

Optimize a Thru Flight Inspection of a Fighting Falcon using Routing Algorithms

2011-01-2750

Published
10/18/2011

Athanasios Lois, Nikolaos Bertos and Athanasios Ziliaskopoylos
University of Thessaly

Copyright © 2011 SAE International

doi:10.4271/2011-01-2750

ABSTRACT

The process of checking inspection points on combat aircraft after a mission, is critical for their operational readiness. Manufacturers include specific inspection procedures in their maintenance handbooks. These procedures consist of detailed instructions for each check, the minimum time required to complete each check as well as a suggested sequence. However, it has been observed, that technical crews can complete inspection in less time than suggested by the manual, without violation of the time prescribed for each inspection point. In this work we will try to apply routing algorithms, to improve the total inspection time, by finding the optimal inspection sequence. This will be achieved without violating any constraint set by the manufacturer, except for the small reduction of the service time on some points. The algorithms we will use is the algorithmic set usually applied for the well-known PDPTW (pickup and delivery problem with time windows). Every inspection area is considered as a network $G(N,A)$ consisting of nodes N , and arcs A . Every node (Inspection Point) is characterized by the Time Window, defined as the time interval earliest-latest $[e_i, l_i]$, during which the point should be inspected. It is also characterized by the Service Time s_i , defined as the time required to complete the check on this point. Every arc between two nodes describes the time c_{od} , needed for the technician to move from the first node (Origin) to second node (Destination). These algorithms will be run to produce the optimal path (sequence) or the best attainable suboptimal

path that minimizes the objective
$$\sum_{v \in V} \sum_{i \in P^{all}} \sum_{j \in P^{all}} x_{ij}^v c_{ij}$$
.

The data set consists of real data from inspections logs.

1. Introduction

The inspection procedures for the maintenance of aircraft require strict adherence to a specific sequence. The manufacturers have produced a very large number of manuals on maintenance procedures. These manuals provide extremely detailed information about each step of the maintenance procedure. Given the fact that combat aircraft are extremely critical for a country's defense, these manuals emphasize the need for strict adherence to the procedural steps and their order. They promote this as the best way to ensure that aircraft will be operational. However the time parameter arises in many situations, especially when time is a critical factor in conflict situations. To the best of the authors' knowledge there is no clear description of how these maintenance procedures may be optimized for time, while keeping all other requirements satisfied. Although the research literature dealing with airline operations is quite rich, none of the reported works deals with the problem that we address in this work. Mattila et al.[5] have addressed some of the issues we dealing with in this paper, using discrete event simulation, to model the maintenance of military aircraft in the Finnish Air Force. They studied the impact of the application of such an approach on aircraft availability in both normal and conflict operational conditions. The Department of the Air Force of the United States in association with the constructor of the aircraft, supplies all client countries with the basic maintenance literature. There are technical orders according to the level of inspection, the aircraft's systems and the type of maintenance. We consulted a technical order which thoroughly describes thru flight inspection and quick turnaround procedures. In this paper a new methodology for time optimization is presented, in the hope that it could be a useful concept to the maintenance personnel and to the maintenance manuals authors.

This methodology is based on algorithms used to solve the problem PDWTW (pickup and deliver with time windows problem). The PDWTW problem class has received considerable attention in the literature. Several versions of the PDWTW problem have been studied over the past 30 years. Some of the major developments in the PDWTW Problem are discussed. The single-vehicle problem has been first studied by Psaraftis [1], who developed an exact dynamic programming algorithm for the case where time windows are imposed on each pick-up and drop-off. Jaw et al. [2] have analyzed a version of the problem where windows are imposed on the pick-up time of pickup requests and on the drop-off time of delivery requests. The basic concept of the proposed algorithm is simple. The algorithm tests the feasible assignment for the current request (pickup or delivery) for each available vehicle. For each assignment a COST function is calculated. From all available feasible insertions the one with the smaller COST is selected. Diana-Dessouky [3] presented a parallel regret insertion heuristic to solve a PDWTW problem. A new route initialization procedure is implemented, that keeps into account both the spatial and the temporal aspects of the problem. A regret insertion is then performed to serve the remaining requests. Lois A.[4] presented an advanced version of insertion heuristic that improves the vehicle usage, eliminating this way a drawback of Jaw algorithm. A regret insertion algorithm by the same author focused on the online problems where the demand is not known a-priori. In addition Lois A.[4] presented an exact version with dynamic programming that finds the absolutely optimal solution which handles any kind of PDWTW problem, but is limited to small problems. With this in mind, our work in this paper treads on relatively new ground, contributing an important practical problem, namely the minimization of maintenance time in urgent situations.

The remainder of this paper is structured as follows.

Section 2, we describe in detail the problem formulation and its MILP model.

Section 3 contains the proposed routing algorithms for solving large and small problem instances.

Section 4 describes the:

- types of inspection we are dealing with.
- application of our algorithms on inspections drawn from the HAF.
- computational results.

Section 5 summarizes this work.

2. Problem Description and Formulation

In this study we formulate the inspection in such a way such as to make it possible to apply routing algorithms. Namely

we shall use the inspection data in order to create a network, which consists of nodes and arcs. Let's call this graph $G(N,A)$. We shall use the traditional notation for routing algorithms, where N describes the set of Nodes while A describes the sets of Arcs between the Nodes. Usually routing algorithms refer to vehicles having to satisfy transportation needs. In our case the role of vehicles can be replaced by specialized inspection personnel. Every inspection point is considered as a node of our network. From every node we define arcs that lead to each of the rest of the nodes. Arcs represent an abstract cost which in this case will be the time required to move from one node (inspection point) to the other. The type of algorithms we will use in our case is Pickup and Deliver algorithms with Time Windows (PDWTW). The parameters for the application of these algorithms are as follows:

- Network consists of nodes and arcs. Each node describes an inspection point. Each arc describes the time required to move from one inspection point to the other.
- Vehicles - Inspectors Personnel in our case - that will visit each inspection node.
- Parameters concerning the vehicles - inspectors such as the maximum work time at each inspection point (node).
- Parameters concerning the inspection points like the time window in which the inspection point should be visited (served).

A limited notation of the PDWTW adapted to our inspection problem follows:

$V = \{1, 2, 3, \dots, |V|\}$ = Set of Inspectors
 N = is the Number of inspection points. Each inspection point consists of one origin point and one exit point.
 $VSD = \{0\}$ = Set of "starting" point for all inspectors.
 $VED = \{2N+1\}$ = Set of "ending" point for all inspectors.
 $VSED = \{0, 2N+1\}$ = Set of "start" & "end" points for all inspectors.
 $P^+ = \{1, 2, \dots, N\}$ = Set of nodes describing the inspection point in the network.
 $P^- = \{N+1, N+2, \dots, 2N\}$ = Set of nodes describing the exit point in accordance to every inspection point which belongs to P^+
 $P = P^+ \cup P^-$ = Union of P^+ and P^-
 $P^{all} = P \cup VED \cup VSD$ = Union of P, VED, VSD
 $P^{ved} = P \cup VED$ = Union of P and VED
 $P^{vsd} = P \cup VSD$ = Union of P and VSD
 $[e_i, l_i]$ = time windows [earlier, late] for each inspection point $i \in P^{all}$.
 τ_{ij} = travel time from node i to j where $i, j \in P^{all}$.
 c_{ij} = travel cost from node i to j where $i, j \in P^{all}$.
 s_i = service time at node $i \in P^{all}$.
 $x_{ij}^v, i, j \in P^{all}, i \neq j$ Is a binary variable describing whether arc $i \rightarrow j$ is used by vehicle v .
 $T_i^v, i \in P^{all}$ Time variable, which describes the time that the service begins at node i .

As pickup nodes we define all nodes that need an inspection. As delivery nodes we call the exit points for each specific

inspection (point). Every pickup point is paired to a delivery point. We are interested only for the pickup - inspection points while delivery points can be the same for all pairs.

Time windows describe the time interval within which an inspection point must be visited. Suppose that inspection time starts at time 0 (expressed in seconds) and the earliest time for the inspection of a specific point is at least 10 minutes (or otherwise 600 seconds) then the time window can be described as $[600, x]$ where x an arbitrarily large number. For every inspection point we have the service time which is the inspection time specified in the manuals.

A commonly used objective function minimizes total inspection time as follows:

$$\sum_{v \in V} \sum_{i \in P^{all}} \sum_{j \in P^{all}} x_{ij}^v c_{ij}$$

The Mixed Integer Linear Program that can solve the above defined problem is as follows:

$$\min \sum_{v \in V} \sum_{i \in P^{all}} \sum_{j \in P^{all}} x_{ij}^v c_{ij} \quad (1)$$

Subject to:

$$\sum_{v \in V} \sum_{j \in P^{all}} x_{ij}^v = 1, i \in P^+ \quad (2)$$

$$\sum_{j \in P^{all}} x_{ij}^v - \sum_{j \in P^{all}} x_{ji}^v = 0, i \in P, v \in V \quad (3)$$

$$\sum_{j \in P^+} x_{ij}^v = 1, i \in VSD, v \in V \quad (4)$$

$$\sum_{i \in P^-} x_{ij}^v = 1, j \in VED, v \in V \quad (5)$$

$$\sum_{j \in P^{all}} x_{ij}^v - \sum_{j \in P^{all}} x_{j(N+i)}^v = 0, i \in P^+, v \in V \quad (6)$$

$$T_i^v + s_i + \tau_{i(i+N)} \leq T_{(i+N)}^v, i \in P^+, v \in V \quad (7)$$

$$x_{ij}^v = 1 \Rightarrow T_i^v + s_i + \tau_{ij} \leq T_j^v, i, j \in P, v \in V \quad (8)$$

$$x_{ij}^v = 1 \Rightarrow T_i^v + s_i + \tau_{ij} \leq T_j^v, i \in VSD, j \in P^+ \quad (9)$$

$$x_{ij}^v = 1 \Rightarrow T_i^v + s_i + \tau_{ij} \leq T_j^v, j \in VED, i \in P^- \quad (10)$$

$$e_i \leq T_i^v \leq l_i, i \in P, v \in V \quad (11)$$

$$e_i \leq T_i^v \leq l_i, i \in VSD, v \in V \quad (12)$$

$$e_i \leq T_i^v \leq l_i, i \in VED, v \in V \quad (13)$$

$$x_{ij}^v, \text{ binary}$$

The Objective function (1) minimizes the total time produced by all inspectors. Constraint (2) guarantees that one inspector is leaving one inspection point and heading to a unique exit point. Constraint (3) guarantees that, the number of inspectors entering inspection point i will leave inspection point i .

Constraint (4) guarantees that, there is only one unique inspection point after the “start” point for each inspector.

Constraint (5) guarantees that, there is one unique exit point before “end” point for each inspector.

Constraint (6) is the pairing constraint, meaning that the same inspector visits node $i, i+n$.

Constraints (7,8,9,10) are precedence constraints ensuring that node i is visited before node $i+n$ and the compatibility between routes and schedule.

Constraint (11) ensures that each request time windows is not violated.

Constraints (12,13) ensure that each inspector time window's constraint is not violated.

The above MILP problem can be solved in various ways. We can refer to some commercial solvers like AMPL/CPLEX combination. However given the fact that it has been proven [8] that the PDWTW problem is NP hard, these solvers can't solve large scale problems due to fact that these solvers always produce the absolute optimal solution. This is the main reason why we choose to use heuristic algorithms. Heuristic algorithms can handle large and very large scale programs however they produce sub-optimal solutions but optimized enough in comparison to random schedules. In this study we will use two heuristic algorithms and one exact

algorithm. Heuristic algorithms will be used for the type of inspections where the number of inspection points is larger than ten points while the exact algorithm will be used for the inspections where the number of inspection points is less than or equal to ten. Following paragraphs refer with details to these algorithms.

3. Proposed Routing algorithms

3.1. The Insertion Algorithm “InsertionH”

The insertion algorithm that we developed is a variation of the one presented by Jaw [2]. The basic function of this insertion algorithm is to make an initial assignment for each inspection point. This solution can be refined later by other optimization heuristics. In a preprocessing step the algorithm creates an empty route for each inspector. Then for every inspection request the algorithm finds the best position -in terms of cost- to insert the request by searching all inspector routes. If this position is found, then the request is inserted; otherwise, if it is not possible to find at least one position where the request could be inserted, the inspection request is rejected. At the end of the execution, specific inspection requests have been assigned to specific inspectors.

Notation

$V = \{1, 2, 3, \dots, |V|\}$ = Set of Inspectors
 $R_{V\{v=1, 2, \dots, |V|\}}$ = is the route schedule for every $v \in V$
 $RC_{V\{v=1, 2, \dots, |V|\}}$ = is the route cost for the for every $v \in V$
 N = is the number of inspection requests.
 $TR_{i\{i=1, 2, \dots, N\}}$ = is the sequence of inspection requests.
 $EPT_{i\{i=1, 2, \dots, N\}}$ = Earlier inspection time of inspection request i
 BIP_{iv} = the best insertion position of i -th inspection task TR_i to Route schedule R_v
 $MinCostBIP_{iv}$ = is the cost of the best insertion position of the inspection request TR_i to Route schedule R_v

Pseudo code

Step0: for every v in V build an empty R_v
Step1: Sort R_v in descending order according to RC_v
Step2: Sort TR_i in ascending order according to request EPT_i
Step3: for every $TR_{i\{i=1, 2, \dots, N\}}$
Step3.1: Sort R_v in ascending order according to RC_v
Step3.2: for every $R_{v\{v=1, 2, \dots, |V|\}}$ do
find the BIP_{iv}
Calculate $MinCostBIP_{iv}$
Step3.3: if no $BIP_{iv\{v=1, 2, \dots, |V|\}}$ found goto step 3.6
Step3.4: From all $MinCostBIP_{iv}$ select the minimum one
Step3.5: Assign that TR_i to the appropriate R_v and goto step3
Step3.6: Reject that inspection request and repeat step3
end for

3.2. The Regret Algorithm “RegretH”

The Regret based Heuristic algorithm aims to reduce the chances to get trapped in a severely suboptimal solution. It does this by considering the opportunity which is defined as the difference between the cost actually calculated by the

algorithm (total time in our case) and the cost calculated for a better position that could be obtained if a “different” course of action had been chosen. The basic idea is to use a fast algorithm in order to quickly get a feasible suboptimal solution. Then, based on the existing solution, we try to find a better solution by reassigning the most expensive inspection tasks. The reallocation which provides the largest gain in the system is chosen. Our regret algorithm uses the original regret concept where the absolute difference between the “best” lower cost and the second “best” lower cost alternative is used as a metric for guiding problem solving.

Notation

$V = \{1, 2, 3, \dots, |V|\}$ = Set of Inspectors.
 $R_{V\{v=1, 2, \dots, |V|\}}$ = is the route schedule for every $v \in V$.
 $|R_v|$ = size of route schedule defined as the number of inspection requests included in this route.
 $RC_{v\{v=1, 2, \dots, |V|\}}$ = is the route cost for every $v \in V$.
 $TRC = \sum_{v=1}^{|V|} RC_v$ is the total cost for all routes.
 $DR_{kv\{k=1, 2, \dots, |R_v|, v=1, 2, \dots, |V|\}}$ = the assigned k inspection request belongs to route v .
 $DMaxR_v \{v=1, 2, \dots, |V|\}$ = the maximum inspection request cost concerning route v .
 $RCostDMaxR_{vm\{v=1, 2, \dots, |V|, m=1, 2, \dots, |V|, v \neq m\}}$ = is the extra cost incurred in route m , if we move the inspection request (with the maximum cost) that belongs to route v to the route m .
 $CostMatrix[v, m] \{v=1, 2, \dots, |V|, m=1, 2, \dots, |V|\}$ = is 2-d matrix cost for the Regret Algorithm. Diagonal positions $CostMatrix[v, v]$ contains $DMaxR_v$ while other positions $CostMatrix[v, m] \{m > v\} \{m=1, 2, \dots, |V|, v=1, 2, \dots, |V|\}$ contains $RCostDMaxR_{vm}$
 $RegretCostMatrix[v] \{v=1, 2, \dots, |V|\}$ = is the one dimensional matrix containing the maximum value of the difference between the element with the least $RCostDMaxR_{vm\{v=1, 2, \dots, |V|, m=1, 2, \dots, |V|\}}$ and the corresponding $DMaxR_v$.

Pseudocode

Step0: Run the “InsertionH” Algorithm
Calculate TRC
 $NewTRC = TRC + \text{Arbitrary small value}$
Step 1: While ($NewTRC - TRC > 0$)
 $NewTRC = TRC$
Step 1.1: for every v in V
for every $DR_{kv\{k=1, 2, \dots, |R_v|\}}$
find $DMaxR_v$
 $CostMatrix[v, v] = DMaxR_v$
Step 1.2: for every $DMaxR_v \{v=1, 2, \dots, |V|\}$
for every m in V
Find $RCostDMaxR_{vm\{v=1, 2, \dots, |V|, m=1, 2, \dots, |V|, v \neq m\}}$
 $CostMatrix[v, m] = RCostDMaxR_{vm}$
Step 1.3: for every v in V
 $RegretCostMatrix[v] = \text{positive } Max(DMaxR_v - CostMatrix[v, m]) \{m=1, 2, \dots, |V|\}$
the greatest positive difference.
Move inspection request with $DMaxR_v$ cost from route v to route m
Step 1.4: Calculate TRC

3.3. The Exact Algorithm “ExactDP”

The Exact algorithm is a DP implementation. The algorithm examines all possible valid combinations of inspectors and inspection request assignments. During the execution the algorithm improves the solution continuously i.e., when a combination of possible request assignments has higher cost than a previous solution, then further searching based on this combination is discontinued. The algorithm searches for the exact solution in all feasible solution combinations. Due to the nature of the problem, if we want to get the exact solution for the PDWTW problem it is necessary to evaluate every feasible combination of inspection requests and inspectors that we cannot rule out based on some technique. Given the problem's combinatorial nature it is obvious that the number of combinations explodes, making any exact approach meaningless for realistically sized problems. The algorithm starts with an initial combination. Based on that combination, it finds the next one, which contains the parent - previously produced - combination. The process continues that way, until finding the “final” or “bottom” combination.

Notation

$T = \{t_1, t_2, \dots, t_n\}$ = the set of unassigned inspection requests

$T_k = \{t_1, t_2, \dots, t_k\}, k \leq n$ = the set of assigned requests in a feasible solution. Starting value of T_k is $\{\}$.

S_{T_k} = the set of feasible solutions resulting from all possible inspection requests – belongs to set T_k - combinations concerning routing and scheduling.

$MinCost$ = represents the lower cost limit that allows the exact algorithm to reject all feasible solutions with larger costs.

$MinCost$ could be:

Initial Arbitrary Large Value

Or

The cost of a feasible solution produced by the “InsertionH” algorithm in a preprocessing phase

Or

The cost of a feasible solution - containing all requests of set T - produced by the “ExactDP” algorithm.

$PS(S_{T_k}, t_i, MinCost)$ = returns all feasible solutions by combining each solution contained in S_{T_k} , with the new inspection request t_i , given the $MinCost$.

Pseudo-code

Step0: $T_k = \{\}, k = 0$

$S_{T_k} = \{\}$

$MinCost = \text{Min}(\text{arbitrary large number, Cost of the best feasible solution produced by any InsertionH algorithm})$

Step1: For every request t_i in set T

Step2: $S_{T_{k+1}} = PS(S_{T_k}, t_i, MinCost)$

Step3: Reject from $S_{T_{k+1}}$ all feasible solutions with cost more than $MinCost$

Step4: $T_{k+1} = T_k \cup t_i, T = T \cap t_i$

Step5: If $T \neq \{\}$ recursive call of Step1
else

$MinCost = \text{Min}(S_{T_{k+1}})$

Mark solution with cost equal to $MinCost$

Recursive return to the previous state

End if

The recursive relationship and the $S_{T_{k+1}} = PS(S_{T_k}, t_i, MinCost)$ problem reduce to $S_{T_k=n}$. In this case set $S_{TK} = n$ contains all inspection requests included in T . After the evaluation of every feasible solution included in $S_{TK=n}$ the optimal solution is finally found.

4. Computational results

To prove the gain acquired in terms of time by the application of the algorithms presented above, we simulated two different types of inspection concerning combat aircraft under the general name FXX. These types of aircrafts are common to the majority of NATO country members. The first one is called Thru-Flight inspection and can be considered as a meticulous and time consuming inspection. The second is called Quick-Turnaround inspection and is a quick inspection which takes place between landing and take-off in urgent situations.

4.1. Thru- flight Inspection

This type of inspection consists of 93 inspection points. The minimum allowable visiting time and the service time for each inspection point is defined by the aircraft manufacturer. Table 1 presents all inspections points for the Thru-Flight inspection. The “Description” column defines the description of the inspection point, the “T_{literature}” column defines manufacturers proposed inspection time, and the “T_{HAF time}” column defines the inspection time derived from the experienced HAF technicians.