ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

Μεταπτυχιακή Εργασία

ΜΕΙΚΤΗ ΑΚΕΡΑΙΑ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΕΛΑΧΙΣΤΩΝ ΤΕΤΡΑΓΩΝΩΝ ΓΙΑ ΣΧΕΔΙΑΣΜΟ ΠΤΗΣΕΩΝ ΚΑΙ ΣΥΝΤΗΡΗΣΕΩΝ ΑΕΡΟΣΚΑΦΩΝ ΑΠΟΣΤΟΛΩΝ

υπό

ΕΥΤΥΧΙΑ ΚΩΣΤΑΡΕΛΟΥ

Διπλωματούχου Μηχανικού Παραγωγής και Διοίκησης της Πολυτεχνικής Σχολής Ξάνθης,

ΔΠΘ, 2006

Υπεβλήθη για την εκπλήρωση μέρους των

απαιτήσεων για την απόκτηση του

Μεταπτυχιακού Διπλώματος Ειδίκευσης

2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

Μεταπτυχιακή Εργασία

ΜΕΙΚΤΗ ΑΚΕΡΑΙΑ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΕΛΑΧΙΣΤΩΝ ΤΕΤΡΑΓΩΝΩΝ ΓΙΑ ΣΧΕΔΙΑΣΜΟ ΠΤΗΣΕΩΝ ΚΑΙ ΣΥΝΤΗΡΗΣΕΩΝ ΑΕΡΟΣΚΑΦΩΝ ΑΠΟΣΤΟΛΩΝ

υπό

ΕΥΤΥΧΙΑ ΚΩΣΤΑΡΕΛΟΥ

Διπλωματούχου Μηχανικού Παραγωγής και Διοίκησης της Πολυτεχνικής Σχολής Ξάνθης,

ΔΠΘ, 2006

Υπεβλήθη για την εκπλήρωση μέρους των

απαιτήσεων για την απόκτηση του

Μεταπτυχιακού Διπλώματος Ειδίκευσης

2009

Η έγκριση της μεταπτυχιακής εργασίας από το Τμήμα Μηχανολόγων Μηχανικών της Πολυτεχνικής Σχολής του Πανεπιστημίου Θεσσαλίας δεν υποδηλώνει αποδοχή των απόψεων του συγγραφέα (Ν. 5343/32 αρ. 202 παρ. 2).

Εγκρίθηκε από τα Μέλη της Τριμελούς Εξεταστικής Επιτροπής:

Πρώτος Εξεταστής (Επιβλέπων)	Δρ. Γεώργιος Κοζανίδης Λέκτορας, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Θεσσαλίας
Δεύτερος Εξεταστής	Δρ. Γεώργιος Λυμπερόπουλος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών Πανεπιστήμιο Θεσσαλίας
Τρίτος Εξεταστής	Δρ. Αθανάσιος Ζηλιασκόπουλος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες

Πρώτα απ' όλα, θέλω να ευχαριστήσω τον επιβλέποντα της μεταπτυχιακής μου εργασίας, Λέκτορα Γεώργιο Κοζανίδη, για την πολύτιμη βοήθεια και καθοδήγησή του κατά τη διάρκεια της δουλειάς μου. Επίσης, είμαι ευγνώμων στα υπόλοιπα μέλη της εξεταστικής επιτροπής της μεταπτυχιακής εργασίας μου, Καθηγητές κκ Γεώργιο Λυμπερόπουλο, Αθανάσιο Ζηλιασκόπουλο, για την προσεκτική ανάγνωση της εργασίας μου και για τις πολύτιμες υποδείξεις τους. Οφείλω ευχαριστίες σε όλους τους καθηγητές μου, οι οποίοι συνετέλεσαν καθοριστικά στην απόκτηση του θεωρητικού μου υποβάθρου, το οποίο αποδείχτηκε απαραίτητο για την ολοκλήρωση μιας ιδιαίτερα απαιτητικής εργασίας. Επίσης, ευχαριστώ το Γιώργο Μαμάνη για την υποστήριξη του κατά τη διάρκεια των μεταπτυχιακών μου σπουδών. Πάνω απ' όλα, είμαι ευγνώμων στους γονείς μου, Σπύρο και Pitsa Κωσταρέλου, και στην αδερφή μου Ρούλα για την ολόψυχη αγάπη και υποστήριξή τους όλα αυτά τα χρόνια, και τους αφιερώνω αυτήν την μεταπτυχιακή εργασία.

Ευτυχία Κωσταρέλου

ΜΕΙΚΤΗ ΑΚΕΡΑΙΑ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΕΛΑΧΙΣΤΩΝ ΤΕΤΡΑΓΩΝΩΝ ΓΙΑ ΣΧΕΔΙΑΣΜΟ ΠΤΗΣΕΩΝ ΚΑΙ ΣΥΝΤΗΡΗΣΕΩΝ ΑΕΡΟΣΚΑΦΩΝ ΑΠΟΣΤΟΛΩΝ

ΕΥΤΥΧΙΑ ΚΩΣΤΑΡΕΛΟΥ

Πανεπιστήμιο Θεσσαλίας, Τμήμα Μηχανολόγων Μηχανικών, 2009

Επιβλέπων Καθηγητής: Δρ. Γεώργιος Κοζανίδης, Λέκτορας Βελτιστοποίησης Συστημάτων Παραγωγής / Υπηρεσιών

Περίληψη

Η παρούσα μελέτη αφορά τη βελτιστοποίηση ενός μη γραμμικού μεικτού ακέραιου προβλήματος ελαχίστων τετραγώνων, για το σχεδιασμό πτήσεων και συντηρήσεων αεροσκαφών αποστολών. Η επίλυση του προβλήματος έγινε με τη χρήση ενός αναλυτικού αλγόριθμου αναζήτησης, ο οποίος βασίστηκε σε έναν προγενέστερο αλγόριθμο τετραγωνικού προγραμματισμού (quadratic programming). Φυσικά, επί του συγκεκριμένου αλγόριθμου έγιναν όλες οι τροποποιήσεις οι οποίες κρίθηκαν απαραίτητες για την επίλυση του προβλήματος που πραγματεύεται η παρούσα εργασία.

Αρχικά, παρουσιάζεται το κύριο μοντέλο στο οποίο βασίστηκε η παρούσα μελέτη. Στη συνέχεια, αναπτύσσεται ένα θεωρητικό υπόβαθρο σχετικά με την επίλυση του συγκεκριμένου προβλήματος, και βάσει αυτού, η μεθοδολογία που μπορεί να χρησιμοποιηθεί στην πράξη και ο τρόπος με τον οποίον αυτή μπορεί να εφαρμοστεί στο συγκεκριμένο πρόβλημα.

Παράλληλα, περιγράφονται και ερεξηγούνται οι προσθήκες στο βασικό μοντέλο και αναπτύσσεται το μαθηματικό μοντέλο του προβλήματος που επιλύθηκε σε αυτή τη μελέτη, το οποίο αποτελεί και το κύριο μέρος αυτής της μεταπτυχιακής εργασίας. Ο κώδικας που χρησιμοποιήθηκε για την υλοποίηση του αλγόριθμου, γράφτηκε σε γλώσσα προγραμματισμού C, και τα απαραίτητα αριθμητικά πειράματα εκτελέστηκαν σε έναν από τους εξυπηρετητές του πανεπιστημίου.

Τέλος, παρέχονται τα αποτελέσματα των πειραμάτων αυτών, δύο αριθμητικά παραδείγματα για την κατανόηση της διαδικασίας επίλυσης, καθώς και συμπεράσματα και προτάσεις για μελλοντικές βελτιώσεις αυτού του προβλήματος.

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

Postgraduate Work

MIXED INTEGER LEAST SQUARES OPTIMIZATION FOR FLIGHT AND MAINTENANCE PLANNING OF MISSION AIRCRAFT

by

EFTYCHIA KOSTARELOU

PRODUCTION AND MANAGEMENT ENGINEERING, DEMOCRITUS UNIVERSITY OF THRACE, 2006

Submitted in partial fulfillment

requirements for

Postgraduate Specialization Diploma

2009

The approval of this postgraduate work by the Department of Mechanical Engineering of the School of Engineering of the University of Thessaly does not imply acceptance of the writer's opinions (Law 5343/32 article 202 par.2).

Approved by:

First Examiner (Supervisor)	Dr. George Kozanidis Lecturer, Department of Mechanical Engineering, University of Thessaly
Second Examiner	Dr. George Liberopoulos Professor, Department of Mechanical Engineering, University of Thessaly
Third Examiner	Dr. Athanasios Ziliaskopoulos Professor, Department of Mechanical Engineering, University of Thessaly

Acknowledgements

First and foremost, I want to thank my postgraduate work supervisor, Lecturer Dr. George Kozanidis, for his valuable help and guidance throughout this work. I am also grateful to the other members of the examining committee of my postgraduate work, Dr. George Liberopoulos, Dr Athanasios Ziliaskopoulos, for the close examination of my work and for the valuable knowledge I was given during my post-graduate studies. I owe grateful thanks to all the teachers I learned from, so as to obtain the essential theoretical background for the completion of a demanding study. Also, I thank Giorgo Mamani for his supporting during my postgraduate studies. Above all, I am grateful to my parents, Spyro and Ritsa, and my sister Roula for their wholehearted love and constant support, to whom I dedicate this postgraduate work.

Eftychia Kostarelou

MIXED INTEGER LEAST SQUARES OPTIMIZATION FOR FLIGHT AND MAINTENANCE PLANNING OF MISSION AIRCRAFT

EFTYCHIA KOSTARELOU

University of Thessaly, Department of Mechanical Engineering, 2007

Supervising Professor: Dr. George Kozanidis, Lecturer in Optimization Methods of Production/ Service Systems

Abstract

This thesis considers a mixed integer least squares optimization problem for flight and maintenance planning of mission aircraft. The problem is solved through an exact search algorithm, which is based on an existing quadratic programming algorithm. Of course, the proposed algorithm includes all necessary modifications that were required for the solution of the problem that we address in this work.

Initially, we introduce the basic model upon which the present work builds. Then, we develop the theoretical background required for the solution of this problem, we introduce a methodology that can be used for that purpose, and we illustrate how this methodology can be applied for its solution.

At the same time, we describe and illustrate the adaptations that were made to the basic model and we develop the mathematical model, which constitutes the main part of this thesis. The code that was used for the implementation of the algorithm was written in C programming language, and the necessary computational experiments were performed on one of the university's servers.

Finally, we provide the results of the experiments that we conducted, we illustrate the solution procedure through two numerical examples, and we summarize our conclusions and suggestions for future problem enhancements.

Contents

List of Tables

List of Figures

Chapter 1 INTRODUCTION

1.1 General Description

The Air Force and the commercial airline industry have several similarities, but also exhibit significant differences. Safety is the most important factor in both industries. However, while maximization of profit is naturally the overall objective in the commercial airline industry, maximization of the readiness to respond to external threats is the main objective in the Air Force. Therefore, military aircraft operational problems should generally be treated differently than traditional problems arising in the commercial airline industry.

Despite this crucial difference, any aircraft, whether military or civilian, must be grounded for maintenance after it has completed a certain number of flight hours since its last maintenance check. The safety standards used by Air Force organizations of different countries are often similar, due to the fact that they are usually prescribed by the aircraft manufacturer and there are a few such manufacturers worldwide.

Flight and Maintenance Planning (FMP) addresses the question of which available aircraft should fly and for how long, and which grounded aircraft should perform maintenance operations in a group of aircraft that comprise a unit. FMP is an important decision making problem arising at the operation level of numerous types of mission fleets, involving military or fire-fighting aircraft, rescue choppers, etc. FMP decisions affect more than just economic

performance measures. In the context of military planning that we study in this work, for example, the FMP problem arises as a routine decision making problem in the typical operation of a combat wing of the Hellenic Air Force (HAF). The HAF is primarily responsible for Greece's national air defense. Therefore, a good/poor flight and maintenance plan can have a serious impact on national security, in this particular application.

In this thesis, we present a nonlinear mixed integer optimization model for a special case of Problem FMP, in which the planning horizon consists of a single time period and all aircraft belong to the same squadron. Typically, each aircraft is distinguished by its availability status. By "available" we refer to the aircraft that can participate in missions, and by "grounded" we refer to the aircraft that are currently undergoing maintenance operations, and are therefore unable to participate in missions. An available aircraft must be grounded for service as soon as it finishes its remaining flight hours, while a grounded aircraft becomes available as soon as it finishes its maintenance service.

For the maintenance needs of the unit, there exists a station, responsible for providing service to the aircraft of the unit. This station has certain space and time capacity capabilities. Given the flight requirements of the unit, and the physical constraints that stem from the capacity of the maintenance station, the objective is to issue a flight and maintenance plan for each individual aircraft, so that some appropriate measure of effectiveness is optimized. The most appropriate measure of effectiveness for this problem is to maximize the balance between the remaining flight times of the available aircraft, and the remaining maintenance times of the grounded aircraft. In other words, our objective is to maximize the smoothness of the distribution of the total residual flight and maintenance times. In Chapter 4, we introduce a mathematical formulation that addresses this effectively.

In order to solve this problem, we introduce an exact nonlinear programming algorithm, which is based on an existing quadratic programming algorithm. Of course, the proposed algorithm includes all necessary modifications that were necessary in order to address the special requirements of the problem under consideration.

1.2 Structure of Postgraduate Work

The remaining of this postgraduate work is structured as follows:

In Chapter 2, we review related works that have been published in the past, most of which refer to various applications encountered in the commercial airline industry, and few of which refer to problems encountered in the Air Force.

In Chapter 3, we present a detailed description of the problem under consideration. We give a thorough insight into the various aspects of the problem, and we lay the foundation for the mathematical formulation that follows.

In Chapter 4, we introduce the mathematical model that we develop in this thesis and we elaborate on the definition of its objective and constraints.

In Chapter 5, we provide some insight into our problem and we develop an analytical methodology that can be used for its solution. We also present the pseudo-code that illustrates the various steps of this methodology.

In Chapter 6, we analyze the computational complexity of the proposed algorithm. The same chapter also includes the computational results of the experiments that were conducted, after the proposed algorithm was implemented in C programming language. The chapter concludes with two numerical examples, which illustrate the proposed methodology.

In Chapter 7, we present the conclusions that we reached from the analysis of our results and we point to some promising directions for future research.

Appendix A contains the C Programming Language implementation of the proposed algorithm. Appendix B contains the C Programming Language implementation of the algorithm that was used to generate the random problem instances.

Chapter 2 LITERATURE REVIEW

In this chapter, we review works that have been published in the past, which refer to various types of problems related to the flight and maintenance planning problem. The research literature dealing with airline operations is quite rich. Most of the published research in this area, however, has been directed towards problems in the commercial airline industry, which have different objectives and requirements than those in the Air Force.

Several authors have presented reviews of models and methods for problems related to airline operations. Arguello et al. (1997) study models and methods for dynamic management of airline operations in case of irregular situations. Gopalan and Talluri (1998) are survey models and solution techniques for various airline problems that include fleet assignment and maintenance routing decisions. Barnhart et al. (2003) present an overview of several important areas of operations research applications in the air transport industry, as well as a brief summary of the state of the art.

In the context of military aircraft operations, Radosavljevic and Babic (2000) consider the problem of determining the optimal assignment of fighter plane formations to enemy formations and solve it via fuzzy logic and integer linear programming. Kurokawa and Takeshita (2004) propose a neural network method for air transportation planning in the Japan Air-Self Defense Force. This method partitions the master problem into three sub-problems which are successively solved by three neuron blocks. Yeung et al. (2007) develop a model-based methodology for mission assignment and maintenance scheduling of systems with

multiple states. The authors utilize heuristics and simulation to solve the model and illustrate its application on a hypothetical scenario of a fleet of aircraft.

FMP is an important decision making problem encountered in several diversified areas (see for example Jardine and Hassounah, 1990). The published research dealing with the FMP problem as approached in this work, however, is rather limited. Sgaslik (1994) introduces a decision support system for maintenance planning and mission assignment of a helicopter fleet, which partitions the master problem into two subproblems. The first subproblem is used to assign helicopters to inspections and to exercises, while the second one is used to assign helicopters to missions. The author develops two elastic mixed integer programs to formulate these two sub-problems and solves them using standard optimization software. Pippin (1998) develops a mixed integer linear program and a quadratic program to model the flight hour allocation problem. Both models try to find a flight hour allocation that ensures a steady-state sequence of aircraft into phase maintenance. The U.S. Department of the Army has released a Field Manual (US DoA, 2000), which describes the aircraft flowchart for scheduling periodic inspections and deciding which aircraft should fly in certain missions.

The Hellenic Air Force (HAF) and many other Air Force organizations worldwide solve the FMP problem empirically, utilizing in an ad-hoc manner a 2-dimensional graphical tool called the "aircraft flowchart" (US DoA, 2000). In current practice, the aircraft flowchart is at best used as a graphical device by the officer responsible for issuing the flight and maintenance plans. Kozanidis and Skipis (2006) introduce a biobjective model for flight and maintenance planning of military aircraft in order to achieve maximum fleet availability. Kozanidis (2009), introduces a multiobjective mixed integer linear model for maximizing fleet availability under the presence of flight and maintenance requirements, provides an application of this model on a real-life instance drawn from the HAF, and presents two

heuristic approaches that can be utilized for solving large instances of the problem. Kozanidis et al. (2010) extend that work, by developing a single objective optimization model that adopts one out of these objectives (wing aircraft availability) and incorporates the remaining ones with the introduction of associated constraints.

Chapter 3 PROBLEM DESCRIPTION

In this chapter, we provide some important information about the problem setting that we address in this work. This information is important because it justifies the mathematical developments that we present in the next chapters.

In this work, we study the optimization of a nonlinear mixed integer problem for flight and maintenance planning of mission aircraft. The problem is solved for a simple case with a single-period time horizon and a single squadron. Our aim is to achieve a balance between the remaining flight times of the available aircraft, and the remaining maintenance times of the grounded aircraft.

We consider a setting, in which the aircraft of a unit are partitioned into available ones which are able to participate in missions, and grounded ones which are currently receiving maintenance service. At the beginning of each particular time period, the wing command issues the flight requirements of the unit. These requirements (also called "flight load") denote the total time that the aircraft of the unit should fly during this time period. They are expressed as target values, from which only small deviations are permitted.

For each specific aircraft, we define its "residual flight time" as the total remaining time that the aircraft can fly until it has to undergo a maintenance check. The residual flight time of an aircraft is positive if and only if this aircraft is available to fly. Similarly, we define the "residual maintenance time" of an aircraft as the total remaining time that the aircraft needs in order to complete its maintenance check. The residual maintenance time of an

aircraft is positive if and only if this aircraft is undergoing a maintenance check and is therefore not available to fly.

For the maintenance needs of the unit, there exists a station, responsible for providing service to the aircraft of the unit. This station has certain space (also referred to as "dock space") and time capacity capabilities. Given the flight requirements, and the physical constraints that stem from the capacity of the maintenance station, the objective is to issue a flight and maintenance plan for each individual aircraft, so that some appropriate measure of effectiveness is optimized.

Consider a 2-dimensional graphical tool called the "aircraft flowchart". The vertical axis of this flowchart represents residual flight time measured in some appropriate unit, and the horizontal axis represents the indices of the available aircraft in non-decreasing order of their residual flight times, 1 being the index of the aircraft with the smallest and V being the index of the aircraft with the largest residual flight time, where V is the total number of available aircraft.

On the aircraft flowchart, consider the line segment connecting the origin with the point with coordinates (V, Y), where Y is the maximum time that an aircraft can fly between two consecutive maintenance checks, often referred to as "phase interval" in the related military literature. This line segment is also referred to as the "diagonal". By mapping the available aircraft of each squadron on the aircraft flowchart, we can visualize the total availability of the unit. Substituting the residual flight times of the available aircraft with the residual maintenance times of the grounded aircraft and parameter Y with parameter G (the total maintenance requirements of an aircraft grounded for service), we can get a similar graphical depiction of the maintenance requirements of the grounded aircraft of the unit.

To describe the smoothness of the distribution of the total residual flight time of the available aircraft, we use a "total deviation index". This index is equal to the sum of the vertical distances (deviations) of each point mapping an available aircraft from the diagonal. The smaller this sum is, the smoother the distribution of the total residual flight time. Ideally, the total deviation index is equal to zero, when all points lie on the diagonal. When issuing the individual aircraft flight plans, the intention is to keep each point as close to the diagonal as possible, in order to keep the total deviation index as small as possible.

A similar "total deviation index" can be used to describe the smoothness of the distribution of the total residual maintenance times of the grounded aircraft. In correspondence with the previous index, this index is equal to the sum of the vertical distances (deviations) of each point mapping a grounded aircraft from the diagonal. The smaller this sum is, the smoother the distribution of the total residual maintenance time. Ideally, the total deviation index is equal to zero, when all points lie on the diagonal. When issuing the individual aircraft maintenance plans, the intention is to keep each point as close to the diagonal as possible, in order to keep the total deviation index as small as possible. Our objective is to minimize the cumulative deviation index, which is equal to the sum of the two individual deviation indices.

In order to compute the optimal flight and maintenance times of the unit's aircraft, we use an analytical solution procedure which is introduced in Chapter 5. This approach utilizes two integer decision variables, one that denotes the number of available aircraft that will enter the station for service at the end of the current period, and one that denotes the number of grounded aircraft that will finish their service and exit the maintenance station.

Of course, these decision variables have trivial upper bounds. The number of available aircraft that will be grounded, for example, cannot be larger than the total number of available

aircraft, or larger than the station's space capacity. Similarly, the number of aircraft that will exit the maintenance station cannot be larger than the total number of grounded aircraft, or larger than the station's space capacity, too.

Tighter upper and lower bounds are computed at the beginning of the solution procedure for these decision variables, based on the remaining parameter values. For every feasible value-pair of these two variables, we solve two quadratic programming problems, one for the available aircraft and one for the grounded aircraft of the unit. Then, we add the optimal objective function values of these two sub-problems, in order to compute the optimal value of the cumulative deviation index for this particular pair of values of variables z_1 and z_2. Next, this value is compared to the best cumulative deviation index value that has been found so far. The significance of the proposed solution procedure stems from the fact that each quadratic programming sub-problem can be solved very easily, expediting the total computational effort. Additionally, the solution procedure is exact, ensuring this way that the global optimal solution will be found.

Chapter 4 MODEL DEVELOPMENT

In this section, we present the mathematical model that was developed for the problem under consideration. We use the following mathematical notation:

Decision Variables:

x_i : flight time of available aircraft i during the current period,

h_j : maintenance time of grounded aircraft j during the current period,

y_{in} : residual flight time of aircraft i at the beginning of the next time period,

g_{jn} : residual maintenance time of available aircraft j at the beginning of the next time period,

b_i : binary decision variable that takes the value 1 if available aircraft i enters the maintenance station at the beginning of the next time period, and 0 otherwise,

c_j : binary decision variable that takes the value 1 if grounded aircraft i exits the maintenance station at the beginning of the next time period, and 0 otherwise,

z_1 : number of aircraft that will enter the maintenance station at the beginning of the next time period,

z_2 : number of aircraft that will exit the maintenance station at the beginning of the next time period.

Parameters:

S: required flight load during the current period,

B : time capacity of the maintenance station during the current period,

y_{ip} : residual flight time of available aircraft i at the beginning of the current period,

g_{jp} : residual maintenance time of grounded aircraft j at the beginning of the current period,

X_{max}: maximum time an available aircraft can fly in a single time period,

Y_{min} : minimum residual flight time of an available aircraft,

G_{min} : minimum residual flight time of a non-available aircraft,

C : maximum number of aircraft that the maintenance station can accommodate,

G : residual maintenance time of an aircraft immediately after it enters the maintenance station,

Y : residual flight time of an aircraft immediately after it exits the maintenance station,

n_a : number of available aircraft during the current period,

$\overline{n_a}$: number of non-available aircraft during the current period,

L , U : real numbers denoting the maximum deviation from the value of S that can be tolerated,

N : total number of unit aircraft = $n_a + \overline{n_a}$.

Additional auxiliary notation:

$s_1 = \dfrac{Y}{(n_a - z_1 + z_2)}$: the slope of the diagonal in the flowchart of the available aircraft at the beginning of the next time period,

$s_2 = \dfrac{G}{(\overline{n_a} + z_1 - z_2)}$: the slope of the diagonal in the flowchart of the grounded aircraft at the beginning of the next time period.

Then, the problem under consideration can be formulated as follows:

$$\text{Min} \quad Z = \sum_{i=1}^{n_a} \left[(1-b_i) \cdot \left(y_{in} - (i - z_1) \cdot s_1 \right)^2 + b_i \cdot \left(G - (\overline{n_a} + i - z_2) \cdot s_2 \right)^2 \right]$$
$$\sum_{j=1}^{\overline{n_a}} \left[(1-c_j) \cdot \left(g_{jn} - (j - z_2) \cdot s_2 \right)^2 + c_j \cdot \left(Y - (n_a + j - z_1) \cdot s_1 \right)^2 \right]$$

s.t. $$y_{in} = y_{ip} - x_i, \quad i = 1, \ldots, n_a \qquad (1)$$

$$g_{jn} = g_{jp} - h_j, \quad j = 1, \ldots, \overline{n_a} \qquad (2)$$

$$z_1 = \sum_{i=1}^{n_a} b_i, \quad i = 1,...,n_a \tag{3}$$

$$z_2 = \sum_{j=1}^{\overline{n_a}} c_j, \quad j = 1,...,\overline{n_a} \tag{4}$$

$$L \cdot S \le \sum_{i=1}^{n_a} x_i \le U \cdot S \tag{5}$$

$$\sum_{j=1}^{\overline{n_a}} h_j = \min\left(B, \sum_{j=1}^{\overline{n_a}} g_{jp} \right), \tag{6}$$

$$\overline{n_a} + z_1 - z_2 \le C \tag{7}$$

$$y_{in} \ge \left(1 - b_i\right) \cdot Y_{\min}, \quad i = 1,...,n_a \tag{8}$$

$$g_{jn} \ge \left(1 - c_j\right) \cdot G_{\min}, \quad j = 1,...,\overline{n_a} \tag{9}$$

$$y_{in} \le \left(1 - b_i\right) \cdot y_{ip}, \quad i = 1,...,n_a \tag{10}$$

$$g_{jn} \le \left(1 - c_j\right) \cdot g_{jp}, \quad j = 1,...,\overline{n_a} \tag{11}$$

$$x_i \le X_{\max}, \quad i = 1,...,n_a \tag{12}$$

$$x_i \le y_{ip}, \quad i = 1,...,n_a \tag{13}$$

$$h_j \le g_{jp}, \quad j = 1,...,\overline{n_a} \tag{14}$$

$$z_1 \le n_a \tag{15}$$

$$z_2 \le \overline{n_a} \tag{16}$$

$$x_i, y_{in} \ge 0, \quad i = 1,...,n_a \tag{17}$$

$$h_j, g_{jn} \ge 0, \quad j = 1,...,\overline{n_a} \tag{18}$$

$$b_i \ \textit{binary}, \quad i = 1,...,n_a \tag{19}$$

$$c_j \ \textit{binary}, \quad j = 1,...,\overline{n_a} \tag{20}$$

$$z_1 , z_2 \in Z_+ \qquad (21)$$

The objective function minimizes the cumulative deviation index, which is equal to the sum of squares of all deviations of the residual (flight and maintenance) times from their diagonal target values. More specifically, the first summation is associated with the available aircraft of the unit and consists of two terms, the first one referring to those that will retain availability and the second one referring to those that will be grounded. The second summation is associated with the grounded aircraft of the unit and consists of two terms, the first one referring to those that will remain grounded and the second one referring to those that will become available.

The first set of constraints is used to update the residual flight time of each available aircraft at the beginning the next period, based on its residual flight time at the beginning of the current period and the time that it will fly during this period. The second set of constraints is used to update the residual maintenance time of each grounded aircraft at the beginning of the next period, based on its residual maintenance time at the beginning of the current period and the time that it will receive maintenance during this period.

Constraint sets (3) and (4) are used to compute the number of aircraft that will enter and exit the maintenance station, respectively. These computations utilize binary variables b_i and c_j, for which the following hold:

$$b_i = \begin{Bmatrix} 1, & if\ i \le z_1 \\ 0, & otherwise \end{Bmatrix}, \quad i = 1,\ldots,n_a$$

$$c_j = \begin{Bmatrix} 1, & if\ j \le z_2 \\ 0, & otherwise \end{Bmatrix}, \quad j = 1,\ldots,\overline{n_a}$$

If b_i is equal to 1, then available aircraft i will be grounded at the beginning of the next period. Similarly, if c_j is equal to 1, grounded aircraft j will become available at the beginning of the next period.

Constraint set (5) ensures that the flight requirements are met. Variables L and U define an interval $[LS, US]$, in which the actual flight time of the available aircraft should lie. For example, when $L = 0.95$ and $U = 1.05$ a maximum of 5% deviation from the flight requirements is permitted.

Constraint (6) is introduced to ensure that the maintenance crew will not idle whenever there is at least one aircraft waiting for service. More specifically, this constraint ensures that the total maintenance time provided by the station will either be equal to the total time capacity of the station during this period, or to the total maintenance requirements of this period, whichever of these two is smaller. Constraint (7) ensures that the space capacity of the maintenance station is not violated.

Constraint set (8) imposes a lower bound on the residual flight time of each available aircraft, and constraint set (9) imposes a lower bound on the residual maintenance time of each non-available aircraft. These constraints are introduced to eliminate the situation in which an aircraft has negligible but positive residual flight or maintenance time. Constraint set (10) states that the residual flight time of an available aircraft in the next time period cannot exceed that of the current time period, and ensures that it will be zero whenever this aircraft is grounded. Similarly, constraint set (11) states that the residual maintenance time of a grounded aircraft in the next time period cannot exceed that of the current time period, and ensures that it will be zero whenever this aircraft becomes available.

Constraint set (12) imposes an upper bound on the maximum time that an available aircraft can fly during a single time period. Such a restriction is usually present due to technical reasons. Constraint set (13) ensures that the total time that an available aircraft will fly during the current time period does not exceed its residual flight time at the beginning of the same period. Similarly, constraint set (14) ensures that the total time that the maintenance

crew will work on a particular aircraft during the current time period will not exceed the residual maintenance time of this aircraft at the beginning of the same period. Constraints (15) and (16) impose trivial upper bounds on decision variables z_1 and z_2. Constraints (17)-(18) and (19)-(21) are the non-negativity and integrality constraints, respectively.

The above formulation utilizes implicitly a fundamental property of the problem, i.e., that the order of available and grounded aircraft in the optimal solution can always remain unchanged. In other words, the available aircraft can always be grounded in non-decreasing order of their residual flight times, and the grounded aircraft can always finish their service in non-decreasing order of their residual maintenance times. The validity of this property becomes immediately clear with the following line of reasoning. If this is not the case at the optimal solution of the problem, then we can directly transfer flight (or maintenance) time from an aircraft that disrupted this order to the aircraft that was originally next in line to be grounded (or finish service). This action retains feasibility, and additionally, the optimal solution remains the same.

Chapter 5 SOLUTION ALGORITHM

5.1 Solution Methodology

In this chapter, we develop our solution methodology. We start by developing some theoretical background, and based on that, we develop an exact solution algorithm that returns the problem's global optimal solution. We also document in detail the proposed algorithm, providing its various steps in pseudo-code.

Initially, the available aircraft of the unit are arranged in a non-decreasing order of their residual flight times. Thus, the available aircraft with the lowest residual flight time appears first in this arrangement, assuming an index of 1 (i=1) and the available aircraft with the highest residual flight time appears last in the arrangement, assuming an index of n_a (i=n_a), where n_a is the total number of the unit's available aircraft. The same procedure is also followed for the grounded aircraft; they are arranged in non-decreasing order of their residual maintenance times with indices between j=1 and j=$\overline{n_a}$, where $\overline{n_a}$ is the total number of the unit's grounded aircraft.

The next step is to determine the feasible values for z_1 and z_2. As already mentioned, z_1 is the number of available aircraft that will enter the station at the beginning of the next time period and z_2 is the number of grounded aircraft that will exit the maintenance station at the beginning of the next time period. Several checks are performed in order to eliminate infeasible values for these decision variables. Initially, z_1 can take integer values in the

interval [0, min(C,n_a)] and z_2 can take integer values in the interval [0,$\overline{n_a}$]. Then, a search procedure is applied, in order to find the feasible value-pairs of variables z_1 and z_2.

Let *Sum* be an auxiliary variable. In order to find feasible values for z_1, we set *Sum* = 0, and then we start adding to it the residual flight time of an aircraft, if it is less or equal to X_{max}, always in the order that the aircraft appear in the corresponding arrangement. As long as *Sum* remains less or equal to *US*, the index of the aircraft that was considered last is a feasible value for z_1. The procedure stops either when Sum exceeds *US*, or when the residual flight time of the next aircraft is higher than X_{max}.

A lower bound for z_1 is obtained using a similar procedure. Initially, we set again *Sum* = 0. Then, we add to it, the minimum between (y_{ip}-Y_{min}) and X_{max}, for all available aircraft *i*, in the reverse order of the corresponding arrangement. If this summation becomes greater or equal to *LS*, then the minimum feasible value for z_1 is 0. If not and the complete list of aircraft is scanned, then the minimum feasible value for z_1 greater than 0. In this case, we can divide the grounded aircraft into two categories: those that can be grounded without exceeding the maximum flight load, X_{max}, and those that cannot. Thus, we allocate the extra flight load needed to reach *LS* to the former aircraft, until this summation becomes greater or equal to *LS*. The minimum feasible value for z_1 will be equal to the number of aircraft that will be grounded until this condition is satisfied.

As far as the bounds for z_2 are concerned, their computation is based on the fact that the station works continuously in any time period, until either its time capacity is fully utilized, or the service of all grounded aircraft is completed. Thus, if the total maintenance load of all grounded aircraft is less or equal to the station's time capacity, then all grounded aircraft can become available and the maximum feasible value for z_2 is $\overline{n_a}$. Otherwise, we set

again $Sum = 0$, and we start adding to it the residual flight times of all grounded aircraft, in the order that they appear in the corresponding arrangement. As long as this summation remains less or equal to B, the index of the last aircraft that was considered is a feasible value for z_2. The procedure stops when Sum exceeds B.

In order to obtain a lower bound for z_2, we set again $Sum = 0$. Then, we add to it, the quantity $(g_{jp}-G_{min})$, for all grounded aircraft j, in the reverse order of the corresponding arrangement. If this summation becomes greater or equal to B, then the minimum feasible value for z_2 is 0. If not and the complete list of aircraft is scanned, then we continue adding to Sum the quantity G_{min} for each grounded aircraft, until this summation becomes greater or equal to B. The number of times that this quantity needs to be added before this condition is satisfied is the minimum feasible value for z_2. The last step after the bounds for z_1 and z_2 have been obtained is to check which value-pairs are feasible with respect to constraint (7).

The proposed methodology is based on the fact that for a particular value-pair of variables z_1 and z_2, the problem is split into two sub-problems that can be solved independently rather easily. The first of these sub-problems involves decisions related to the available aircraft, and the second one involves decisions related to the grounded aircraft. More specifically, since we know the values of z_1 and z_2, we know which available aircraft will be grounded and which grounded aircraft will finish their service at the beginning of the next time period. Thus, the indices of the aircraft are recomputed as follows.

The index of an available aircraft i becomes:

$$\begin{Bmatrix} i-z_1, & \text{if } i > z_1 \\ \overline{n_a}+i-z_2, & \text{if } i \le z_1 \end{Bmatrix}$$

The index of a grounded aircraft j becomes:

$$\left\{ \begin{array}{ll} j-z_2, & \text{if } j > z_2 \\ n_a+j-z_1, & \text{if } j \le z_2 \end{array} \right\}$$

Note additionally, that when z_1 and z_2 are known, variables b_i and c_j are determined, as well. Thus, the mathematical model becomes simpler, because we have two smaller sub-problems that can be solved separately.

The first sub-problem that involves decisions related to the available aircraft is:

$$\text{Min} \quad Z_1 = \sum_{i=z_1+1}^{n_a} \left[y_{in} - (i-z_1)s_1 \right]^2 + \sum_{i=n_a-z_1+1}^{n_a-z_1+z_2} \left[Y - is_1 \right]^2$$

$$\text{s.t.} \quad y_{in} = y_{ip} - x_i \,, \quad i = (z_1+1),...,(n_a)$$

$$L \cdot S - \sum_{i=1}^{z_1} y_{ip} \le \sum_{i=z_1+1}^{n_a} x_i \le U \cdot S - \sum_{i=1}^{z_1} y_{ip} \,,$$

$$y_{in} \ge Y_{\min} \,, \quad i = (z_1+1),...,n_a$$

$$x_i \le X_{\max} \,, \quad i = (z_1+1),...,n_a$$

$$x_i, y_{in} \ge 0 \,, \quad i = (z_1+1),...,n_a$$

The second sub-problem that involves decisions related to the grounded aircraft is:

$$\text{Min} \quad Z_2 = \sum_{j=z_2+1}^{\overline{n_a}} \left[g_{jn} - (j-z_2)\cdot s_2 \right]^2 + \sum_{j=\overline{n_a}-z_2+1}^{\overline{n_a}-z_2+z_1} \left[G - j \cdot s_2 \right]^2$$

$$\text{s.t.} \quad g_{jn} = g_{jp} - h_j, \quad j = z_2+1,...,\overline{n_a}$$

$$\sum_{j=1}^{z_2} g_{jp} + \sum_{j=z_2+1}^{\overline{n_a}} h_j = \min\left(B, \sum_{j=1}^{\overline{n_a}} g_j \right),$$

$$g_{jn} \ge G_{\min} \,, \quad j = z_2+1,...,\overline{n_a}$$

$$h_j, g_{jn} \geq 0, \quad j = z_2 + 1, \ldots, \overline{n_a}$$

Each of these two nonlinear problems can be solved independently of the other, using an efficient procedure called "Sweep" (see Gavranis, 2007) that we describe briefly next. From the optimal objective function values of these two sub-problems, we obtain the cumulative deviation index value ($Z = Z_1 + Z_2$) of the original problem for this particular value-pair of variables z_1 and z_2. The main idea of the proposed algorithm is to compare the optimal cumulative deviation index value of the original problem for all feasible values of z_1 and z_2, and choose the best out of them.

5.2 Sweep Algorithm

The two sub-problems introduced above are quadratic programming problems. The Hessian of their objective function is diagonal with all diagonal elements equal to 2; therefore, their objective function is convex. Hence, the KKT conditions (see Bazaraa et al., 2006) are necessary and sufficient for optimality. We introduce next a procedure called "Sweep" that can be utilized to obtain their optimal solution.

Consider the flowchart that maps the available aircraft. An aircraft that will exit the maintenance station at the beginning of the next time period is considered to have residual flight time equal to Y at the beginning of the current period in the aircraft arrangement, but its flight time is also restricted to 0-value (since this aircraft will be grounded during the current time period). On the other hand, an aircraft that will enter the maintenance station at the beginning of the next time period is not portrayed on this graph, since its flight time will be

equal to its residual flight time, which implies that this aircraft will affect the total deviation index that refers to the grounded aircraft of the unit.

On this flowchart, consider a line parallel to the diagonal which is initially placed far enough to the top, so that all the aircraft lie below it, as shown in Figure 5-1 (in what follows, we do not distinguish between a point on the graph and the aircraft that this point maps). Assume now that this line starts moving towards the diagonal (and past it, while always remaining parallel to it), sweeping along *vertically* each aircraft that it comes across. Throughout this move, flight times are accordingly assigned to the aircraft in the order that they are swept by the line. If during this procedure the flight time of an aircraft i reaches its maximum possible value, $X_{ui} = \min(X_{max}, y_{ip}-Y_{min})$, then the line should "disengage" this aircraft and continue its move without sweeping it further, to ensure that the resulting solution will remain feasible.

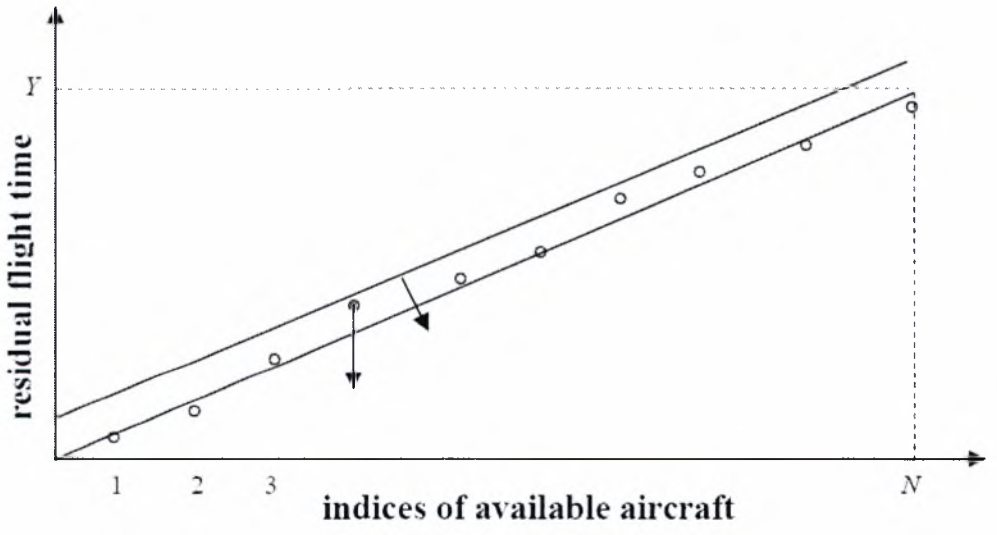

Figure 5-1: Illustration of the Procedure "Sweep"

Consider now the following 4 solutions that can be obtained during the application of this procedure:

1. The solution in which the sum of the assigned aircraft flight times is equal to *LS*.

2. The solution in which the sum of the assigned aircraft flight times is equal to *US*.

3. The solution in which each aircraft, i, is assigned its maximum possible flight time, X_{ui}. In what follows, we refer with "*X*" to the sum of the assigned aircraft flight times of this solution.

4. The solution in which the sweeping line coincides with the diagonal. In what follows, we refer with "*D*" to the sum of the assigned aircraft flight times of this solution.

The following is a very crucial and interesting result, utilized in the development of our proposed methodology:

Proposition 1. If the quantities *LS*, *US*, *X* and *D* are placed in non-decreasing order, then:

a) If, after taking into consideration any ties present, there does not exist an arrangement in which *LS* precedes *X*, then the problem is infeasible.

b) If an arrangement in which *LS* precedes *X* exists, then the optimal solution of the problem is the one obtained by Procedure Sweep when the sum of the assigned aircraft flight times becomes equal to the quantity that appears second in this arrangement.

Proof: See Kozanidis et al., 2008.

The application of Procedure Sweep produces the flight time of each available aircraft. The same procedure can also be applied for the production of the maintenance time of each non-available aircraft. The only differences are that, in that case, $L = U$ (since the total maintenance load carried out must be exactly equal to the minimum between B and $\sum_{j=1}^{\overline{n}_0} g_j$), and $X_{uj} = (g_{jp}\text{-}G_{min})$, since there does not exist an upper bound on the maintenance time of

each grounded aircraft. Therefore, the optimal solution in this case is obtained by Procedure Sweep when the sum of the assigned aircraft maintenance times becomes equal to B.

5.3 Solution Algorithm

Based on the above discussion, the detailed steps of our solution methodology are introduced next. The following additional notation is used in the pseudo-code:

Sx = total flight time in current time period

Sy = total residual flight time in current time period

Sh = total maintenance time in current time period

Sg = total residual maintenance time in current time period

C_{res} = residual maintenance space capacity

C = maintenance space capacity

Z = cumulative deviation index value of the original problem

M = sufficiently large number

N = total number of aircraft of the unit

Step 0: initialization

$n_a = 0$, $\overline{n_a} = 0$, $Sy = 0$, $Sg = 0$, $C_{res} = C$, $Z = M$

for $n = 1$ to N do

$y_{np} = Y1n$, $Sy = Sy + y_{np}$

if $y_{np} > 0 \longrightarrow n_a = n_a + 1$

$g_{np} = G1n$, $Sg = Sg + g_{np}$

if $g_{np} > 0 \longrightarrow \overline{n_a} = \overline{n_a} + 1$, $C_{res} = C_{res} - 1$

end for

arrange in non-decreasing order of y_{np} the available aircraft and determine their indices i

arrange in non-decreasing order of g_{np} the grounded aircraft and determine their indices j

Step 1: bounds for feasible values of z_1 and z_2

find feasible integer values for variables z_1 and z_2

for every feasible pair of (z_1, z_2)

Step 2: initialization of sub-problems

$s_1 = Y / (n_a - z_1 + z_2)$

$s_2 = G / (\overline{n_a} + z_1 - z_2)$

for $i = 1$ to z_1

$X_{u(\overline{n_a} - z_2 + i)} = 0$

$x_i = y_{ip}$, $y_{in} = 0$, $g_{in} = G$

end for

for $j = 1$ to z_2

$X_{u(n_a - z_1 + j)} = 0$

$h_j = g_{jp}$, $g_{jn} = 0$, $y_{jn} = Y$

end for

Step 3: development of flight and maintenance plans

using Procedure Sweep, issue the aircraft flight and maintenance plans for the current time period

Step 4: evaluation of solution

Compute Z_{cur} = objective function value of original problem

if $Z_{cur} < Z \longrightarrow Z = Z_{cur}$, keep current solution as best so far

end for

Print best solution

Chapter 6 COMPUTATIONAL IMPLEMENTATION

In this chapter, we analyze the computational complexity of the proposed algorithm and we present the computational results of the experiments that we conducted after this algorithm was implemented in C programming language. The full implementation code is included in Appendix A.

6.1 Computational Complexity

As already mentioned, when we fix particular values for variables z_1 and z_2, the problem that we address is split into two sub-problems, each of which can be solved to optimality with Algorithm Sweep. Kozanidis et al. (2008) have proven that the worst-case computational complexity of Algorithm Sweep is O(N), where N is the total number of decision variables. Therefore, the worst-case computational complexity for solving *once* both these problems is also O(N), where N is the total number of aircraft (both available and grounded). The worst-case complexity for finding the feasible values for z_1 and z_2 is O(N) + O(C^2), since the procedure performs first a scanning of the aircraft lists a finite number of times and then performs a maximum of $(C+1)^2$ checks in order to eliminate feasible value-pairs.

Of course, the worst-case computational complexity for solving the original problem depends on the total number of times that these sub-problems must be solved, which is equal to the total number of feasible value-pairs of the variables z_1 and z_2. This number is not known

in advance, but cannot be larger than $(C+1)^2$, since the number of aircraft that will exit or enter the maintenance station cannot be larger than C. Therefore, the worst-case computational complexity for solving the original problem is $O(N) + O(C^2) + O(NC^2) = O(NC^2)$.

6.2 Computational Results

Our computational experiments were performed on a Dual Xeon server with a 2 GHz processor and 2 GB system memory. We used 5 different values for the unit's total number of aircraft (N = 500, 1000, 1500, 2000 and 2500), and solved 10 random problem instances for each of them. The procedure for generating these random instances was the following: Parameter C was set equal to $0.2N$, rounded up to the nearest integer. The number of grounded aircraft was generated randomly, using a uniform discrete probability function that considered the integer values between $0.15N$ and $0.2N$, inclusive. Of course, the number of available aircraft was equal to N minus the number of grounded aircraft.

The residual flight time of each available aircraft was a random number distributed uniformly in the interval $[0,Y]$ and the residual maintenance time of each grounded aircraft was a random number distributed uniformly in the interval $[0,G]$. Parameter B was set equal to $0.8\sum_j g_{jp}$, where index j runs over all grounded aircraft. Parameter S was set equal to $0.75\sum_i \min(y_{ip}, X_{max})$, where index i runs over all available aircraft. Actual values drawn from real applications were used for the other problem parameters, i.e., L = 0.95, U = 1.05, $X_{max} = 50$, $Y_{min} = 0.1$ and $G_{min} = 0.1$.

Table 6.1 presents the results of our experiments. More specifically, rows 2-4 of this table show the algorithm's maximum, average and minimum computational times,

respectively, over the 10 instances. Rows 5-7 of the same table show the maximum, average and minimum feasible value-pairs of variables z_1 and z_2, respectively.

Number of aircraft (N)	**500**	**1000**	**1500**	**2000**	**2500**
Max Run Time (seconds)	4.06	50.23	223.43	702.82	1656.23
Average Run Time (seconds)	3.601	46.837	203.836	623.419	1454.049
Min Run Time (seconds)	3.16	44.02	185.02	561.71	1279.59
Average Number of feasible pairs (z_1, z_2)	3822	15223	33688	60626	95157
Max Number of feasible pairs (z_1, z_2)	4031	15821	34390	62865	98666
Min Number of feasible pairs (z_1, z_2)	3481	14460	32115	59508	92073

Table 6-1: Computational results for different values of N

As seen in Table 6.1, as the total number of aircraft increases, the average computational time increases, too. For N = 2500, the average time is about 25 minutes. Considering the large number of feasible value-pairs of variables z_1 and z_2, it becomes clear that the actual computational effort required for the solution of the two sub-problems (when variables z_1 and z_2 have been fixed to particular values) is practically negligible. In any case, the worst-case computational complexity of $O(NC^2)$ reveals that the increase of the

computational effort is polynomial and not exponential. Additionally, it is possible that the proposed C implementation code can be further improved through appropriate enhancements.

6.3 Numerical examples

In this subsection, we illustrate the application of the proposed algorithm through two small numerical examples.

6.3.1 Numerical example 1

Parameter values:

$S = 125$

$B = 350$

$X_{max} = 50$

$Y_{min} = G_{min} = 0.1$

$L = 0.95$, $U = 1.05$

$Y = 300$

$G = 320$

$C = 3$

$N = 6$

i: index for available aircraft

j: index for grounded aircraft

N	1	2	3	4	5	6
y_{ip}	0	0	50	298	38	273
g_{jp}	300	130	0	0	0	0

Table 6-2: First problem's initial flight and maintenance times

Results:

LS = 118.75, US = 131.25

$n_a = 4$, $\overline{n_a} = 2$

Non-decreasing order:

y_{ip}	38	50	273	298	-	-
i	1	2	3	4	-	-
g_{jp}	130	300	-	-	-	-
j	1	2	-	-	-	-

Table 6-3: Sorting in non-decreasing order

Initially, z_1 can take integer values in the interval $[0, \min(C,n_a)] = [0,3]$.

Feasibility check 1:

$Sum = 0$, $Sum = Sum + 38 = 38 < 131.25$, $Sum = Sum + 50 = 88 < 131.25$. Since the residual flight time of the third aircraft is strictly larger than X_{max}, z_1 cannot be larger than 2.

Feasibility check 2:

$Sum = 0$, $Sum = Sum + \min\left(y_{4p} - Y_{\min}, X_{\max}\right) = 0 + 50 = 50 < 118.75$, $Sum = Sum + \min\left(y_{3p} - Y_{\min}, X_{\max}\right) = 50 + 50 = 100 < 118.75$, $Sum = Sum + \min\left(y_{2p} - Y_{\min}, X_{\max}\right) = 100 + 49.9 = 149.9 > 118.75$. Therefore, $z_1 = 0$ is feasible.

Combined with the previous check, z_1 can take integer values in the interval [0,2].

Initially, z_2 can take integer values in the interval $[0, \overline{n_a}] = [0,2]$.

Feasibility check 3:

$Sum = 0$, $Sum = Sum + 130 = 130 < 350$, $Sum = Sum + 300 = 430 > 350$. Therefore, $z_2 = 2$ is not a feasible value. Thus, z_2 can take integer values in the interval [0, 1].

Feasibility check 4:

$Sum = 0$, $Sum = Sum + \left(g_{2p} - G_{\min}\right) = 0 + 299.9 = 299.9 < 350$, $Sum = Sum + \left(g_{1p} - G_{\min}\right) = 299.9 + 129.9 = 429.8 > 350$. Therefore, $z_2 = 0$ is feasible. Combined with the previous check, z_2 can take integer values in the interval [0, 1].

Feasibility check 5:

$(z_1, z_2) = (0, 0)$

$\overline{n_a} + z_1 - z_2 \leq C \longrightarrow 2 + 0 - 0 = 2 < 3$, OK

$(z_1, z_2) = (0, 1)$

$\overline{n_a} + z_1 - z_2 \leq C \longrightarrow 2 + 0 - 1 = 1 < 3$, OK

$(z_1, z_2) = (1, 0)$

$\overline{n_a} + z_1 - z_2 \leq C \longrightarrow 2 + 1 - 0 = 3 = 3$, OK

$(z_1, z_2) = (1, 1)$

$\overline{n_a} + z_1 - z_2 \leq C \longrightarrow 2 + 1 - 1 = 2 < 3$, OK

$(z_1, z_2) = (2, 0)$

$\overline{n_a} + z_1 - z_2 \leq C \longrightarrow 2 + 2 - 0 = 4 > 3$ infeasible

$(z_1, z_2) = (2, 1)$

$\overline{n_a} + z_1 - z_2 \leq C \longrightarrow 2 + 2 - 1 = 3 = 3$, OK

Therefore, the feasible pairs are (0,0), (0,1), (1,0), (1,1), (2,1).

Iteration 1:

$(z_1, z_2) = (0, 0)$

Available

$s = 300 / 4 - 0 + 0 = 75$

$y_5 = 38$ $s = 75$ $i = 1$

$y_3 = 50$ $2s = 150$ $i = 2$

$y_6 = 273$ $3s = 225$ $i = 3$

$y_4 = 298$ $4s = 300$ $i = 4$

Grounded

$s = 320 / 2 + 0 - 0 = 160$

$g_2 = 130$ $s = 162$ $j = 1$

$g_1 = 300$ $2s = 320$ $j = 2$

After application of Algorithm Sweep, we get the following results:

i,j	1	2	3	4	5	6
y_{in}	-	-	50	248	19.25	223
x_i	-	-	0	50	18.75	50
g_{jn}	104.9	0.1	-	-	-	-
h_j	195.1	129.9	-	-	-	-

Table 6-4: Results of first iteration of the first problem

The value of the objective function is: Z =87652.0825

Iteration 2:

$(z_1, z_2) = (0, 1)$

Available

$s = 300 / 4 - 0 + 1 = 60$

$y_5 = 38$ $s = 60$ $i = 1$

$y_3 = 50$ $2s = 120$ $i = 2$

$y_6 = 273$ $3s = 180$ $i = 3$

$y_4 = 298$ $4s = 240$ $i = 4$

$y_2 = 300$ $5s = 300$ $i = 5$ ($x_2 = -$, $h_2 = g_2$)

Grounded

$s = 320 / 2 + 0 - 1 = 320$

$g_1 = 300$ $s = 320$ $j = 1$

After application of Algorithm Sweep, we get the following results:

i,j	1	2	3	4	5	6
y_{in}	-	300	50	248	19.25	223
x_i	-	-	0	50	18.75	50
g_{jn}	105	-	-	-	-	-
h_j	195	130	-	-	-	-

Table 6-5: Results of second iteration of the first problem

The value of the objective function is: Z =54698.56

Iteration 3:

$(z_1, z_2) = (1, 0)$

Available

$s = 300 / 4 - 1 + 0 = 100$

$y_3 = 50$ $s = 100$ $i = 1$

$y_6 = 273$ $2s = 200$ $i = 2$

$y_4 = 298$ $3s = 300$ $i = 3$

Grounded

$s = 320 / 2 + 1 - 0 = 106.6$

$g_2 = 130$ $s = 106.6$ $j = 1$

$g_1 = 300$ $2s = 213.3$ $j = 2$

$g_5 = 320$ $3s = 320$ $j = 3$ ($h_2 = -$, $x_2 = y_2$)

After application of Algorithm Sweep, we get the following results:

i,j	1	2	3	4	5	6
y_{in}	-	-	50	267.25	-	223
x_i	-	-	0	30.75	38	50
g_{jn}	104.9	0.1	-	-	320	-
h_j	195.1	129.9	-	-	-	-

Table 6-6: Results of third iteration of the first problem

The value of the objective function is: Z =27215.805

Iteration 4:

$(z_1, z_2) = (1, 1)$

Available

$s = 300 / 4 - 1 + 1 = 75$

$y_3 = 50 \quad s = 75 \quad i = 1$

$y_6 = 273 \quad 2s = 150 \quad i = 2$

$y_4 = 298 \quad 3s = 225 \quad i = 3$

$y_2 = 298 \quad 4s = 300 \quad i = 4$

Grounded

$s = 320 / 2 + 1 - 1 = 160$

$g_1 = 300 \quad s = 160 \quad j = 1$

$g_5 = 320 \quad 2s = 320 \quad j = 2$

After application of Algorithm Sweep, we get the following results:

i,j	1	2	3	4	5	6
y_{in}	-	300	50	254.75	-	223
x_i	-	-	0	43.25	38	50
g_{jn}	105	-	-	-	-	-
h_j	195	130	-	-	-	-

Table 6-7: Results of forth iteration of the first problem

The value of the objective function is: Z = 9864.0625

Iteration 5:

$(z_1, z_2) = (2, 1)$

Available

$s = 300 / 4 - 2 + 1 = 100$

$y_6 = 273 \quad s = 100 \quad i = 1$

$y_4 = 298 \quad 2s = 200 \quad i = 2$

$y_2 = 300 \quad 3s = 300 \quad i = 3$

Grounded

$s = 320 / 2 + 2 - 1 = 106.6$

$g_2 = 300 \quad s = 106.6 \quad j = 1$

$g_5 = 320 \quad 2s = 213.3 \quad j = 2$

$g_3 = 320 \quad 3s = 320 \quad j = 3$

After application of Algorithm Sweep, we get the following results:

i,j	1	2	3	4	5	6
y_{in}	-	300	-	298	-	229.75
x_i	-	-	50	0	38	43.25
g_{jn}	105	-	320	-	320	-
h_j	195	130	-	-	-	-

Table 6-8: Results of fifth iteration of the first problem

The value of the objective function is: Z = 37819.6185

After the solution of all feasible pairs, we compare the values of Z, to find the pair that has the minimum Z. For this problem, the optimum solution is given for the pair $(z_1, z_2) = (1, 1)$.

6.3.2 Numerical example 2

Parameter values:

$S = 150$

$B = 425$

$X_{max} = 50$

$Y_{min} = G_{min} = 0.1$

$L = 0.95$, $U = 1.05$

$Y = 300$

$G = 320$

$C = 4$

$N = 8$

i: index for available aircraft

j: index for grounded aircraft

N	1	2	3	4	5	6	7	8
y_{ip}	175	30	48	105	79	130	0	0
g_{jp}	0	0	0	0	0	0	320	105

Table 6-9: Second problem's initial flight and maintenance times

Results:

LS = 142.5, US = 157.5

$n_a = 6$, $\overline{n_a} = 2$

Non-decreasing order:

y_{ip}	30	48	79	105	130	175	-	-
i	1	2	3	4	5	6	-	-
g_{jp}	105	320	-	-	-	-	-	-
j	1	2	-	-	-	-	-	-

Table 6-10: Sorting in non-decreasing order

Initially, z_1 can take integer values in the interval $[0, \min(C, n_a)] = [0, 4]$.

Feasibility check 1:

$Sum = 0$, $Sum = Sum + 30 = 30 < 157.5$, $Sum = Sum + 48 = 78 < 157.5$. Since the residual flight time of the third aircraft is strictly larger than X_{max}, z_1 cannot be larger than 2.

Feasibility check 2:

$\sum_i \min(y_i - Y_{\min}, X_{\max}) \le LS$: if this is valid, then z_1=0 infeasible

$29.9 + 47.9 + 50 + 50 + 50 + 50 = 277.8 > LS$

Therefore, the case that none of the aircraft will become non-available at the start of the next period is feasible. As result, and considering check 2, z1 is feasible in [0, 2].

$Sum = 0$, $Sum = Sum + \min(y_{6p} - Y_{\min}, X_{\max}) = 0 + 50 = 50 < 142.5$, $Sum = Sum + \min(y_{5p} - Y_{\min}, X_{\max}) = 50 + 50 = 100 < 142.5$, $Sum = Sum + \min(y_{4p} - Y_{\min}, X_{\max}) = 100 + 50 = 150 > 142.5$, $Sum = Sum + \min(y_{3p} - Y_{\min}, X_{\max}) = 150 + 50 = 200 > 142.5$, $Sum = Sum + \min(y_{2p} - Y_{\min}, X_{\max}) = 200 + 47.9 = 247.9 > 142.5$, $Sum = Sum + \min(y_{1p} - Y_{\min}, X_{\max}) = 247.9 + 29.9 = 277.8 > 142.5$. Therefore, $z_1 = 0$ is feasible.

Combined with the previous check, z_1 can take integer values in the interval [0,2].

Initially, z_2 can take integer values in the interval $[0, \overline{n_a}] = [0,2]$

Feasibility check 3:

$Sum = 0$, $Sum = Sum + 105 = 105 < 425$, $Sum = Sum + 320 = 425 = 425$. Therefore, $z_2 = 2$ is the only feasible value. Thus, z_2 can take only the value 2.

Feasibility check 4:

$Sum = 0$, $Sum = Sum + \left(g_{2p} - G_{\min}\right) = 0 + 319.9 = 319.9 < 425$, $Sum = Sum + \left(g_{1p} - G_{\min}\right) = 319.9 + 104.9 = 424.8 < 425$. Therefore, $z_2 = 0$ is infeasible, and $z_2=1$ as well. Combined with the previous check, z_2 can only take the value 2

Feasibility check 5:

$(z_1, z_2) = (0,2)$

$\overline{n_a} + z_1 - z_2 \le C \longrightarrow 2 + 0 - 2 = 0 < 4$, OK

$(z_1, z_2) = (1,2)$

$\overline{n_a} + z_1 - z_2 \leq C \longrightarrow 2 + 1 - 2 = 1 < 4$, OK

$(z_1, z_2) = (2,2)$

$\overline{n_a} + z_1 - z_2 \leq C \longrightarrow 2 + 2 - 2 = 2 < 4$, OK

Therefore, the feasible pairs are (0,2), (1,2), (2,2).

Iteration 1:

$(z_1, z_2) = (0, 2)$

Available

$s = 300 / 6 - 0 + 2 = 37.5$

$y_2 = 30$ $s = 37.5$ $i = 1$

$y_3 = 48$ $2s = 75$ $i = 2$

$y_5 = 79$ $3s = 112.5$ $i = 3$

$y_4 = 105$ $4s = 150$ $i = 4$

$y_6 = 130$ $5s = 187.5$ $i = 5$

$y_1 = 175$ $6s = 225$ $i = 6$

$y_8 = 300$ $7s = 262.5$ $i = 7$

$y_7 = 300$ $8s = 300$ $i = 8$

Grounded

There is none grounded aircraft

After application of Algorithm Sweep, we get the following results:

i,j	1	2	3	4	5	6	7	8
y_{in}	159.88	0.1	9.88	84.88	47.38	122.38	300	300
x_i	15.12	29.9	38.12	20.12	31.62	7.62	-	-
g_{jn}	-	-	-	-	-	-	-	-
h_{jn}	-	-	-	-	-	-	320	105

Table 6-11: Results of first iteration of the second problem

The value of the objective function is: Z = 24008.082

Iteration 2:

$(z_1, z_2) = (1, 2)$

Available

s = 300 / 6 - 1 + 2 = 42.86

$y_3 = 48$ s = 42.86 i = 1

$y_5 = 79$ 2s = 85.72 i = 2

$y_4 = 105$ 3s = 128.58 i = 3

$y_6 = 130$ 4s = 171.44 i = 4

$y_1 = 175$ 5s = 214.3 i = 5

$y_8 = 300$ 6s = 257.16 i = 6

$y_7 = 300$ 7s = 300 i = 7

Grounded

s = 320 / 2 + 1 - 2 = 320

$g_2 = 320$ s = 320 j = 1

After application of Algorithm Sweep, we get the following results:

i,j	1	2	3	4	5	6	7	8
y_{in}	170.4	0	0.1	84.7	41.8	127.5	300	300
x_i	4.6	30	47.9	20.3	37.2	2.5	-	-
g_{jn}	-	320	-	-	-	-	-	-
h_j	-	-	-	-	-	-	320	105

Table 6-12: Results of second iteration of the second problem

The value of the objective function is: Z = 11373.747959

Iteration 3:

$(z_1, z_2) = (2, 2)$

Available

s = 300 / 6 - 2 + 2 = 50

$y_5 = 79$ s = 50 i = 1

$y_4 = 105$ 2s = 100 i = 2

$y_6 = 130$ 3s = 150 i = 3

$y_1 = 175$ 4s = 200 i = 4

$y_8 = 300$ 5s = 250 i = 5

$y_7 = 300$ 6s = 300 i = 6

Grounded

s = 320 / 2 + 2 - 2 = 160

$g_2 = 320$ s = 160 j = 1

$g_3 = 320$ 2s = 320 j = 2

After sweep algorithm and update we have the results

i,j	1	2	3	4	5	6	7	8
y_{in}	175	-	-	84.75	34.75	130	300	300
x_i	0	30	48	20.25	44.25	0	-	-
g_{jn}	-	320	320	-	-	-	-	-
h_j	-	-	-	-	-	-	320	105

Table 6-13: Results of third iteration of the second problem

The value of the objective function is: Z = 29590.125

After the solution of all feasible pairs, we compare the values of Z, to find the pair that has the minimum Z. For this problem, the optimum solution is given for the pair $(z_1, z_2) = (1, 2)$.

Chapter 7 CONCLUSIONS - FUTURE RESEARCH

In this thesis, we studied a nonlinear mixed integer optimization problem for flight and maintenance planning of mission aircraft, and we developed an analytical methodology that can be utilized for its solution. This methodology is based on the fact that the original problem can be solved through the consecutive solution of several sub-problems. Each of these sub-problems can be solved separately and quite easily, using an existing algorithm for quadratic programming.

The proposed algorithm was implemented in C programming language in order to test its performance. An additional code implementation was developed for the generation of random problem instances. The obtained computational results are very satisfactory, since they reveal that the computational effort does not increase very fast with problem size. This observation is also supported by the computational complexity analysis that we present.

The main contribution of the research reported in this work is that we have developed an exact mixed integer nonlinear programming algorithm for the solution of the Flight and Maintenance Planning problem with a single time period and a single squadron, having very reasonable solution times.

Future research should be directed towards the improvement of the existing algorithm, possibly through more thorough feasibility checks, which would lead to smaller computational times. The question of whether the considered objective functions have special convexity-related properties is also a very important question that remains open for future

research. Note that if this were true, we would be able to reduce the number of considered value-pairs for variables z_1 and z_2, which would lead to significant computational savings.

Another possible direction is the embedment of the present approach within a more generalized algorithm that addresses the general Flight and Maintenance Planning problem with several time periods and squadrons. Finally, it would also be very interesting to evaluate the computational effort required by commercial optimization packages, such as LINGO, in order to solve the problem under consideration. This was not done as part of this work, because the commercial optimization software that our laboratory currently owns (AMPL/CPLEX) does not include a subroutine for solving mixed integer nonlinear programming models.

Appendix A C Implementation of main Algorithm

```
/*------------------------------------------------------------------------

The solution of FMP problem using the aircraft flowchart heuristic

Eftychia Kostarelou<ekostarelou@yahoo.gr> March 2009

This code is part of an implementation for the purposes

of a postgraduate research.

------------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#define N 2500
#define LIMIT 1e-36
#define MAXFLOW 1e38
#define SWAP(a,b) { long double temp=(a);(a)=(b);(b)=temp; }

typedef struct ACTag{
int  n;
long double g_y;
struct ACTag *RightLink;
} ACNode;
```

```
struct node
{
int index;
int data1;
int data2;
struct node *link;
};


/*------------------- Functions: ANSI C prototypes ------------------------*/

long double qmedian(long double a[], int n) ;
void swap(long double *x, long double *y);
void bsort1(long double list[], int n);
void pardalos(long double x[], long double a[], long double b[], long double d, int n);
long double choose_bound(long double x[],long double b[],long double L, long double U,
int n);
long double choose_bound1(long double x[],long double b[],long double L, long double U,
int n);

long double minimum(long double x1 , long double x2);

long double maximum(long double x1 , long double x2);

void mod_solve(long double Y[], long double Xleft[] ,int n, long double apokl[], long double
L, long double U);

void mod_solve1(long double G[], long double Hleft[] ,int n, long double apokl[], long
double L, long double U);

void total_flow(long double apoklisi[N_SIZE]);

void total_flow1(long double apoklisi[N_SIZE]);

void sweep(ACNode **L, int Z2);

void sweep1(ACNode **L,int Z1);

void teliko(int Z1, int Z2, long double *k, long double *p, long double *r, long double *q, int
*v, int *f1, int *f2, long double *Q);

int elegxos();

int elegxos1(int Z1);

void elegxos2(int *number2, int *number3);

void allagi_thesis(int Z1, int Z2);

void teliko(int Z1, int Z2, long double *k, long double *p, long double *r, long double *q, int
*v, int *f1, int *f2, long double *Q);

void epanafora(int Z1, int Z2);

void epanafora1(int Z1);

void epanafora2(int Z2);
```

```
void insert_inplace (int n, long double g_y, ACNode **L, ACNode **T);
void append (int n, long double g_y, ACNode **L, ACNode **K);
void append1 (int n, long double g_y, ACNode **L, ACNode **K);
void delete_head(ACNode **L, ACNode **R);
void delete_end(ACNode **L, ACNode **R);
void  getData();


/*------------------------- main() ---------------------------------------*/

main( )
{
long double start,stop,duration;
int i;
long double s,Y,Xmax,target,G,Ymin,Gmin;
long double L,U;
long double *a;
long double *b;
long double *x;
long double y[N];
long double g[n];
long double x[N];
long double h[N];

FILE *myfile;
myfile=fopen("exodos1.txt","w");
start=clock();

getData();
number = elegxos();
elegxos2(&number2, &number3);
for(Z1=0;Z1<=Sa;Z1++)
        {
                number1 = elegxos1(Z1);
```

```
for(Z2=0;Z2<=(N_SIZE-1-Sa);Z2++)
{
        printf("\nZ1=%d,,,Z2=%d\n\n",Z1,Z2);

        if((Z1-Z2)>dock || number2<Z2 || number<Z1 || number1 == 0)
        {
                //printf("\n\n(Z1,Z2)=(%d,%d) INFEASIBLE\n\n",Z1,Z2);
        }                       }
        else if(number3 > Z2)
        {
                //printf("\n\n(Z1,Z2)=(%d,%d) INFEASIBLE\n\n",Z1,Z2);
        }
        else
        {
                counter++;
                allagi_thesis(Z1, Z2);
                sweep(&Available,Z2);
                sweep1(&Grounded,Z1);
                teliko(Z1, Z2, k, p, r, q, v, &f1, &f2, &Q);

                epanafora(Z1, Z2);
                epanafora1(Z1);
                epanafora2(Z2);
        }
FPRINT OPTIMAL SOLUTION
stop=clock();
duration=(long double)(stop-start)/CLOCKS_PER_SEC;
FPRINT DURATION
fclose(myfile);

/*************************************************/
```

```
/*********************************FUNCTIONS*********************************/

/*****************BASIC FUNCTIONS*************/

void insert_inplace (int n, long double g_y, ACNode **L, ACNode **T){
        ACNode *N, *K, *R;

        N = (ACNode *)malloc(sizeof(ACNode));
        N->n = n;
        N->g_y = g_y;
        N->RightLink = NULL;

        if ((*L) == NULL)
                (*L) = N;
        else if ((*L)->g_y > g_y){
                N->RightLink = (*L);
                (*L) = N;
        }
        else{
                K = (*L);
                R = K->RightLink;
                while (R != NULL && R->g_y < g_y){
                        K = K->RightLink;
                        R = R->RightLink;
                }
                K->RightLink = N;
                N->RightLink = R;
        }

        if ((*T) == NULL) (*T) = (*L);
        else if ((*T)->RightLink != NULL) (*T) = (*T)->RightLink;
}
```

```
void append (int n, long double g_y, ACNode **L, ACNode **K){
        ACNode *N;

        N = (ACNode *)malloc(sizeof(ACNode));
        N->n = n;
        N->g_y = g_y;
        N->RightLink = NULL;

        if ((*K) == NULL){
                (*K) = N;
                (*L) = N;
        }
        else{
                (*K)->RightLink = N;
                (*K) = N;
        }
}

void append1 (int n, long double g_y, ACNode **L, ACNode **K){
        ACNode *N, *R, *current, *M;

        N = (ACNode *)malloc(sizeof(ACNode));
        N->n = n;
        N->g_y = g_y;

        R = N;
        R->RightLink = (*L);
        (*L) = R;
        if ((*K) == NULL){
                (*K) = R;
        }
        else
        {
                current=(*L);
```

```
                while(current != NULL)
                {
                        M=current;
                        (*K)=M;
                        current=current->RightLink;
                }
        }
}

void delete_head(ACNode **L, ACNode **R)
{
        ACNode *K;

        if ((*L) == (*R)){
                free(*L);
                *L = NULL;
                *R = NULL;
        }
        else{
                K = (*L)->RightLink;
                free(*L);
                (*L) = K;
        }
}

        void delete_end(ACNode **L, ACNode **R)
        {
                ACNode *K,*T,*M;

                if ((*L) == (*R)){
                        free(*L);
                        *L = NULL;
                        *R = NULL;
                }
```

```
            else{
                    T=(*L);
                    while(T->RightLink != NULL)
                    {
                            if(T->RightLink == (*R))
                                    M = T;
                            T=T->RightLink;
                    }

                    K = T;
                    free(K);
                    (*R)=M;
                    (*R)->RightLink = NULL;
            }
    }

/****************FEASIBILITY CHECKS*********************/
int elegxos()
{
    int number;
    long double e=0;
    long double sum;
    long double value[N_SIZE]={0};
    ACNode *current;

    sum=u*S;
    number=0;

    current = Available;
    while(current != NULL)
    {
                    value[current->n]=minimum(current->g_y,Xmax);
                    sum -= value[current->n];
                    e=current->g_y-value[current->n];
```

```
                        if(sum>=0 && e==0)
                                number++;
                        current=current->RightLink;
        }
        return number;
}

int elegxos1(int Z1) {
        int number1;
        int i;
        long double sum;
        long double value[N_SIZE]={0};
        ACNode *current;

        sum=l*S;
        number1=0;
        i=0;

        current = Available;
        while(current != NULL)
        {
                if(i<Z1)
                {
                        sum -= current->g_y;
                        i++;
                }
                else
                {
                        value[current->n]=minimum(current->g_y-Ymin,Xmax);
                        sum -= value[current->n];
                }

                current=current->RightLink;
```

```
        }
   if(sum>0)
                number1=0;
        else
                number1=1;
        return number1;
}

void elegxos2(int *number2, int *number3)
{
        int numb2;
        long double sum;
        long double value[N_SIZE]={0};
        long double target;
        ACNode *current;

        sum=0;
        numb2=0;
        target=minimum(B, Sg);

        current = Grounded;
        while(current != NULL)
        {
                value[current->n]=current->g_y;
                sum += value[current->n];
                if(sum<=target
                        numb2++;
                current=current->RightLink;
        }
        *number2=numb2;
        if(numb2==(C-dock))
                *number3=numb2;
        else
                *number3=0;
```

```
}

/**************CHANGE POSITIONS AT FLOWCHARTS*************/
void allagi_thesis(int Z1, int Z2)
{
        int i,j;

        entering=0;
        exiting=0;
        entering_time = 0;
        exiting_time = 0;

        if(Z1>0)
        {
                i=1;
                while(i<=Z1)
                {
                        entering = entering + i;
                        x1[Available->n]=Available->g_y;
                        append(Available->n,G,&Grounded,&Tail_Grounded);
                        entering_time=entering_time+Available->g_y;
                        delete_head(&Available,&Tail_Available);

                        Sa=Sa-1;
                        dock=dock-1;
                        i++;
                }
        }
        else
        {
                entering = 0;
                entering_time=0.0;
        }
        if(Z2>0)
```

```
        {
                j=1;
                while(j<=Z2)
                {
                        exiting = exiting + j;
                        h1[Grounded->n]=Grounded->g_y;
                        append(Grounded->n,Y,&Available,&Tail_Available);
                        exiting_time=exiting_time+Grounded->g_y;
                        delete_head(&Grounded,&Tail_Grounded);
                        Sa=Sa+1;
                        dock=dock+1;
                        j++;
                }
        }
        else
        {
                exiting_time=0;
                exiting = 0;
        }

        LS=l*S;
        LS=LS-entering_time;
        US=u*S;
        US=US-entering_time;
}

/******CONVERTING TO GENERAL FORM FOR AVAILABLE**********/

b=(long double *) malloc (N*sizeof(long double));
for (i=0;i<N;i++) b[i]=Y[i] - apokl[i];
/*apokl[i]=(i+1)*Y/(Sa-z1+z2)*/
a=( long double *) malloc (N*sizeof(long double));
for (i=0;i<N;i++) a[i]=Y[i]- apokl[i] -Xleft[i];
xpar=(long double *) malloc (N*sizeof(long double));
```

```
for (i=0;i<N;i++) xpar[i]=0;
X=( long double *) malloc (N*sizeof(long double));
for (i=0;i<N;i++) X[i]=0;
SWAP(L,U);
L=-L;
U=-U;
for (i=0;i<N;i++) {
L+=(Y[i]- apokl[i]);
U+=(Y[i]- apokl[i]);
}

/******CONVERTING TO GENERAL FORM FOR GROUNDED**********/

b=(long double *) malloc (N*sizeof(long double));
for (i=0;i<N;i++) b[i]=G[i] - apokl[i];
a=( long double *) malloc (N*sizeof(long double));
for (i=0;i<N;i++) a[i]=G[i]- apokl[i] -Hleft[i];
xpar=(long double *) malloc (N*sizeof(long double));
for (i=0;i<N;i++) xpar[i]=0;
H=( long double *) malloc (N*sizeof(long double));
for (i=0;i<N;i++) H[i]=0;
SWAP(L,U);
L=-L;
U=-U;
for (i=0;i<N;i++) {
L+=(G[i]- apokl[i]);
U+=(G[i]- apokl[i]);
}

/******COMPUTE IDEAL VALUES FOR AVAILABLE**********/

apokl[i]=(i+1)*Y/(Sa-z1+z2)
```

```
/******COMPUTE IDEAL VALUES FOR GROUNDED**********/

apokl[i]=(i+1)*G/(N-Sa+z1-z2)

long double minimum(long double x1 , long double x2)
{long double value;
value= (x1<x2) ? x1 : x2;
return value;
}
long double maximum(long double x1 , long double x2)
{long double value1;
value1= (x1>x2) ? x1 : x2;
return value1;
}

void swap(long double *x,long double *y)
{
  long double temp;
  temp = *x;
  *x = *y;
  *y = temp;
}

void bsort1(long double list[], int n)
{
  int i,j;
  for(i=0;i<(n-1);i++)
    for(j=0;j<(n-(i+1));j++)
        if(list[j] > list[j+1])
            swap(&list[j],&list[j+1]);
}
```

```
long double choose_bound(long double a[], long double L, long double U,int n)
{int i;
long double bound[4];
bound[0]=L;
bound[1]=U;
bound[2]=0;
for (i=0;i<n;i++) bound[2]+=a[i];
bound[3]=0;
for (i=0;i<n;i++) bound[3]+=minimum(Xmax,maximum(a[i],0));

bsort1(bound,4);
return bound[2];
}

long double choose_bound1(long double a[], long double L, long double U,int n)
{int i;
long double bound[4];
bound[0]=L;
bound[1]=U;
bound[2]=0;
for (i=0;i<n;i++) bound[2]+=a[i];
bound[3]=0;
for (i=0;i<n;i++) bound[3]+=maximum(a[i],0);

bsort1(bound,4);
return bound[2];
}

void pardalos(long double x[], long double a[], long double b[],long double d,int n)
{int *unsetv;
long double *intervalpts;
long double *temp1;
int *temp2;
long double min=-(long double)MAXFLOW,
```

```
        max=(long double)MAXFLOW;
long double tightsum=0,
        slackweight=0,
        testsum=0;
int i,j=1,counter;
long double mid;
int pts_size;
int unsetv_size;

unsetv= (int *) malloc(n*sizeof(int));

intervalpts= (long double *) malloc((2*n+2)*sizeof(long double));

pts_size=2*n+2;
unsetv_size=n;

for (i=0;i<n;i++) unsetv[i]=(i+1);

for (i=0;i<n;i++) intervalpts[i]=a[i];
for (i=n;i<2*n;i++) intervalpts[i]=b[i-n];
intervalpts[2*n]=-(long double)MAXFLOW;
intervalpts[2*n+1]=(long double)MAXFLOW;

for (;(unsetv_size!=0);){

temp1 = (long double *)malloc(pts_size*sizeof(long double));
memcpy(temp1,intervalpts,pts_size*sizeof(long double));

mid=qmedian(temp1,pts_size);

free (temp1);

testsum=0;
for (i=0;i<unsetv_size;i++) if ((b[unsetv[i]-1]-mid)<0) testsum+=b[unsetv[i]-1];
```

```
                                                    else        if        ((a[unsetv[i]-1]-mid)>0)
testsum+=a[unsetv[i]-1];
                                                    else testsum+=mid;

testsum=testsum+tightsum+slackweight*mid;

/*********UPDATE*******************/

if (testsum<=d) min=mid;
if (testsum>=d) max=mid;

temp1 = (long double *)malloc(pts_size*sizeof(long double));
counter=0;

for (i=0;i<pts_size;i++) if (     ((intervalpts[i]-min)>0) && ((intervalpts[i]-max)<0)        ) {

                        temp1[counter]=intervalpts[i];

                        counter++;

                                                    };
pts_size=counter;
free (intervalpts);
intervalpts = (long double *)malloc(pts_size*sizeof(long double));
memcpy(intervalpts,temp1,pts_size*sizeof(long double));
free (temp1);

temp2 = (int *)malloc(unsetv_size*sizeof(long double));
counter=0;

for (i=0;i<unsetv_size;i++) if ((b[unsetv[i]-1]-min)<LIMIT) tightsum+=b[unsetv[i]-1];
                                                    else    if    ((a[unsetv[i]-1]-max)>-LIMIT)
tightsum+=a[unsetv[i]-1];
                                                    else            if            (((a[unsetv[i]-1]-
min)<LIMIT)&&((b[unsetv[i]-1]-max)>-LIMIT)) slackweight++;
```

```
                                                    else {
                                                            temp2[counter]=unsetv[i];
                                                            counter++;
                                                            };

unsetv_size=counter;

free(unsetv);
unsetv= (int *)malloc(unsetv_size*sizeof(int));
memcpy(unsetv,temp2,unsetv_size*sizeof(int));
free(temp2);

j++;
}

for (i=0;i<n;i++) if  (b[i]<=min)        x[i]=b[i];
                                                    else if (a[i]>=max) x[i]=a[i];
                                                    else     if     ((a[i]<=min)&&(b[i]>=max))
x[i]=(d-tightsum)/slackweight;
}

long double qmedian(long double a[], int n)
{
   int low, high ;
   int median;
   int middle, ll, hh;

   low = 0 ; high = n-1 ; median = (low + high) / 2;
   for (;;) {
      if (high <= low) /* One element only */
         return a[median] ;

      if (high == low + 1) {  /* Two elements only */
         if (a[low] > a[high])
```

```
        SWAP(a[low], a[high]) ;
    return a[median] ;
}

/* Find median of low, middle and high items; swap into position low */
    middle = (low + high) / 2;
    if (a[middle] > a[high])    SWAP(a[middle], a[high]) ;
    if (a[low] > a[high])       SWAP(a[low], a[high]) ;
    if (a[middle] > a[low])     SWAP(a[middle], a[low]) ;

/* Swap low item (now in position middle) into position (low+1) */
    SWAP(a[middle], a[low+1]) ;

/* Nibble from each end towards middle, swapping items when stuck */
    ll = low + 1;
    hh = high;
    for (;;) {
        do ll++; while (a[low] > a[ll]) ;
        do hh--; while (a[hh]  > a[low]) ;

        if (hh < ll)
        break;

        SWAP(a[ll], a[hh]) ;
    }

    /* Swap middle item (in position low) back into correct position */
    SWAP(a[low], a[hh]) ;

    /* Re-set active partition */
    if (hh <= median)
        low = ll;
    if (hh >= median)
```

```
            high = hh - 1;
        }
    }

/******CONVERTING SOLUTION TO PROPER FORM FOR AVAILABLE**********/
for (i=0;i<n;i++) X[i]=(Y[i]-apokl[i])- xpar[i];

for (i=0;i<n;i++) Y[i]=xpar[i]+apokl[i];

Temp = Available;
while (Temp != NULL){
            x[Temp->n] = X[i];
            Temp->g_y = Temp->g_y - X[i];

            Temp = Temp->RightLink;
}
/*************************************************/

/******CONVERTING SOLUTION TO PROPER FORM FOR GROUNDED**********/
for (i=0;i<n;i++) H[i]=(G[i]-apokl[i])- xpar[i];

for (i=0;i<n;i++) G[i]=xpar[i]+apokl[i];

Temp = Grounded;
while (Temp != NULL){
            h[Temp->n] = H[i];
            Temp->g_y = Temp->g_y - H[i];

            Temp = Temp->RightLink;
}
/*************************************************/
#undef SWAP
```

```
/***********FIND THE OPTIMUM SOLUTION**********/

if(KOSTOS[Z1][Z2]<P)
        {
                P=KOSTOS[Z1][Z2];
                *Q=P;
                *f1=Z1;
                *f2=Z2;

                for (i=0;i<N;i++) k[i]=x[i+1];
                for (i=0;i<N;i++) p[i]=y[i+1];
                for (i=0;i<N;i++) q[i]=g[i+1];
                for (i=0;i<N;i++) r[i]=h[i+1];
                for (i=0;i<N;i++) v[i]=a[i+1];
        }

/***********RETURN TO THE INTIAL STATE AND VALUES*********/
if(Z1>0)
        {
                while(i<=Z1)
                {
                        k=Tail_Grounded->n;
                        q=y[Tail_Grounded->n];
                        append1(k,q,&Available,&Tail_Available);
                        delete_end(&Grounded,&Tail_Grounded);
                        i++;
                }

        }

        if(Z2>0)
        {
                while(j<=Z2)
                {
```

```
                p=Tail_Available->n;
                r=g[Tail_Available->n];
                append1(p,r,&Grounded,&Tail_Grounded);
                delete_end(&Available,&Tail_Available);
                j++;
        }
    }

{
    ACNode *current;

    i=1;
    current=Available;
    while(current != NULL)
    {
        if(i<=Z1)
            current->g_y=current->g_y+x1[current->n];
        else
            current->g_y=current->g_y+x[current->n];

        i++;
        current=current->RightLink;

    }
}

{
    ACNode *current;

    i=1;
    current=Grounded;
    while(current != NULL)
    {
        if(i<=Z2)
```

```
                current->g_y=current->g_y+h1[current->n];
        else
                current->g_y=current->g_y+h[current->n];

        i++;
        current=current->RightLink;
    }
}

/***********END OF FUNCTIONS**************/
```

Appendix B C Implementation of Algorithm for the Generation of Random Data

```
int *a;
long double *y,*g;
long double Xmax=50,Y=300,G=320,Ymin=0.1,Gmin=0.1,l=0.95,u=1.05;

FILE *myfile;
myfile=fopen("fleet1.txt","w");

printf("DOSE N:\n");
scanf("%d", &N);

C=0.2*N;
C1=ceil(c);
0.15*N ≤ nao  ≤0.2*N;  //nao: possible # of grounded aircraft
/*Count how many integers are between these two numbers and assume that there are
"count".*/
W=(double)rand()/(double)(RAND_MAX);
        p=0.15*N;

        for(i=0;i<count;i++){
                k=(i*j);
                m=((i+1)*j);

                if(W == 0)
                        f=p;
                else if(W>k && W<=m)
```

```
                f=p;

                p++;

        } /**f is the final number of non-available aircraft**/

for (n=0; n<N-f; n++)
                a[n] = 1;
        if(nao>0)
        {
                for (n=N-f; n<N; n++)
                        a[n] = 0;
        }

for (i=0;i<N;i++){
   if (a[n] == 1){
        y[n]=(((double)rand()/(double)(RAND_MAX)))*300;
        g[n]=0;
   }
}
for (i=0;i<N;i++){
        if (a[n] == 0){
            g[n]=(((double)rand()/(double)(RAND_MAX)))*320;
           y[n]=0;
     }
}
B = 0.80*Sg ;
S = 0.75*Sy;

/**************PRINTF DATA***********/
fprintf(myfile,"%d\n",N);
fprintf(myfile,"%d\n",C);
fprintf(myfile,"\n%.2Lf\n%.2Lf\n%.2Lf\n%.2Lf\n%.2Lf\n%.2Lf\n%.2Lf\n\n", Xmax,
G, Y, Gmin, Ymin, l, u);
```

```
for(n=1; n<=N; n++){
fprintf(myfile,"%d ", a[n-1]);
fprintf(myfile,"%.2Lf ", y[n-1]);
fprintf(myfile,"%.2Lf ", g[n-1]);
}
fclose(myfile);

/**********END OF THE PRODUCTION OF RANDOM DATA**********/
```

References

Arguello, M.F., Bard, J.F. and Yu, G., "Models and Methods for Managing Airline Irregular Operations Aircraft Routing", *Operations Research in the Airline Industry*, Ed: Yu, G., pp: 1-45, Kluwer Academic Publishers, Boston, MA, 1997.

Barnhart, C., Belobaba, P. and Odoni, A.R., "Applications of Operations Research in the Air Transport Industry", *Transportation Science*, 37-4, pp: 368-391, 2003.

Bazaraa, M.S., Sherali, H.D. and Shetty, C.M. (2006). *Nonlinear Programming: Theory and Algorithms.* Wiley-Interscience, New York, USA.

Gavranis, A. (2007), "An exact algorithm for a box constrained class of quadratic programs subject to upper bounds", MSc Thesis, University of Thessaly, Department of Mechanical & Industrial Engineering, Volos, GREECE.

Gopalan, R. and Talluri, K.T., "Mathematical Models in Airline Schedule Planning: A Survey", *Annals of Operations Research*, 76, pp: 155-185, 1998.

Jardine, A.K.S. and Hassounah, M.I., "An Optimal Vehicle-Fleet Inspection Schedule", *The Journal of the Operational Research Society*, 41-9, pp: 791-799, 1990.

Kozanidis, G. and Skipis, A. (2006). "Flight and Maintenance Planning of Military Aircraft for Maximum Fleet Availability: A Biobjective Model". *18th International Conference on*

Multiple Criteria Decision Making. Chania, Greece, http://www.dpem.tuc.gr/fel/mcdm2006/Papers/Kozanidis_Skipis.pdf.

Kozanidis, G. (2009). "A Multiobjective Model for Maximizing Fleet Availability under the Presence of Flight and Maintenance Requirements". *Journal of Advanced Transportation*, 43(2), 155-182.

Kozanidis, G., Gavranis, A. and Liberopoulos, G. (2008). "Heuristics for maximizing fleet availability subject to flight & maintenance requirements". *10th International Conference on Applications of Advanced Technologies in Transportation, Athens, Greece, May 27-31 2008.*

Kozanidis, G., Liberopoulos, G. and Pitsilkas, C. (2010). "Flight and maintenance planning of military aircraft for maximum fleet availability". *Military Operations Research*, accepted.

Kurokawa, T. and Takeshita K., "Air Transportation Planning using Neural Networks as an Example of the Transportation Squadron in the Japan Air Self-Defense Force", *Systems and Computers in Japan*, 35-12, pp: 1223-1232, 2004.

Pardalos, P.M. and Kovoor, N., "An Algorithm for a Singly Constrained Class of Quadratic Programs Subject to Upper and Lower Constraints", *Mathematical Programming*, 46, pp: 321- 328, 1990.

Pippin, B.W., "Allocating Flight Hours to Army Helicopters", MSc Thesis, Naval Postgraduate School, Monterey, CA, USA, 1998.

Radosavljevic, Z. and Babic, O., "Assigning Fighter Plane Formations to Enemy Aircraft using Fuzzy Logic", *Transportation Planning and Technology*, 23-4, pp: 353-368, 2000.

Sgaslik, A., "Planning German Army Helicopter Maintenance and Mission Assignment", MSc Thesis, Naval Postgraduate School, Monterey, CA, USA, 1994.

U.S. DoA. (2000). "Field Manual No. 3-04.500: Army Aviation Maintenance (Appendix D: Maintenance Management Tools)", U.S. Department of the Army, Washington, DC. http://www.globalsecurity.org/military/library/policy/army/fm/3-04-500/.

Yeung, T.G., Cassady C.R., and Pohl, E.A., "Mission Assignment and Maintenance Scheduling for Multi-State Systems", Military Operations Research, 12-1, pp: 19-34, 2007.