 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

\[ h_k = (k, s, \phi_k, \psi_k), h_k \in H \forall k = 5, ..., 12 \]

\( k \equiv \) base ID

\( s \equiv \) vehicle speed

\( \phi_k \equiv \) row coordinate of base \( k \)

\( \psi_k \equiv \) column coordinate of base \( k \)

The set 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

\( \rho = \begin{cases} 
1 & \text{if mission requires rotary-wing helicopter} \\
0 & \text{otherwise}
\end{cases} \)
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 considered for each set in order to perform accurate scheduling. Missions have specific requirements where only a helicopter occupied base may perform certain missions.

There are a set of variables for the problem, with the first being a binary $x_{ijk}$. A respective 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 \cite{12}. For the exclusion of subtours the value can be $i = \{1, ..., n\}$ 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 hours 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 300km/h \cite{13} and fixed-wing planes are assigned a speed of 500km/h \cite{14}. 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:

$$d_{ijl} = \frac{2r \sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi - \phi_p}{2}\right) + \cos(\phi) \cos(\phi_p) \sin^2\left(\frac{\psi - \psi_p}{2}\right)}\right)}{K_s}$$  \hspace{1cm} (1)$$

In the above formula $\phi$ represents row locations, while $\psi$ 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  
\[ \sum_i \sum_j \sum_k x_{ijk} d_{ijl}; \quad l = b_k \]  
(2)  
subject to  
\[ \sum_i \sum_k x_{ijk} = 1; \quad \forall j \]  
(3)  
\[ \sum_j \sum_k x_{ijk} = 1; \quad \forall i \]  
(4)  
\[ \sum_i x_{ink} = \sum_j x_{njk}; \quad \forall k, n \]  
(5)  
\[ x_{ijk} \leq 1; \quad \forall j, k \]  
(6)  
\[ x_{ijk} \leq 1; \quad \forall i, k \]  
(7)  
\[ \sum_i \sum_j x_{ijk} d_{ijl} \leq p; \quad \forall k, l = b_k \]  
(8)  
\[ x_{ijk} d_{ijl} \leq x_{ijk} w_j; \quad \forall j, k, l = b_k, i \leq n \]  
(9)  
\[ x_{ijk} (d_{ijl} w_i) \leq x_{ijk} w_j; \quad \forall j, k, l = b_k, i > n \]  
(10)  
\[ x_{ijk} (b_k - f_i) \leq 0; \quad \forall i, j, k \]  
(11)  
\[ u_i - u_j + n(x_{ijk}) \leq n - 1; \quad 1 \leq i \neq j \leq n, \forall k \]  
(12)  
\[ x_{ijk} \in \{1, 0\}; \quad \forall i, j, k \]  
(13)  
\[ u_i \in \{1, \ldots, n\}; \quad \forall k \]  
(14)  
\[ x_{ijk} \in \{1\};\quad \forall \} \]  
(15)  

Equation (2) describes the objective function, where the total time flown by each vehicle to its assigned missions is 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) to (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 24, which corresponds to the number of hours in a day. Regardless of travel time the vehicle must wait until that missions 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 (15). 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 Algorithms

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 a 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 indicies 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 24, which is represents 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 in order to find a new valid permutation. These swap are performed by Algorithm 2 and discussed below. The algorithm returns an error if no reorganization is possible, no missions have been assigned yet, or following a reorganization the assignment fails.

Lines 43-77 perform nearly the exact same process, with the exception that there is no mission compatibility requirement. In order 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 continue 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 in order 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. In order 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.

Similar to 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 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.

### Algorithm 1: Scheduling Initialization Algorithm

**Data:** Base, mission pickup, and mission destination data  
**Result:** Assignment of missions to bases  

1. $HeliDistances \leftarrow$ distance matrix helicopter bases to each respective mission and each mission to each other mission;
2. $PlaneDistances \leftarrow$ distance matrix plane bases to each respective mission and each mission to each other mission;
3. $MissionTimes \leftarrow$ times by which a vehicle must arrive at a particular mission delivery point by;
4. $FlightLimit \leftarrow$ an integer stating the amount of allowed flight time per day;
5. $Solution \leftarrow$ a matrix initialized with each base in the first and final index for each row;

while all helicopter only missions are not assigned do

for $i \leftarrow 1$ to number of bases do

1. $MinIndex \leftarrow$ current unassigned helicopter only mission with the smallest value in MissionTimes;
2. $CurrentMin \leftarrow$ current best $i$ base for $MinIndex$;
3. if base $i$ does not contain a helicopter then
4.  continue to the next $i$ base;
5. end
6. if base $i$ does not have any missions then
7.  if $HeliDistances$ from base $i$ to $MinIndex$ > $MissionTimes$ of $MinIndex$ then
8.    continue to the next $i$ base;
9.  end
10. if $MissionTimes$ of $MinIndex$ + $HeliDistances$ from previous assigned base $i$ mission to
11. $MinIndex$ > $MissionTimes$ of $MinIndex$ then
12.    continue to the next $i$ base;
13. end
14. if $MissionTimes$ of $MinIndex$ + $HeliDistances$ from previous assigned base $i$ mission to
15. $MinIndex$ > $MissionTimes$ of $MinIndex$ then
16.    continue to the next $i$ base;
17. end
18. if assigning $MinIndex$ mission to base $i$ exceeds $FlightLimit$ then
19.  continue to the next $i$ base;
20. end
21. if $CurrentMin$ is empty or the new $i$ gives a better result then
22.  update $CurrentMin$;
23. end
24. end
25. if $CurrentMin$ is empty then
26.  perform mission swaps from Algorithm 2 and retry for new arrangement
27. end
28. else
29.  update Solution with $CurrentMin$ values;
30. end
31. end
32. end
while all remaining missions are not assigned do
    for $i \leftarrow 1$ to number of bases do
        $MinIndex \leftarrow$ current unassigned mission with the smallest value in MissionTimes;
        $CurrentMin \leftarrow$ current best $i$ base for $MinIndex$;
        $VehicleType \leftarrow$ flag determining whether to select from HeliDistances or PlaneDistances;
        if base $i$ does not have any missions then
            if $VehicleType$ type from base $i$ to $MinIndex >$ MissionTimes of $MinIndex$ then
                continue to the next $i$ base;
            end
            if MissionTimes of $MinIndex + VehicleType$ type from $MinIndex$ to base $i > 24$ then
                continue to the next $i$ base;
            end
        end
        else if base $i$ does have missions assigned to it then
            if MissionTimes of previous assigned base $i$ mission + $VehicleType$ type from previous assigned base $i$ mission to $MinIndex >$ MissionTimes of $MinIndex$ then
                continue to the next $i$ base;
            end
            if MissionTimes of $MinIndex + VehicleType$ type from $MinIndex$ to base $i > 24$ then
                continue to the next $i$ base;
            end
        end
        if assigning $MinIndex$ mission to base $i$ exceeds FlightLimit then
            continue to the next $i$ base;
        end
        if $CurrentMin$ is empty or the new $i$ gives a better result then
            update $CurrentMin$;
        end
    end
end
if $CurrentMin$ is empty then
    perform mission swaps from Algorithm 2 and retry for new arrangement;
else
    update Solution with $CurrentMin$ values;
end

An example of the process can be seen in Figure 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.

Figure 2: The movement of one mission to another base.
Algorithm 2: Neighbourhood Search Fleet Scheduling

Data: Initial Solution Data From Algorithm 1
Result: Assignment of missions to bases

1 $HeliDistances \leftarrow$ distance matrix helicopter bases to each respective mission and each mission to each other mission;
2 $PlaneDistances \leftarrow$ distance matrix plane bases to each respective mission and each mission to each other mission;
3 $MissionTimes \leftarrow$ times by which a vehicle must arrive at a particular mission delivery point by;
4 $FlightLimit \leftarrow$ an integer stating the amount of allowed flight time per day;
5 while improvement do
6   for $i \leftarrow 1$ to number of bases do
7     for $j \leftarrow 1$ to number of assigned missions do
8       $CurrentMin \leftarrow$ current best swap position;
9       for $k \leftarrow 1$ to number of bases do
10          if $i = k$ then continue to next $k$;
11          if vehicle at base $k$ is not compatible with mission $j$ then continue to next $k$;
12          $VehicleType \leftarrow$ flag determining whether to select from $HeliDistances$ or $PlaneDistances$;
13          for $l \leftarrow 1$ to number of assigned missions do
14            if assigning $j$ mission to base $k$ exceeds FlightLimit then continue to the next $k$ base;
15            if base $k$ does not have any missions then continue to the next $k$ base;
16            if $VehicleType$ type from base $k$ to mission $j > MissionTimes$ of $j$ then continue to the next $k$ base;
17            if $MissionTimes$ of $j + VehicleType$ type from $j$ to base $k > 24$ then continue to the next $k$ base;
18            else if base $k$ does have missions assigned to it then continue to the next $k$ base;
19            if $MissionTimes$ mission $l + VehicleType$ type of $l$ to $j > MissionTimes$ of $j$ then continue to the next $k$ base;
20            if $MissionTimes$ of $j + VehicleType$ type from $j$ to mission after $l > MissionTimes$ of mission after $l$ then continue to the next $k$ base;
21            if $CurrentMin$ is empty or the new assignment gives a better result then update $CurrentMin$;
22        end
23    end
24  end
25 end

4.3 Tabu Search Fleet Scheduling Optimization

This algorithm functions as a modification of Algorithm 2, with a nearly 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 a number of 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 outer 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 similar to 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) \( HeliDistances \leftarrow \) distance matrix helicopter bases to each respective mission and each mission to each other mission;
2) \( PlaneDistances \leftarrow \) distance matrix plane bases to each respective mission and each mission to each other mission;
3) \( MissionTimes \leftarrow \) times by which a vehicle must arrive at a particular mission delivery point by;
4) \( FlightLimit \leftarrow \) an integer stating the amount of allowed flight time per day;
5) \( TabuList \leftarrow \) list of recently explored neighbourhoods;
6) \( TabuCounter \leftarrow \) how long a neighbourhood can be held within a list;

while improvement do
  for \( i \leftarrow 1 \) to number of bases do
    for \( j \leftarrow 1 \) to number of assigned missions do
      \( CurrentMin \leftarrow \) current best swap position;
      for \( k \leftarrow 1 \) to number of bases do
        if \( i = k \) then continue to next \( k \);
        if vehicle at base \( k \) is not compatible with mission \( j \) then continue to next \( k \);
        \( VehicleType \leftarrow \) flag determining whether to select from \( HeliDistances \) or \( PlaneDistances \);
        for \( l \leftarrow 1 \) to number of assigned missions do
          if Selection \( k \) or \( l \) are in the TabuList then
            if Selection is of an index \( k \) then continue to next \( k \) index;
            else continue to next \( l \) index;
          end
          if assigning \( j \) mission to base \( k \) exceeds FlightLimit then continue to the next \( k \) base;
          if base \( k \) does not have any missions then continues;
          if \( VehicleType \) type from base \( k \) to mission \( j \) > MissionTimes of \( j \) then continue to the next \( k \) base;
          if MissionTimes of \( j \) + VehicleType type from \( j \) to base \( k \) > 24 then continue to the next \( k \) base;
        end
        if \( CurrentMin \) is empty or the new assignment gives a better result then update \( CurrentMin \);
      end
    end
  end
  if \( CurrentMin \) is not empty then update Solution with result in \( CurrentMin \);
  add selection to the TabuList and initiate TabuCounter for the selection;
  reduce the TabuCounter for each within the TabuList;
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 [16] was applied in order 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 Figures 3 to 4 and Tables 1 to 2. 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
Table 1: 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                        |

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 hour period. This factor in addition, to a 10 hour 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.

Based on the values shown in Figure 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. That being said, 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 2, though it should be noted that not a single case showed a better result for Neighbourhood search. Figure 1 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, Figure 1 does give the impression that upper and lower bounds are quite spread; however, the y-axis utilizes a very small incrementation. In actuality, they are fairly tight when further compared using Table 2. Additionally, total travel time is being compared instead of distance. This means the summed values are much smaller, explaining why it’s 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 seconds which is why Figure 4 shows the last three values as being the same. This problem actually 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 unusual 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.

Based on the results in both Figure 4 and Table 1, 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 of time. In any case, while not an unreasonable result, at this scale the sequential Tabu search is still the best option.
Figure 3: Comparison of scheduling algorithms against the optimized solution.

Figure 4: Comparison of scheduling runtimes.
Table 2: Summary of Total Runtimes.

| Number of Missions | Optimal Distance | Neighbourhood Search | Tabu Search |
|--------------------|------------------|----------------------|-------------|
|                    |                  | U: 25.495            | U: 25.374   |
|                    |                  | L: 24.903            | L: 24.640   |
|                    |                  | A: 25.199            | A: 25.007   |
| 12                 | 23.521           | U: 25.771            | U: 25.623   |
|                    |                  | L: 25.119            | L: 24.858   |
|                    |                  | A: 25.445            | A: 25.240   |
| 15                 | 24.348           | U: 36.834            | U: 36.138   |
|                    |                  | L: 35.526            | L: 34.984   |
|                    |                  | A: 36.008            | A: 35.611   |
| 18                 | 34.027           | U: 49.801            | U: 49.047   |
|                    |                  | L: 48.029            | L: 47.380   |
|                    |                  | A: 48.915            | A: 48.214   |
| 21                 | 46.853           | U: 62.365            | U: 61.793   |
|                    |                  | L: 61.924            | L: 60.434   |
|                    |                  | A: 62.144            | A: 61.164   |
| 24                 | 59.977           | U: 62.393            | U: 61.310   |
|                    |                  | L: 58.950            | L: 58.859   |
|                    |                  | A: 60.672            | A: 59.934   |
| 27                 | 57.841           | U: 63.904            | U: 62.379   |
|                    |                  | L: 61.226            | L: 60.532   |
|                    |                  | A: 62.565            | A: 61.456   |
| 30                 | 60.428           | U: 68.221            | U: 67.947   |
|                    |                  | L: 66.956            | L: 66.425   |
|                    |                  | A: 67.868            | A: 67.638   |
| 33                 | 66.564           | U: 73.923            | U: 76.212   |
|                    |                  | L: 69.253            | L: 72.956   |
|                    |                  | A: 68.869            | A: 70.465   |
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 all the 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 are able to 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.

References

[1] Ornge. https://www.ornge.ca/home, 2019. [Online; accessed 10-Nov-2019].
[2] Timothy A. Carnes, Shane G. Henderson, David B. Shmoys, Mahvareh Ahghari, and Russell D. MacDonald. Mathematical programming guides air-ambulance routing at ornge. Interfaces, 43(3):232–239, 2013.
[3] Hazan Da˘glayan and Murat Karakaya. An optimized ambulance dispatching solution for rescuing injuries after disaster. Universal Journal of Engineering Science, Sep 2016.
[4] Stefan Edelkamp and Stefan Schrödl. Chapter 14 - selective search. In Stefan Edelkamp and Stefan Schrödl, editors, Heuristic Search, pages 633 – 669. Morgan Kaufmann, San Francisco, 2012.
[5] Marco Oberscheider and Patrick Hirsch. Analysis of the impact of different service levels on the workload of an ambulance service provider. BMC Health Services Research, 16:487, 09 2016.
[6] Panagiotis P. Repoussis, Dimitris C. Paraskevopoulos, Alkiviadis Vazacopoulos, and Nathaniel Hupert. Optimizing emergency preparedness and resource utilization in mass-casualty incidents. European Journal of Operational Research, 255(2):531 – 544, 2016.
[7] Daren Lee, Ivo Dinov, Bin Dong, Boris Gutman, Igor Yanovsky, and Arthur W. Toga. Cuda optimization strategies for compute- and memory-bound neuroimaging algorithms. Computer Methods and Programs in Biomedicine, 106(3):175 – 187, 2012.
[8] Cuda toolkit documentation v10.2.89, Nov 2019.
[9] Christian Schulz, Geir Hasle, André R. Brodtkorb, and Trond R. Hagen. Gpu computing in discrete optimization. part ii: Survey focused on routing problems. EURO Journal on Transportation and Logistics, 2(1):159–186, May 2013.
[10] M. M. Hussain, H. Hattori, and N. Fujimoto. A cuda implementation of the standard particle swarm optimization. In 2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pages 219–226, Sep. 2016.
[11] Fábio Fabris and Renato A. Krohling. A co-evolutionary differential evolution algorithm for solving min–max optimization problems implemented on gpu using c-cuda. Expert Systems with Applications, 39(12):10324 – 10333, 2012.
[12] Martin Desrochers and Gilbert Laporte. Improvements and extensions to the miller-tucker-zeimlin subtour elimination constraints. Operations Research Letters, 10(1):27 – 36, 1991.
[13] The world’s best turboprop better than ever: Pilatus reveals the pc-12 ngx, Oct 2019.
[14] Aw139.
[15] Tânia Rodrigues Pereira Ramos, Maria Isabel Gomes, and Ana Paula Barbosa Póvoa. Multi-depot vehicle routing problem: a comparative study of alternative formulations. International Journal of Logistics Research and Applications, 0(0):1–18, 2019.
[16] A high performance python compiler, 2018.