

Simulated Annealing and Iterated Local Search Approaches to the Aircraft Refueling Problem

Karyne Alves Zampirolli(✉) and André Renato Sales Amaral

Graduate School of Computer Science (PPGI), Federal University of Espírito Santo, UFES, Vitória 29075-910, Brazil
karyne.zampirolli@edu.ufes.br, amaral@inf.ufes.br

Abstract. In this paper, we present the Aircraft Refueling Problem (ARP), whose objective is to find the shortest distance traveled by tank trucks to meet fuel demands requested by aircraft at an airport. We describe a mixed-integer linear programming (MILP) model to represent this problem. Besides MILP, Simulated Annealing (SA) and Iterated Local Search (ILS) algorithms were implemented to solve the problem. We provide computational experiments with adapted instances representing a real airport to validate the model. The results show that SA and ILS were able to schedule refueling well.

Keywords: Aircraft refueling problem · Mixed-integer linear programming · Simulated annealing · Iterated local search

1 Introduction

At an airport, after an aircraft's arrival, ground services are provided to prepare the aircraft for its next departure. Ground services include unloading/loading baggage, cleaning toilets, loading catering supplies, and refueling.

Refueling would normally begin not long after landing because it requires some time to complete. Depending on the airport, aircraft might be refueled by means of hydrant systems and/or tank trucks. Small to medium airports usually utilize tank trucks. It is widely recognized that achieving efficiency in refueling operations is necessary for aircraft to land and take off without delay. Therefore, it is important to study the aircraft refueling activity. However, few studies have been found in the literature.

The objective of this work is to study the aircraft refueling problem (ARP) using tank trucks. The Aircraft Refueling Problem (ARP) can be formulated as a Vehicle Routing Problem with Time Windows (VRPTW), which is NP-hard. In the literature, the VRPTW has been studied by many authors such as [2,3,6] and [10]. However, among the many studies published in this field, there is little research that specifically addresses the topic of aircraft refueling. As far as we know, the first paper that presented a model for the aircraft refueling process is that by Babić [1].

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12952, pp. 422–438, 2021.
https://doi.org/10.1007/978-3-030-86973-1_30

Babić [1] studied the ARP with two objectives: minimizing the number of refuelling trucks and minimizing the truck's total travelling distance. In his study, the former objective is assumed to be more important than the latter. He developed a solution approach based on branch-and-bound and solved a real case in which eight aircraft need to be refuelled.

Du et al. [4] solved the aircraft refueling problem using an Ant Colony Optimization (ACO) algorithm with MAX-MIN and Rankbased Ant System. They considered three objectives in order of importance. First: minimizing the number of vehicles; second: minimizing the sum of the service start time for each flight; and third: minimizing the total flow time of trucks. Their study also uses Non-Homogeneous vehicles.

Wang et al. [12] considered a mathematical model with time windows for the airport refueling vehicle scheduling problem (ARVSP). An evaluation function was devised to minimize the distance travelled by the vehicles and the number of vehicles utilized. A greedy algorithm was presented for the ARVSP and a practical example was solved.

Gamayanti et al. [5] examined the problem of refueling aircraft using refuelling trucks at Juanda Airport, Surabaya. The problem was formulated as the Vehicle Routing Problem with Short Time Windows, Short Travel Time and Reused Vehicles (VRPTSR). The objective was to determine the minimum number of trucks. By applying Ant Colony Optimization, they were able to reduce the number of trucks operating in DPPU Juanda Terminal 2.

Given the related works, we incorporated some different conditions to address the problem. Aircraft are located in a straight line. Refueling trucks approach the aircraft from the beginning of this line. To construct the route, it is not mandatory that aircraft i is served before aircraft $i+1$, if $a_i \leq a_{i+1}$, where a_i is the earliest moment at which refueling can start. The papers mentioned above require that refueling of aircraft i must occur before aircraft $i+1$.

We describe a mixed-integer linear programming (MILP) model and implemented Simulated Annealing (SA) and Iterated Local Search (ILS) algorithms for the ARP. The model defined for the ARP is adapted from a VRPTW model to incorporate real characteristics of the aircraft refueling process. Thus, the ARP model considers the refueling rate of an aircraft to calculate the service time. The ARP model also considers the average velocity of the refueling truck to travel from one aircraft to another. So far, no SA and ILS heuristics have been proposed to solve the ARP. We provide computational experiments with adapted instances representing a real airport to validate the model. The paper is structured as follows. We provide an overview of the problem in Sect. 2 and describe the proposed MILP model in Sect. 3. In Sect. 4, the SA and ILS algorithms are presented. Section 5 reports the performance of the proposed model and algorithms using a set of instances representing a real airport, followed by our conclusions in Sect. 6.

2 Problem Description

The ARP can be described as follows. Let N_A be a set of aircraft and K a set of non-homogeneous tank trucks such that $n = |N_A|$ and $\bar{k} = |K|$. Each aircraft i requests a quantity of fuel c_i, $i \in N_A$. Each truck k has a capacity Q_k, $k \in K$. Each truck starts at the storage area (depot) serves some aircraft and returns to the depot. It is considered that for each aircraft $i \in N_A$ the refueling activity begins during a predetermined period of time $[a_i, b_i]$. A truck cannot arrive and refuel aircraft i after moment b_i and if the truck arrives before moment a_i, it must wait to start its service at a_i.

Let us consider an airport where the refueling process assumes the following premises: at the beginning of a certain operational period, all refuelers are in the storage area and each used truck k is filled with its full capacity Q_k; each aircraft $i \in N_A$ is served using a single truck; to construct the route, it is not mandatory that aircraft i is served before aircraft $i+1$, if $a_i \leq a_{i+1}$; the number $\bar{k}$ of available trucks is sufficient for refueling; aircraft i with $c_i > Q_k$ are not considered here; the reason for this is that most of the aircraft with high fuel demands are serviced by refueling using underground piping systems, known as hydrant refueling; aircraft are located in a straight line; refueling trucks approach the aircraft from the beginning of this line; the fleet used is heterogeneous, that is, the capacity of each truck is different. The ability of scheduling trucks with different capacities makes the model more applicable under real circumstances. Thus, the aim of ARP is to find the shortest distance traveled by tank trucks to meet fuel demands requested by aircraft at an airport.

3 Mathematical Model

We describe a MILP model for the ARP. The problem can be represented by a complete graph $G(N, A)$ where $N = N_A \cup \{0, n+1\}$ and $N_A = \{1, 2, ..., n\}$. The graph consists of $|N_A| + 2$ vertices, where the storage area (depot) is represented by vertexes 0 and $n + 1$.

The set of arcs denoted by $A = \{(i, j) : i, j \in N, i \neq j, i \neq n + 1, j \neq 0\}$ represents connections between the depot and the aircraft and among the aircraft. No arc terminates in vertex 0, and no arc originates from vertex $n + 1$. Table 1 shows the list of parameters for each node i, where the column *Par* designates the name of the parameter. The depot is associated with a time window: trucks may not leave the depot before a_0 and must be back before or at time b_{n+1}. Table 2 shows the list of parameters defined for each truck $k \in K$. It is assumed that Q_k, a_i, b_i, c_i, d_i and $D_{i,j}$ are non-negative integers, while the displacement time of each truck is assumed to be a positive integer. The decision variables used in the proposed model are described in Table 3.

We want to find minimal cost routes, one for each truck, where each aircraft will be visited only once. Every route originates at vertex 0 and ends at vertex $n + 1$, and the time windows and capacity constraints must be observed. Based on the previous definitions, the proposed mathematical model for the problem is defined as follows:

Table 1. List of parameters for each node.

Par	Description	*Par*	Description
$D_{i,j}$	Distance between nodes $i, j \in N$	c_i	Fuel quantity requested by aircraft i
a_i	Earliest moment at which refueling can start	b_i	Latest possible starting time for refueling
d_i	Distance between parked aircraft i and the storage area	q_i	Refueling rate for aircraft i

Table 2. List of parameters for each truck $k \in K$.

Par	Description
V_k	Average velocity of truck k
Q_k	Maximum fuel capacity of truck k

$$min : \sum_{k \in K} \sum_{i \in N} \sum_{j \in N, j \neq i} D_{i,j} X_{i,j,k} \tag{1}$$

subject to:

$$\sum_{k \in K} \sum_{j \in N, j \neq i} X_{i,j,k} = 1 \quad i \in N_A \tag{2}$$

$$\sum_{i \in N_A} \sum_{j \in N, j \neq i} c_i X_{i,j,k} \leq Q_k Z_k \quad k \in K \tag{3}$$

$$\sum_{j \in N} X_{0,j,k} = Z_k \quad k \in K \tag{4}$$

$$\sum_{i \in N, h \neq i} X_{i,h,k} - \sum_{j \in N, j \neq h} X_{h,j,k} = 0 \quad h \in N_A, k \in K \tag{5}$$

$$\sum_{i \in N} X_{i,n+1,k} = Z_k \quad k \in K \tag{6}$$

Table 3. List from decision variables.

Decision variable	Description
$X_{i,j,k}$	Binary variable equal to 1 if truck $k \in K$ moves from node $i \in N$ to node $j \in N$, $i \neq j$. Otherwise, it is equal to 0
$T_{i,k}$	Indicates the time when truck $k \in K$ starts serving node $i \in N$
Z_k	Binary variable equal to 1 if truck $k \in K$ is used to meet any request. Otherwise, it is equal to 0
$TotalDis_k$	Indicates total distance traveled by each truck $k \in K$

$$\sum_{i \in N} \sum_{j \in N, j \neq i} D_{i,j} X_{i,j,k} = TotalDis_k \quad k \in K \tag{7}$$

$$T_{j,k} \geq T_{i,k} + \frac{c_i}{q_i} + \frac{d_{i,j}}{V_k} - M(1 - X_{i,j,k}) \quad i, j \in N, k \in K \tag{8}$$

$$a_i \leq T_{i,k} \leq b_i \quad k \in K \tag{9}$$

$$X_{i,j,k} \in \{0, 1\} \quad i, j \in N, k \in K \tag{10}$$

$$Z_k \in \{0, 1\} \quad k \in K \tag{11}$$

The objective function (1) is to minimize the distance traveled by the trucks to perform the refueling process. The $D_{i,j}$ parameter is calculated as the absolute difference between d_i and d_j, i.e. $|d_i - d_j|$. Parameter d_i is the distance between parked aircraft i and the depot. It is only possible to calculate $D_{i,j}$ in this way because the aircraft are parked in a straight line. The deposit is located in the same direction as the aircraft. Constraint (2) states that each aircraft will be visited only once.

Constraint (3) ensures that no truck is loaded with more than its capacity allows it to. The next three Eqs. (4), (5), and (6) ensure that each truck used leaves depot 0. After arriving at an aircraft, the vehicle leaves again and finally arrives at the depot $n + 1$. Constraint (7) counts the total distance traveled by each truck.

Inequality (8) states that a truck k cannot arrive at aircraft j before $T_{i,k} + c_i/q_i + d_{i,j}/V_k$ if it is traveling from i to j, where the service time to refuel the aircraft i is represented by the amount of fuel requested divided by the refueling rate (c_i/q_i); and the time to travel from i to j is represented by the distance between nodes i, j divided by the average velocity of the refueling truck ($d_{i,j}/V_k$). Refueling rate and average velocity are also considered in [1]. Here M is a large scalar. The slack in Constraint (8) corresponds to the waiting time of the vehicle serving aircraft j, that is, the time spent between the moment the vehicle arrives at the aircraft destination and the start of service. Finally, Constraint (9) ensures that refueling time windows are observed and (10)–(11) are the integrality constraints. Note that an unused truck k ($Z_k = 0$) is modeled by driving the empty route $(0, n + 1)$.

4 Simulated Annealing and Iterated Local Search Approaches to the Problem

In this section, we present how a solution to the problem is represented, how the initial solutions are generated, the types of moves used to explore the solution space, the evaluation function used to guide the search and the detailed description of the proposed algorithms.

4.1 Solution Representation

A solution to the problem can be represented by multiple routes, each followed by a copy of the depot. Each copy of the depot is a separator between routes. For example, a solution representation as shown in Table 4 would correspond to a problem solution made of three routes. The first route contains aircraft (customers) 12, 4, 1, 6 and 10, the second route contains aircraft 2, 11, 9 and 3, the third route contains aircraft 8, 5 and 7. Each route starts at the depot, visiting aircraft and ends at the depot. Other authors have also used this representation (e.g. [8]).

Table 4. An example of the solution representation.

12	4	1	6	10	0	2	11	9	3	0	8	5	7

4.2 Initial Solution

To determine a feasible initial solution for the application of the algorithms, two methods were defined in order to analyze the behavior of the algorithm when subjected to different initial solutions. The first approach considered a Random Constructive Heuristic, and the second uses the Nearest-Neighbor Heuristic proposed by Salomon [11], as described below.

Random Constructive Heuristic (RCH). The initial solution to the problem was generated through a procedure that, starting from the depot, inserts a single aircraft on a route at each iteration. This insertion is done by analyzing the time window of each aircraft and the capacity limit of the refueling truck. The choice of the aircraft to be inserted in the route is made randomly among those that were not inserted and whose insertion does not exceed neither the truck's capacity nor the pre-defined time window of each aircraft. If there are no aircraft satisfying these two conditions, the depot is returned, and a new route is started. The procedure comes to an end when all aircraft are inserted in some route. It can be observed that using this construction the initial solutions are always feasible and we obtain an upper limit on the number of trucks to be used.

Nearest-Neighbor Heuristic (NNH). According to Salomon [11], the nearest-neighbor heuristic starts every route by finding the unrouted customer "closest" to the depot. At every subsequent iteration, the heuristic searches for the customer "closest" to the last customer added to the route. This search is performed among all the customers who can feasibly (with respect to time windows, vehicle arrival time at the depot, and capacity constraints) be added to the end of the emerging route. A new route is started any time the search fails unless there are no more customers to schedule. The metric used in this approach tries to account for both geographical and temporal closeness of customers.

Let the last customer on the current partial route be customer i, and let j denote any unrouted customer that could be visited next. It searches for node j that minimizes a weighted cost $x = a * tempDist + b * waiting + c * urgency$, where a, b and c are parameters. The wait related to node j can be characterized by: $wait = max(a_i - (hCurrent + currentServiceTime + tempDist_{current,i}), 0)$, where a_i is the starting time of the time window of node i, $hCurrent$ is the current time (which corresponds to the start of the current node's service), $currentServiceTime$ is the time consumed at the current node to perform its service and $tempDist_{current,i}$ is the time from the current node to node i. Urgency is defined as: $urgency = (b_i - (hCurrent + currentServiceTime + tempDist_{current,i})$. The procedure comes to an end when all aircraft are inserted. It is noted that with this construction the initial solutions are always feasible.

4.3 Neighborhood Structure

A set N of seven different move types is used to make changes between customers (aircraft) of different routes and customers of the same route. *Shift* and *Swap* moves define the inter-route neighborhood structures, and *Reinsertion* and *Swap* moves define the intra-route neighborhood structures. When the algorithm is looking for the next solution, there is a certain probability of choosing one of the seven moves to improve the current solution. It can be observed that by using these moves, both feasible and non-feasible solutions are considered in the exploration of the search space.

Inter-Route Neighborhood Structures. The two structures are based on λ-*interchange* moves (Osman [9]), which consists of swapping up to λ consecutive customers between two routes. To minimize the computational effort, λ was limited to 2. These swaps are best explained as doubles (λ_1, λ_2), in which λ_1 clients are transferred from route r_1 to route r_2 and the λ_2 clients of route r_2 to route r_1. Therefore, disregarding symmetries, the following combination of *2-interchange* moves is possible: $((1;0), (1;1), (2;0), (2;1), (2;2))$. Suppose $r_1 = 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0$ and $r_2 = 0 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 0$.

Shift(1,0) - $N^{(1)}$: A client i is transferred from route r_1 to route r_2. For example, client 3 is removed from route r_1 and inserted into route r_2. The new route is the sequence: $r_1 = 0 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 0$ and $r_2 = 0 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 3 \rightarrow 9 \rightarrow 10 \rightarrow 0$.

Swap(1,1) - $N^{(2)}$: Performs a permutation between a client i of a route r_1 and a client j belonging to another route r_2. For example, client 2 is exchanged with client 7 and the new route is the sequence: $r_1 = 0 \rightarrow 1 \rightarrow 7 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0$ and $r_2 = 0 \rightarrow 6 \rightarrow 2 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 0$.

Shift(2,0) - $N^{(3)}$: Two consecutive clients i and j are transferred from route r_1 to another r_2. Adjacent clients 2 and 3 are moved from one route to another, forming the route $r_1 = 0 \rightarrow 1 \rightarrow 4 \rightarrow 5 \rightarrow 0$ and $r_2 = 0 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 2 \rightarrow 3 \rightarrow 10 \rightarrow 0$.

Swap(2,1) - $N^{(4)}$: Two consecutive clients, i and j, belonging to a route r_1 are exchanged with a client $i^{'}$ from another route r_2. Adjacent clients 1 and 2 of a route are exchanged with client 7 from another route. The new route is the sequence: $r_1 = 0 \rightarrow 7 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0$ and $r_2 = 0 \rightarrow 6 \rightarrow 1 \rightarrow 2 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 0$.

Swap(2,2) - $N^{(5)}$: Two consecutive clients, i and j, belonging to route r_1 are exchanged with two other consecutive clients $i^{'}$ and $j^{'}$ from another route r_2. Adjacent clients 1 and 2 of one route are exchanged with adjacent clients 6 and 7 from another route, forming route $r_1 = 0 \rightarrow 6 \rightarrow 7 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0$ and $r_2 = 0 \rightarrow 1 \rightarrow 2 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 0$.

Note that such combinations include shift $((1;0);\ (2;0))$ and swap $((1;1);\ (2;1);\ (2;2))$ moves.

Intra-Route Neighborhood Structures. The *Reinsertion* and *Swap* moves are defined as follows.

Reinsertion - $N^{(6)}$: A customer is removed and reinserted at another position on the route. Client 3 of the original route r_1 is relocated to another position on the route and the new route is sequence: $r_1 = 0 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 0$

Swap - $N^{(7)}$: It consists of swapping two customers on the same route. On route r_1, clients 2 and 5 are exchanged, forming the route $r_1 = 0 \rightarrow 1 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 0$.

4.4 Evaluation Function

Solution s is evaluated by function f, given by equation (12), which must be minimized:

$$f(s) = Dist(s) + \alpha * f_1(s) + \beta * f_2(s) \tag{12}$$

where: $Dist(s)$ is the distance traveled by the trucks in solution s; $f_1(s)$ is the total load excess among all used vehicles; $f_2(s)$ represents the total amount of time by which the latest times to service aircraft are exceeded, and α, β are penalty weight factors. Note that when s is feasible, the value given by f will only correspond to the traveling distance, since in this case: $f_1(s) = f_2(s) = 0$. The comparison between two solutions is made based only on the value of the evaluation function, without taking into account the number of vehicles associated with each solution.

In the developed method, α equal to the value of the greatest distance between any two clients and β equal to 1000 are adopted. These values are set so that infeasible solutions have a low probability of being chosen.

4.5 Simulated Annealing

The SA algorithm is a metaheuristic that has been successfully used in several works for obtaining a good approximation of a function's optimal solution. SA is based on an analogy of an optimization problem with thermodynamics [7]. SA is presented in Algorithm 1. It has four main parameters: initial temperature (T_0), final or freezing temperature (T_c), cooling factor (α) and the maximum number of iterations at each temperature (SA_{max}).

```
   input : s_0, T_0, T_c, α, SA_max
   output: Best solution s*
 1 numIter ← 0;
 2 s* ← s_0;
 3 s ← s_0;
 4 T ← T_0;
 5 while T > T_c do
 6 |   while numIter < SA_max do
 7 |   |   numIter ← numIter + 1;
 8 |   |   s' ← GenerateNeighbor(s) ;
 9 |   |   Δf ← f(s') − f(s*) ;
10 |   |   if Δf < 0 then
11 |   |   |   s ← s';
12 |   |   |   if f(s') < f(s*) then
13 |   |   |   |   s* ← s';
14 |   |   |   end
15 |   |   end
16 |   |   else
17 |   |   |   Generate random number x ∈ [0, 1];
18 |   |   |   if x < e^{−Δ/T} then
19 |   |   |   |   s ← s';
20 |   |   |   end
21 |   |   end
22 |   end
23 |   T ← T * α;
24 |   numIter ← 0;
25 end
```

Algorithm 1: SA algorithm pseudocode

Considering a problem of minimizing function f and an initial solution s, at each iteration, the temperature T is determined using a cooling factor $0 < \alpha < 1$. Then a new solution s' is found by performing local searches in the neighborhood of the current solution. If the delta energy level, $\Delta = f(s') - f(s)$, is negative, then the new solution is of better quality than the current solution; and s' solution is updated as the current solution. Otherwise, when the solution is of poorer quality, it can be accepted with $e^{-\Delta/T}$ probability. At each temperature SA_{max} iterations are performed where at each iteration, a new solution is found. This procedure is performed until the temperature reaches a final or freezing temperature T_c and the closer to that temperature, the less likely the algorithm will accept bad moves [7].

4.6 Iterated Local Search

The Iterated Local Search metaheuristic is simple but very effective. It starts with the initial solution s_0 and applies a local search to it, obtaining s^*. Then, the algorithm performs the following steps: it perturbs the current best solution s^*; obtains a solution s'; and performs a local search in s', reaching a local optimum s''. If the solution s'' is better than the current best solution s^* then the algorithm transforms s'' into the new current solution. Otherwise, the algorithm performs another iteration. As long as the stopping criterion is not met, the procedure is repeated. Its pseudocode is given by Algorithm 2.

```
   input : Initial solution s_0
   output: Best solution s*
1  numIter ← 0;
2  s* ← LocalSearch(s_0);
3  while numIter < ILS_max do
4  |  s' ← Pertubation(s*) ;
5  |  s'' ← LocalSearch(s') ;
6  |  numIter ← numIter + 1;
7  |  if f(s'') < f(s*) then
8  |  |  s* ← s'';
9  |  |  numIter ← 0;
10 |  end
11 end
```

Algorithm 2: ILS algorithm pseudocode

Line 4 of Algorithm 2 calls a perturbation routine that performs different moves from those used in the neighborhood generation. Similar to a 2-opt move, a customer is randomly chosen and permuted with the next customer within the route, creating a small but distinct perturbation. The local search method is as follows.

Local Search. The objective of Local Search is to find local optima, generating at each iteration, neighboring solutions from the current solution and accepting only better solutions than the current one or those that meet a certain acceptance criterion. At the end of the procedure, the best solution found is returned. In this study, the Hill Climbing local search algorithm was implemented. The algorithm ends after H iterations without improving the objective function, where H is an input parameter. Algorithm 3 shows the pseudo-code as implemented in this work.

```
    input  : Solution s, H
    output: Best solution s*
 1  i ← 0;
 2  s* ← s;
 3  while i < H do
 4  |   s' ← GenerateNeighbor(s*) ;
 5  |   Δf ← f(s') − f(s*) ;
 6  |   if Δf < 0 then
 7  |   |   s* ← s';
 8  |   |   i ← 0;
 9  |   end
10  |   i ← i + 1;
11  end
```

Algorithm 3: Hill Climbing local search algorithm

5 Computational Experiments

For the experiments, a set of eight instances with six to forty aircraft and six to ten available trucks were used to represent a real airport. Let Instance W_i be associated with a pair (Number of aircraft, Number of available trucks). Then, $W_1 = (6, 6)$; $W_2 = (11, 6)$; $W_3 = (15, 6)$; $W_4 = (18, 6)$; $W_5 = (18, 6)$; $W_6 = (25, 7)$; $W_7 = (30, 8)$; $W_8 = (40, 10)$. All instances are based on the scenario of the aircraft refueling activity at medium traffic volume airports. The data used correspond to the premises observed at a real airport.

The MILP model was solved using IBM ILOG CPLEX 12.8, written at language OPL, and the proposed SA and ILS algorithms were coded in Python 3.8. The computational tests were performed on a desktop computer with Intel Core i5-8250U Processor of 1.60 GHz with 4 GB RAM, running Windows 10 64-bit. The algorithms were run 5 times, and in each run, 300 repetitions of the algorithm were performed in order to find the incumbent solution (best solution during the re-optimization process).

5.1 Definition of Parameters

To solve the MILP model, CPLEX was parameterized in the following way: maximum execution time of 2 h (7,200 s); tree memory limit of 80 GB; work memory of 10 GB and number of threads equal to 8. The reason for those parameters is to handle the out of memory situation normally found in larger models.

The implementation of SA was based on Algorithm 1. The parameters of SA are: initial temperature (T_0); cooling factor (α); the maximum number of iterations at each temperature (SA_{max}) and final or freezing temperature (T_c). The parameter SA_{max} was adopted as being equal to the number of aircraft in each instance, that is, equal to n. For the stopping criterion, a freezing temperature was considered, which by the definition of Annealing, occurs when the possibility of improving the current result is very low. In this work, $T_c = 0.001$ was adopted. Parameters T_0 and α were determined by experimentation. Five values

for parameter T_0 were analyzed: $T_0 = 10; 100; 1,000; 10,000; 100,000;$ and for each T_0 different cooling factors were considered, $\alpha = 0.80; 0.85; 0.9; 0.95; 0.98$.

Figure 1 shows the results obtained considering an instance of the problem for the different initial temperatures and cooling factors for the SA algorithm. The horizontal axis represents the different values of T_0 considered and the vertical axis is the solution value to the problem, that is, the route distance ($f(s)$). Note that α is represented by the bars following the colors specified in Fig. 1.

The instances used for parameter calibration (including that in Fig. 1) had $n = 12$ to 18 and $\bar{k} = 6$. Tests with these instances showed that the best combination of parameter values is $T_0 = 10$ and $\alpha = 0.98$.

The ILS_{max} parameter of Algorithm 2 was also defined as the total amount of aircraft ($n = |N_A|$) in each instance.

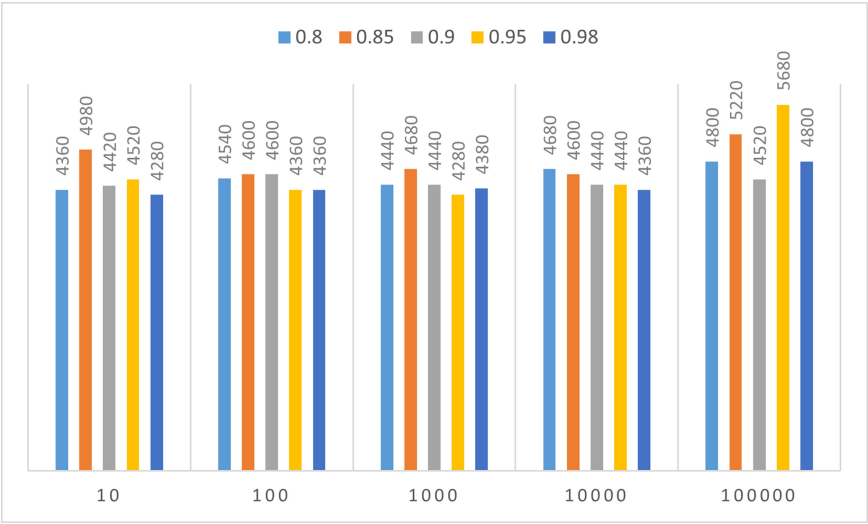


Fig. 1. Parameter analysis T_0 and α

5.2 Study of the Impact of the Initial Solution

Table 5 presents a comparison of the algorithms SA and ILS with initial solutions provided by the different construction algorithms RCH and NNH. For this comparison, five instances were considered (A, B, C, D, E) having 5 to 25 aircraft and 6 to 7 available trucks. For each instance, the algorithms were run five times. The table shows the best solution obtained among the five runs with 300 repetitions of the algorithm in each run. Columns Instance, n, $\bar{k}$, $f(s)$ and $Time$ represent the instance, the number of aircraft, the number of the available trucks, the best solution and the total execution time (in seconds) of the SA and ILS algorithm, respectively.

It can be observed that for Instances B to E, the best solution found by SA and ILS was obtained by using the initial solution given by the NNH heuristic (highlighted in bold). For Instance A, the best solution found by SA was given by the NNH heuristic and ILS found the same solution with both initial solution approaches.

Related to the execution time, observe that for Instances A to D, the solutions were found with a time under one second (< 1). For the largest instance (Instance E), the RCH heuristic takes longer to build a solution due to the fact that the nodes to be visited are chosen randomly. This random choice may lead to many poor solutions. Note that RCH - ILS spent 158s to obtain the best solution. The larger the instance, the longer the RCH heuristic will take to build a solution. Therefore, for this work we used the NNH heuristic to construct the initial solutions.

Table 5. Comparison of the algorithms with initial solutions provided by different construction algorithms.

Instance	n	$\bar{k}$	RCH - SA		RCH - ILS		NNH - SA		NNH - ILS	
			$f(s)$	Time	$f(s)$	Time	$f(s)$	Time	$f(s)$	Time
A	5	6	1,780.00	<1	**1,620.00**	<1	**1,620.00**	<1	**1,620.00**	<1
B	9	6	6,357.00	<1	5,240.00	<1	5,394.00	<1	**3,320.00**	<1
C	15	6	3,862.00	<1	3,520.00	<1	**3,000.00**	<1	3,080.00	<1
D	20	6	6,120.00	<1	5,800.00	<1	**4,100.00**	<1	4,800.00	<1
E	25	7	8,680.00	87.8	8,480.00	158.07	7,194.00	1.67	**6,820.00**	4.22

5.3 Results and Analysis

Six variants of the SA and ILS algorithms were implemented. Table 6 contains the costs of the solutions obtained and the description of the results in each column, where:

I Corresponds to the execution of the SA algorithm using the local search with a random neighborhood considering all seven proposed moves ($N^{(1)}$ to $N^{(7)}$).
II Corresponds to the execution of the ILS algorithm using the local search with a random neighborhood considering all seven proposed moves ($N^{(1)}$ to $N^{(7)}$).
III Corresponds to the execution of the SA algorithm using the local search with a random neighborhood considering only four proposed moves. We analyzed neighborhoods $N^{(1)}$, $N^{(2)}$, $N^{(4)}$ and $N^{(5)}$.
IV Corresponds to the execution of the ILS algorithm using the local search with a random neighborhood considering only four proposed moves. We analyzed neighborhoods $N^{(1)}$, $N^{(2)}$, $N^{(4)}$ and $N^{(5)}$.
V Corresponds to the execution of the SA algorithm using the local search with a random neighborhood considering four moves, which were neighborhoods $N^{(2)}$, $N^{(3)}$, $N^{(6)}$ and $N^{(7)}$.

VI Corresponds to the execution of the ILS algorithm using the local search with a random neighborhood considering four moves, which were neighborhoods $N^{(2)}$, $N^{(3)}$, $N^{(6)}$ and $N^{(7)}$.

Table 6. Result of the execution of the six variants of the SA and ILS algorithms.

W	n	CPLEX			Best total cost obtained					
		LB	UB	$GAP(\%)$	I	II	III	IV	V	VI
1	6	370.00	**370.00**	0.00	**370.00**	**370.00**	**370.00**	**370.00**	**370.00**	**370.00**
2	11	1,400.00	**1,400.00**	0.00	1,760.00	**1,400.00**	1,880.00	**1,400.00**	1,880.00	**1,400.00**
3	15	828.68	**860.00**	3.64	**860.00**	**860.00**	**860.00**	**860.00**	**860.00**	**860.00**
4	18	1,588.00	**1,800.00**	11.78	2,920.00	**1,800.00**	2,660.00	**1,800.00**	2,660.00	**1,800.00**
5	18	1,588.21	1,690.00	6.02	1,690.00	**1,600.00**	1,690.00	**1,600.00**	1,730.00	1,690.00
6	25	2,598.57	**2,640.00**	1.57	3,450.00	2,840.00	3,340.00	**2,640.00**	3,940.00	3,390.00
7	30	2,477.34	3,270.00	24.24	4,050.00	3,270.00	3,960.00	**3,150.00**	4,170.00	3,230.00
8	40	3,189.94	4,850.00	34.23	7,230.00	4,880.00	6,870.00	**4,840.00**	7,370.00	5,220.00

Table 6 shows CPLEX, SA and ILS results. Column W indicates the instance and columm n the number of aircraft that requested service in each instance. Columns UB and LB show the upper bound and lower bound found by CPLEX. The GAP column shows the gap found by CPLEX, $GAP = (UB - LB)/UB$. The results of the six variants are shown in columns $I, II, ..., VI$. The columns indicate the best solution in the 5 runs, respectively.

Looking at the values in bold in Table 6, we can consider that the fourth variant (IV) produced the largest number of best solutions. The second variant (II) also obtained advantageous results. The other variants found solution values greater than or equal to those of CPLEX up to Instance 4. For the remainder instances, the other variants found poor solutions.

We can affirm that the variants (II, IV, VI) of the ILS algorithm obtained the best solutions compared to the SA algorithm. Therefore, considering the ILS algorithm, note that by decreasing the possibilities of the local search from seven (variant II) to four (variant IV), the quality of the solution improves. Note that variant IV is formed by inter-route neighborhood structures and variants II and VI by inter-route and intra-route structures. In this problem, the ILS algorithm presented greater diversity with inter-route moves.

Adopting variant IV of the ILS algorithm and comparing its results in detail with those obtained by the MILP model in Table 7, it is noticeable that the method proposed here made it possible to reduce the average execution time (in seconds) of the algorithm in the 5 runs.

The $Time$ column shows the execution time of CPLEX. Column v shows the number of the used trucks. The columns $Best$, Avg, $Worst$, v_{min} and v_{max} indicate the best solution value, the average solution value, the worst solution value, the minimum number of used trucks and the maximum number of used trucks by ILS in the 5 runs, respectively. The Dev column represents the deviation of the solutions based on the best solution found ($Dev = 100(Worst - Best)/Best$).

Column T_{avg} designates the average execution time (in seconds) of the algorithm in the 5 runs. Table 7 shows the results.

Table 7. Comparison of CPLEX and ILS results with random neighborhood structure considering four proposed moves $N^{(1)}$, $N^{(2)}$, $N^{(4)}$ and $N^{(5)}$.

W	n	CPLEX					Algorithm ILS - Variation IV						
		LB	UB	$GAP(\%)$	$Time$	v	$Best$	Avg	$Worst$	Dev	T_{avg}	v_{min}	v_{max}
1	6	370.00	**370.00**	0.00	<1.00	1	**370.00**	370.00	370.00	0.00	<1.00	1	1
2	11	1,400.00	**1,400.00**	0.00	<1.00	2	**1,400.00**	1,400.00	1,400.00	0.00	1.20	2	2
3	15	828.68	**860.00**	3.64	2.29	1	**860.00**	860.00	860.00	0.00	<1.00	1	1
4	18	1,588.00	**1,800.00**	11.78	6.16	2	**1,800.00**	1,808.00	1,840.00	2.22	27.56	2	2
5	18	1,588.21	1,690.00	6.02	34.05	3	**1,600.00**	1,600.00	1,600.00	0.00	30.03	3	3
6	25	2,598.57	**2,640.00**	1.57	810.11	7	**2,640.00**	2,796.00	3,000.00	13.63	100.23	7	7
7	30	2,477.34	3,270.00	24.24	7,200.00	7	**3,150.00**	3,256.00	3,350.00	6.34	144.5	7	8
8	40	3,189.94	4,850.00	34.23	7,200.00	8	**4,840.00**	4,807.00	4,970.00	2.68	1,376.00	8	9

Since the ARP is NP-hard, it is expected that the exact approach will not be able to efficiently solve medium to large instances. This indeed happened and it can be seen in Table 7, where CPLEX proved solution optimality only for Instances 1 and 2 (with 6 and 11 service requests).

For Instances 3 to 6, which ranged from 15 to 25 service requests, CPLEX was unable to find an optimal solution. For Instances 7 and 8, which ranged from 30 to 40 service requests, CPLEX did not find an optimal solution after running for 2 h (time limit) obtaining gaps of 24.24% and 34.23%, respectively. ILS found slightly better results than CPLEX for Instances 5, 7 and 8. ILS had little deviation between the results of the 5 runs for these instances, 0.00, 6.34 and 2.68 respectively.

As it can be observed in Table 7, the ILS algorithm found solutions for all instances, including the optimal solution for Instances 1 and 2. For Instances 3 and 4, the solution found by ILS was the same found by CPLEX. For the remaining instances, the solution value found by ILS was between the LB and the UB found by CPLEX with reduced execution time.

The ILS heuristic provides good results at an affordable time when compared with the 2 h of CPLEX execution. Note that the average execution time for our largest instance (Instance 8) is of about 22 min. In civil aviation, flight schedules are planned ahead, and even if they suffer adjustments during an operational period, the times required by the proposed ILS algorithm are still acceptable. The results shown here confirm that the proposed algorithm is a valuable tool to help planning daily refueling truck routes at an airport.

6 Conclusions

This article addressed a problem that often occurs at airports: the problem of refueling aircraft using tank trucks. A mixed-integer linear programming

model and SA and ILS algorithms were designed for solving this problem. Variants of methods for generating initial solutions and generating neighborhoods were tested. These variants significantly affected the results. The computational experiments showed that CPLEX could solve only small instances of the problem and that the ILS algorithm is an effective approach to solve the problem. The proposed model can be used in any scenario of the aircraft refueling activity at airports and its approach is a significant advance in terms of preventing routes where flight schedule delays may occur. Moreover, the ability of considering trucks with different capacities makes ILS more applicable under real circumstances.

In future research, strategies to improve the efficiency of the presented algorithm are to be considered. Moreover, strategies can be studied to address instances that involve extremely unfeasible cases, where not all aircraft can be serviced without violating time restrictions. Meanwhile, we apply the proposed model in real practice.

Acknowledgments. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and the Fundação de Amparo à Pesquisa e Inovação do Espírito Santo (FAPES) (Grant Number 2020-LGJV3).

References

1. Babić, O.: Optimization of refuelling truck fleets at an airport. Transp. Res. Part B Methodologica **21**(6), 479–487 (1987)
2. Braekers, K., Ramaekers, K., Nieuwenhuyse, I.: The vehicle routing problem: state of the art classification and review. Comput. Ind. Eng. **99** (2015). https://doi.org/10.1016/j.cie.2015.12.007
3. Bräysy, O., Gendreau, M.: Vehicle routing problem with time windows, part i: route construction and local search algorithms. Transp. Sci. **39**(1), 104–118 (2005)
4. Du, Y., Zhang, Q., Chen, Q.: Aco-ih: an improved ant colony optimization algorithm for airport ground service scheduling. In: IEEE International Conference on Industrial Technology, pp. 1–6 (2008). https://doi.org/10.1109/ICIT.2008.4608674
5. Gamayanti, N., Sahal, M., Wibisono, A.: Optimization of vehicle routing problem with tight time windows, short travel time and re-used vehicles (vrptsr) for aircraft refueling in airport using ant colony optimization algorithm. J. Adv. Res. Electric. Eng. **2**(1), 43–46 (2018)
6. Gendreau, M., Tarantilis, C.D.: Solving large-scale vehicle routing problems with time windows: the state-of-the-art. Montreal (2010)
7. Kirkpatrick, S.: Optimization by simulated annealing: quantitative studies. J. Stat. Phys. **34**(5–6), 975–986 (1984)
8. Lin, S.W., Ying, K.C., Lee, Z.J., Chen, H.S.: Vehicle routing problems with time windows using simulated annealing. In: 2006 IEEE International Conference on Systems, Man and Cybernetics, vol. 1, pp. 645–650 (2006). https://doi.org/10.1109/ICSMC.2006.384458
9. Osman, I.H.: Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Ann. Oper. Res. **41**, 421–451 (1993). https://doi.org/10.1007/BF02023004

10. Rochat, Y., Taillard, É.D.: Probabilistic diversification and intensification in local search for vehicle routing. J. Heurist., 147–167 (1995). https://doi.org/10.1007/BF02430370
11. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper. Res. **35**(2), 254–265 (1987)
12. Wang, Z., Li, Y., Hei, X., Meng, H.: Research on airport refueling vehicle scheduling problem based on greedy algorithm. In: Huang, D.-S., Bevilacqua, V., Premaratne, P., Gupta, P. (eds.) ICIC 2018. LNCS, vol. 10954, pp. 717–728. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95930-6_73