

AD-A248 028



AUTOMATING THE WEEKLY FLIGHT
SCHEDULING PROCESS AT THE
USAF TEST PILOT SCHOOL

THESIS

Gary G. Foster, Captain, USAF

DTIC
ELECTE
MAR 3 1 1992
S B D

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

92 3 31 052

92-08111



AFIT/GOR/ENS/92M-10

AUTOMATING THE WEEKLY FLIGHT
SCHEDULING PROCESS AT THE
USAF TEST PILOT SCHOOL

THESIS

Gary G. Foster, Captain, USAF

Approved for public release; distribution unlimited

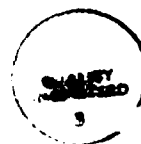
AUTOMATING THE WEEKLY FLIGHT SCHEDULING PROCESS
AT THE USAF TEST PILOT SCHOOL

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

Gary G. Foster, B.S.
Captain, USAF

March 1992



Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Approved for public release; distribution unlimited

THESIS APPROVAL

STUDENT: Captain Gary G. Foster

CLASS: GOR-92M

THESIS TITLE: Automating the Weekly Flight Scheduling Process at
the USAF Test Pilot School

DEFENSE DATE: 3 March 1992

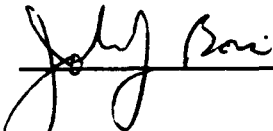
COMMITTEE:

NAME/DEPARTMENT

SIGNATURE

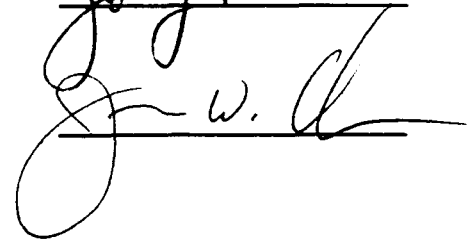
Advisor

Captain John Borsi
(AFIT/ENS)



Reader

Dr James Chrissis
(AFIT/ENS)



Preface

The objective of this study was to develop and automate a solution technique for producing weekly flight schedules at the USAF Test Pilot School. Currently, the flight scheduling process at the Test Pilot School is performed manually. In addition, due to the large quantity of data that must be processed to develop a flight schedule for an entire week, flight schedules at the Test Pilot School are developed on a daily basis. Such a shortsighted approach can often lead to scheduling problems.

Stemming from earlier work accomplished in this area by Captain Lisa Hassel, a mixed integer programming approach to solving this problem was investigated. However, due to the large size of the resulting mixed integer programming model, all known solution techniques were impractical. Consequently, a heuristic algorithm was developed. The heuristic algorithm presented in this thesis demonstrated the capability to produce reasonable weekly flight schedules in less than ten minutes of processing time.

In developing the heuristic algorithm, as well as writing this thesis, I have had a great deal of assistance from others. I would like to express my appreciation to my faculty advisor, Captain John Borsi, for his insight and direction. I would also like to thank Dr. James Chrissis for his help and constructive comments. Finally, I would especially like to thank my family, for making "it" all worthwhile.

Gary G. Foster

Table of Contents

	Page
Preface	ii
List of Figures	vi
List of Tables	vii
Abstract	viii
1. Introduction	1
1.1 Background	1
1.2 TPS Curriculum	3
1.3 TPS Flight Scheduling Process Description	4
1.4 Quality of Training Requirements	6
1.5 Research Objective	7
1.6 Overview	8
2. Literature Review	9
2.1 Scheduling Theory Overview	9
2.1.1 Constraints	9
2.1.2 Scheduling Performance Measures	11
2.2 The Resource-Constrained Scheduling Problem	12
2.3 Problem Formulation of RCS Problems	13
2.3.1 Network Approach	13
2.3.2 Binary Integer Programming Approach	14
2.4 Solution Methods for RCS Problems	15
2.5 Complexity Theory	19
2.6 Heuristic Methods for Solving RCS Problems	20

3. Problem Formulation	24
3.1 Model Formulation	24
3.1.1 Decision Variable Set	27
3.1.2 Objective Function	28
3.1.3 Constraint Sets	29
3.1.4 Workload Leveling	34
3.2 A Small Example Problem	36
3.2.1 Example Problem Description	36
3.2.2 Example Problem Formulation	39
3.2.3 Example Problem Objective Function	40
3.2.4 Example Problem Constraint Sets	41
3.2.5 Example Problem Solution	46
4. Heuristic Approach	48
4.1 Heuristic Approach Background	48
4.1.1 Flight Scheduling Problem Characteristics	49
4.1.2 Flight Scheduling Algorithm Goals	51
4.2 Scheduling Algorithm Description	52
4.3 Coded Algorithm Development and Description	61
4.3.1 Algorithm Testing	63
4.3.2 Testing Limitations	63
4.3.3 Coded Algorithm Output	64
4.3.4 Example Problem Solution	65
4.4 Full Scale Application	66
5. Conclusions and Suggestions	68
5.1 Conclusions	68

5.2 Suggestions for Further Work	70
Appendix A. Heuristic Algorithm Applied to the Example Problem	72
Appendix B. Heuristic Algorithm Applied a Full-Size Problem	79
Appendix C. TPS Flight Scheduling Program Operating Instructions	95
Appendix D. TPS Flight Scheduling Program Source Code	99
Bibliography	129
Vita	131

List of Figures

Figure		Page
1.1	TPS Flight Scheduling Time Line	4
4.1	Scheduling Algorithm Flow	53
4.2	Resource Allocation Flow	56
4.3	Example of Algorithm Progression	62

List of Tables

Table		Page
3.1	Example Problem Instructor Availability	37
3.2	Example Problem Aircraft Availability	38
3.3	Example Problem Mission List	39
3.4	Example Problem Solution Schedule (Mixed Integer Programming Approach)	46
4.1	Example of Re-Prioritizing Missions	60
4.2	Example Problem Solution Schedule (Heuristic Algorithm Approach)	66

Abstract

This study investigated different scheduling solution approaches that could be automated and applied at the USAF Test Pilot School (TPS). Currently at the TPS, flight schedules are manually developed on a daily basis. Weekly flight schedules are not developed due to the large quantity of data that must be processed to develop a flight schedule for an entire week. A weekly flight scheduling approach would reduce the occurrence of scheduling problems and unbalanced resource utilization, both of which are often the result of a daily flight scheduling approach. In addition, posting a flight schedule for the entire week would improve communication between the scheduling staff and affected personnel.

A literature search revealed that the TPS flight scheduling problem belongs to the class of resource-constrained scheduling problems. Furthermore, since such problems are placed in the class of NP-complete problems, heuristic methods are the most practical approach to solving real-size resource-constrained scheduling problems. A heuristic scheduling approach which is capable of producing reasonable weekly flight schedules at the TPS is detailed in this thesis and was incorporated into a software package. The computerized heuristic has demonstrated the capability of producing reasonable flight schedules in minutes for weekly flight scheduling problems consisting of multiple aircraft types, 25 instructor pilots, 50 students, and up to 24 different mission types.

AUTOMATING THE WEEKLY FLIGHT SCHEDULING PROCESS
AT THE USAF TEST PILOT SCHOOL

1. Introduction

1.1 Background

"In support of aircraft test and evaluation, the USAF Test Pilot School (TPS), located at Edwards AFB, trains technically competent test pilots, navigators, and engineers" (8:1). To ensure quality training, the sequence of and time between training events are crucial. Hence, the scheduling of training events is a significant responsibility of the TPS staff -- unfortunately, it is also a time consuming responsibility.

Before each class of students enters the TPS program, major training events (items such as flight techniques and classroom academics) are formed into an overall integrated academics and operations schedule (9). This schedule, although useful as a planning guide for the flight scheduling process, cannot represent or incorporate the inherently dynamic nature of flight operations. The weekly flight schedule is dependent on many variables (aircraft availability, weather, etc.) that cannot be controlled or predicted by the TPS staff. Therefore, the TPS staff is forced to perform flight scheduling on a weekly basis.

In developing the weekly TPS flight schedule, the scheduler must ensure that enough flights are scheduled to maintain pace with the integrated academics and operations schedule. At the same time, the scheduler must also ensure that the students and instructor pilots are not overworked with too many flights (4:3). The scheduler must consider quality of training requirements which govern the minimum and maximum number of days allowed between specific types of student flights. The combination of the guidelines above, unforeseen circumstances (weather, illness, etc), and resource constraints (aircraft, students, instructors, etc) cause the development of a weekly flight schedule to be a complex and time-consuming process.

An investigation into automating the scheduling process at the TPS was conducted in late 1990. The emphasis of this investigation focused on a zero-one (binary) integer programming approach to automating the overall TPS schedule (8:vii). The investigation indicated that a binary integer programming approach is unsuitable for optimizing the overall TPS schedule because the resulting number of variables makes the problem computationally impractical (8:45). Consequently, the investigation recommended the development of heuristic scheduling methods for the TPS scheduling problem (8:46). Heuristic methods are solution approaches that apply knowledge of and experience with a particular problem in order to obtain a solution. They do not guarantee optimal solutions. However, heuristic methods often produce reasonable solutions. The investigation also

classified the TPS scheduling problem as a resource-constrained scheduling problem -- a scheduling situation where resource limitations affect the schedule (1:268). The weekly flight schedule problem falls under this classification because its solution is dependent on the levels of resources (students, instructors, and aircraft) available throughout different periods of the flight week.

1.2 TPS Curriculum

The USAF Test Pilot School provides training in two distinct courses -- the Experimental Test Pilot Course and the Flight Test Engineer/Navigator Course. Although they are separate, the two courses are integrated to instill cooperation and understanding among test team members. Each class consists of 25 students -- typically fifteen pilots, seven engineers, and three navigators. All students attend the same academic courses (6:5). The program duration is currently 44 weeks. TPS classes enter in both January and July; therefore, there is approximately a six month period when two classes are present (9).

The curriculum is divided into four phases; Performance, Flying Qualities, Systems Test, and Test Management (6:5). All four phases have integrated academic and flying programs. Ideally, the Performance, Flying Qualities, and Systems Test phases occur sequentially, and the Test Management phase spans the entire program (9). Various topics are covered in each phase by the following methods; 1) academic theory, 2) flight test

techniques (FTT), 3) flying, and 4) final reports (6:9-35). Students are trained in various flight test techniques through academic lessons, FTT demonstration flights, and FTT data flights. In FTT demonstration missions, students receive hands on training of flight test techniques from instructors.

1.3 TPS Flight Scheduling Process Description

Although flight schedules at the TPS are made on a daily basis, aircraft requests are made for an entire week at a time. The TPS scheduling officer must request resource support (aircraft type, number, day of week, time of day, etc.) from the 6510th Test Wing (4:3). The deadline for this request is six days prior to the start of a given flight week; however, the

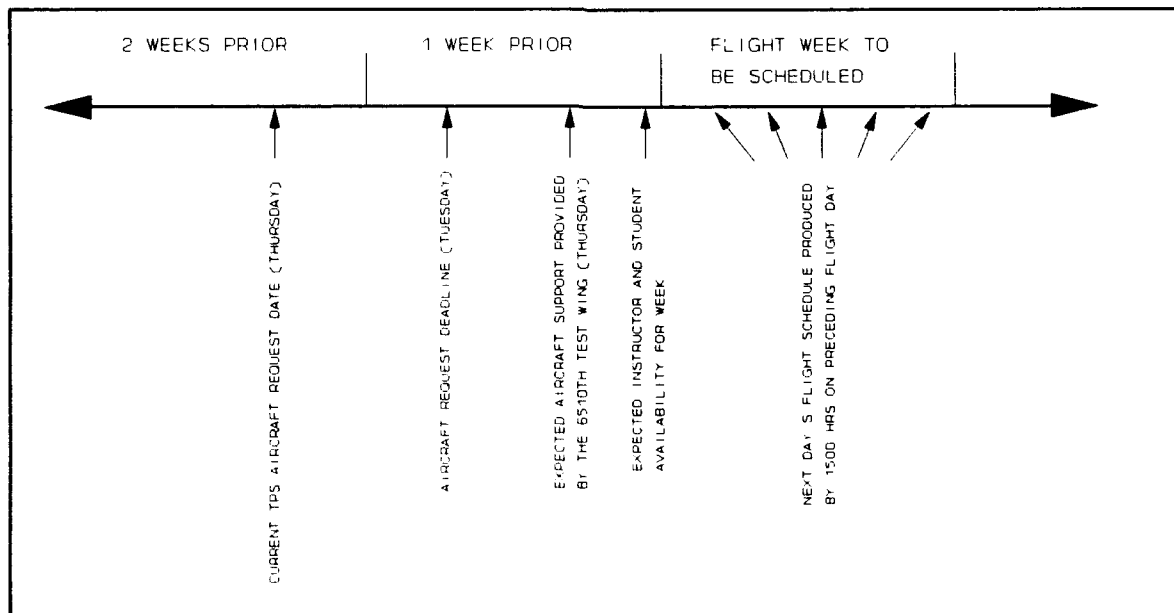


Figure 1.1 TPS Flight Scheduling Time Line

request is usually made on the Thursday two weeks prior to the flight week of concern. Figure 1.1 depicts a time line of the TPS flight scheduling process. Aircraft requests are based on 1) the integrated academics and operations schedule which contains a list of expected flights that should be flown during a particular week, 2) a database containing student flights that are currently scheduled, and 3) flights needed for staff checkouts, upgrades, and proficiency (4:3). On the Thursday before the flight week, the scheduler receives a list of aircraft (type, number, day of week, and time of day) that the 6510th Test Wing expects to be able to support (4:5). Aircraft support is not considered firm until 1500 hours the day prior to a given flight day. Prior to this time, requests can be made for additional aircraft, as well as cancellations of current projected aircraft support. There is no guarantee that requests for additional aircraft will be provided. When the projected aircraft support becomes available, the availability of students and instructors must be obtained in order to develop a complete flight schedule.

Because each scheduled flight has an aircraft type and instruction requirement, and instructor pilots have different aircraft qualifications, flights may require specific instructors as well as specific aircraft. The expected daily availability of students and instructors must also be considered. Student and instructor availability throughout periods of a given week is recorded on a non-availability form. It is the responsibility of the individual students and instructors to ensure this form

accurately represents their availability status throughout the week. There are three flight periods per day; however, students fly only in the first two periods due to academics in the afternoon. Once all information regarding flights and resource availability is obtained, it is used to develop the flight schedule. Within the flight week, non-mission capable aircraft, weather, incomplete missions, and other factors may require that some missions be changed. However, changes to the schedule are kept to a minimum (5:6).

1.4 Quality of Training Requirements

Within the last year, the TPS incorporated additional scheduling requirements in order to enhance the quality of training (9). The purpose of these scheduling requirements primarily concern workload and safety. Many of these requirements address specific FTT missions. Yet, others are common to all FTT missions. For instance, flight restrictions regarding academic test days as well as restrictions on the minimum and maximum number of days between FTT academics, demonstration, and data flights apply to all FTT missions. A complete list of the scheduling restrictions regarding quality can be found in the *Quality of Training Guidelines* (3) at the USAF Test Pilot School.

1.5 Research Objective

The primary objective of this research is to develop a solution method which can be used to automate the TPS weekly flight scheduling process in order to reduce the work required by the TPS staff to produce a weekly flight schedule. The solution method must take into consideration the major guidelines that currently impact the present flight scheduling process. In addition, the solution method must incorporate quality of training requirements which are common to all flight test techniques (3). Feasibility is obviously a goal of a weekly flight schedule; however, the flight schedule must also 1) maintain the pace of the integrated academics and operations schedule, and 2) minimize the number of violations to the quality of training requirements. The importance of student and instructor workload versus quality of training is determined by the TPS and must be reflected in the resulting scheduling method. The automated scheduling method must produce an effective and feasible flight schedule based on the expected availability of resources (students, instructors, and aircraft) for each day of the week. If unforeseen changes in the availability of resources occur during the middle of a flight week, the flight schedule should be adjusted manually in accordance with daily scheduling procedures. The goal of the automated scheduling process is to produce a reasonable initial weekly flight schedule for any given flight week given the projected aircraft support and expected availability of instructor pilots and students.

1.6 Overview

The remaining chapters provide a detailed description of this thesis effort. Chapter 2 contains an overview of the literature that relates to and contributed to the work accomplished in this study. In Chapter 3, the TPS flight scheduling problem is presented as a mathematical programming model. Also in this chapter, a small sample problem is solved based on the mathematical formulation. In Chapter 4, the heuristic algorithm is presented. This chapter describes the development, flow, and performance of the heuristic algorithm. Finally, conclusions and suggestions are presented in Chapter 5.

2. Literature Review

The focus of this chapter is to illustrate and summarize current literature that contributes to the topic of resource-constrained scheduling (RCS). This chapter first presents an introduction to scheduling theory and typical goals of common scheduling problems. A description of the RCS problem and its characteristics is provided, including a short discussion of complexity theory. Various approaches and techniques used for solving RCS problems are also presented.

2.1 Scheduling Theory Overview

In general, scheduling problems are concerned with finding the sequence of activities which is 1) compatible with all constraints and 2) optimal with respect to some criterion of performance (7:5). Activities refer to tasks (or jobs) which must be scheduled for processing/completion. In this study, activities are TPS flights.

2.1.1 Constraints. The sequence in which activities are processed often depends upon three primary types of constraints: technological, precedence, and resource (7:5,48,197). Technological constraints restrict the order in which the operations that comprise a particular activity must be processed

(7:48). For example, if the activity is fixing a flat tire, the operation of removing the lug nuts must be performed prior to removing the flat tire. However, if the level of scheduling detail in a given problem assumes that the activities consist of only a single operation, technological constraints do not apply. Because TPS flights (activities) are scheduled as single operation activities, technological constraints do not apply in this study.

Precedence constraints are similar to and often confused with technological constraints. Precedence constraints limit the choice of schedule by demanding that certain activities (rather than operations within an activity) are accomplished before others (7:48). For example, a valid TPS flight schedule must ensure that students receive their demonstration flight(s) for a given flight test technique prior to their data flight(s) on the same technique.

Resource constraints are usually the most limiting constraint type. Resource constraints limit the choice of schedule based on limitations in resources needed for the completion of activities (7:197). For instance, if a given flight requires an F-4 aircraft, yet an F-4 is only available on Tuesday morning, then the given flight must be flown Tuesday morning. The remaining flights must be scheduled around this resource limitation. In short, constraints are the driving force in most scheduling problems. Once all constraints are satisfied (if possible), schedule feasibility is obtained.

2.1.2 Scheduling Performance Measures. Given that a feasible schedule is attainable, the goal is to find the schedule which is optimal with respect to some criterion of performance. Probably the most common scheduling goal is to minimize the time duration of completing all activities (7:12). With regard to the TPS, the program duration is fixed (44 weeks); therefore, a scheduling goal of trying to minimize the time to complete every TPS activity is not appropriate. Activities must be completed within the duration of the program, but there is no incentive for early completion.

Another common scheduling goal is to minimize the number of late jobs (activities completed after a requested due date) (7:9). Scheduling algorithms that seek to minimize the number of late jobs typically follow the same general pattern. They initially sequence the activities in order of increasing due dates (20:16). That is, the job with the earliest due date is ordered first, and the job with the latest due date is ordered last. Using the initial ordering of jobs, a search is made to find the first late job. If no job is late, the sequence is optimal. If a late job is found, the sub-sequence of jobs up to and including the first late job is examined. The job in the sub-sequence with the longest processing time is moved to the end of the sequence. This process is repeated until the number of late jobs cannot be reduced by re-sequencing the jobs.

With the introduction of resource limitations, an algorithm that only considers processing times would have to be modified.

Given that the jobs require different resources, the sequence in which they are processed also depends on the resource availability over the time that the sequence spans. For example, given that a job is late, moving it to the top of the sequence would not yield any improvement if the resource required to process that job is not available at that time. The scheduling problem could also be expanded to include a sub-goal of producing a schedule that utilizes resources (such as instructors) on a near equal basis. Scheduling objectives may incorporate multiple goals, ranked in order of importance (7:25). Scheduling objectives are numerous, complex, and often conflicting; therefore, it is often difficult to determine a specific objective as being the most beneficial for a particular problem (7:9).

2.2 The Resource-Constrained Scheduling Problem

Resource-constrained scheduling (RCS) problems are concerned with the allocation of resources as well as the processing sequence of the activities (1:5). In most RCS problems, the general resource structure contains multiple units of several different resource types (1:268). At the TPS, there are three resource classes; aircraft, instructors, and students. The manner in which resources are allocated is dependent on the selected sequence of activities. At the same time, a sequence's feasibility depends on the availability of the resources at different time periods during the project duration.

Problem characteristics play a key role in how a given scheduling problem is formulated and solved. The TPS weekly flight scheduling problem has the following characteristics:

- multiple resource types are present;
- resource levels are integral;
- resource levels vary by period, but are known;
- activities are of single operation;
- activity durations are known;
- activities should not be interrupted;
- precedence constraints exist.

Scheduling problems containing these elements can be classified as resource-constrained scheduling problems (19:413). Methods of problem formulation and solution techniques for RCS problems are the topics of discussion in the following sections.

2.3 Problem Formulation of RCS Problems

RCS problems are most commonly formulated as assignment problems with side constraints (12:560). The two primary methods by which this formulation is accomplished are 1) networks with side constraints to represent the resource limitations and other constraints (1:268; 18:1163), and 2) zero-one (binary) integer program (12:560; 14:1206; 15:449).

2.3.1 Network Approach. Since the 1960s, networks have been a popular approach in solving scheduling problems (14:1206). By the pure nature of their design, networks are ideal for handling precedence constraints (7:193). Yet, although many scheduling problems can be solved very efficiently with a network formulation, some scheduling constraints cannot be handled

efficiently within a network structure. Moreover, unless the underlying design matrix of the formulated network is totally unimodular, an integral solution to the LP-relaxation is not guaranteed (2). Non-integral solutions violate the assumption that activities are not to be interrupted once started. For example, in the TPS weekly flight schedule, it is unrealistic to schedule the first half of a flight on Tuesday and the second half of the flight on Friday. Consequently, a network approach may require post-processing of the solution in order to obtain an integral solution.

In addition, networks often obtained their efficiency in obtaining a solution by inherently requiring preprocessing of the problem in order to formulate the problem as a network. Preprocessing refers to reducing the problem size by eliminating solution options which are determined to be infeasible by a review of the constraints or problem structure. Although preprocessing is a beneficial process, it creates a trade-off between the time needed to formulate the problem and the time needed to solve the problem. Networks can be time-consuming and difficult to build -- especially efficient networks which require preprocessing the problem.

2.3.2 Binary Integer Programming Approach. There are various ways in which binary integer programming approaches can be implemented. Such approaches use a zero-one variable to indicate if an activity is to occur in a specified period

(15:450). Each possible assignment is represented by a zero-one variable. If a given zero-one variable is set to one in the final solution, the activity with its corresponding assigned resources should be scheduled. For example, consider the zero-one variable, X_{ijt} .

Where: i represents the activity of interest
 j represents a specific resource
 t represents the time period

If X_{ijt} equals one, then activity i is to be completed using resource j during time period t . Constraints are added to represent problem characteristics such as precedence relationships between activities and resource levels for each period. Formulating the problem as a BIP problem is relatively easy (15:450); however, the number of zero-one variables needed can increase drastically given only linear increases in the number of activities, resource types, and time periods of a given problem. As a result, RCS problems of only moderate size typically result in having an impractical number of zero-one variables (14:1203).

2.4 Solution Methods for RCS Problems

Since LP problems can be solved relatively efficiently, it would seem that solving a BIP problem should be just as easy. After all, with a bounded feasible region, a BIP problem is guaranteed to have a finite number of feasible solutions (10:486). Unfortunately, there are two primary reasons why integer programming problems are much more difficult to solve.

First, even though a bounded feasible region guarantees a finite number of feasible solutions, finite numbers can often be very large. For example, consider a BIP problem with only 20 variables. Such a problem would have $2^{20} = 1,048,576$ possible solutions. Although some of these solutions would be eliminated because they are not feasible, the resulting feasible solution set could still be quite large. Secondly, the simplex method solves LP problems efficiently based on the guarantee that the optimal solution occurs at an extreme point of the feasible region (an intersection of constraints). Unless the optimal extreme point turns out to be integer, the simplex method cannot guarantee an optimal integer solution. As a result, LP problems are generally much easier to solve than integer linear programming problems.

A common approach used in solving integer programming problems is LP-relaxation (10:486). This approach relaxes the integer requirement and solves the problem using an LP-solution approach such as the simplex method. The solution values to the LP-relaxation problem are then rounded to the nearest integral values. Unfortunately, such an approach has two flaws: 1) the resulting integer solution may not be feasible, and 2) it is not guaranteed to be the optimal integer solution. In order to avoid these pitfalls, various techniques have been developed for solving integer programming problems. The next few paragraphs discuss solution approaches that can be applied to BIP problems.

Because of the large number of possible solutions for even a relatively small BIP problem, exhaustive enumeration is not practical in most cases (13:7). Implicit enumeration and branch-and-bound algorithms search through the possible solutions using upper and lower bounds on the objective value to eliminate large numbers of possible solutions without explicitly evaluating them. Assuming a maximization problem, if the solution to the given LP-relaxation sub-problem is less than the lower bound, all solutions expanding out of that sub-problem can be eliminated. Possible solutions can also be eliminated if the solution to the LP-relaxation of a sub-problem is either all integer or infeasible. The efficiency of such approaches can be improved further by incorporating branching selections that are efficient for a particular problem. For example, a depth-first approach usually obtains tight upper and lower bounds more quickly, resulting in more possible solutions being fathomed (eliminated) in fewer computations (13). Lagrangian relaxation is another frequently used technique to develop bounds on the optimal solution (10:498). Lagrangian relaxation restructures the original problem formulation by placing a portion of the problem's constraint set in the objective function. The constraints which make the problem difficult to solve are placed in the objective function with corresponding Lagrangian multipliers such that they act together as a penalty function if constraints are violated. If the Lagrangian multipliers associated with the constraints are chosen well, the solution

will yield a reasonably tight bound on the original problem (10:498).

Cutting plane algorithms are solution approaches that relax the integer constraints but iteratively add constraints that reduce the feasible region of the original problem such that the extreme points of the feasible region become integer. Cutting plane algorithms are careful not to cut off any feasible integer solutions. Given that all the extreme points are integer, the simplex method guarantees optimality.

Additionally, specialized algorithms designed specifically for BIP problems can often be applied. These algorithms obtain their efficiency by exploiting special characteristics (such as angular block structures in the underlying constraint matrix of the problem) that are present in many BIP problems (13). Unfortunately, these techniques can only be applied if the needed characteristics (such as specific types of constraint sets) are present.

In conclusion, although solution procedures for BIP problems exist, they have proved to be unsuccessful in dealing with many problems of practical size due to their computational complexity (14:1203). The computational work required in solving BIP problems is often highly sensitive to both increases in the number of constraints as well as the number of variables (12:570). A BIP approach to solving general RCS problems has been successful for small problems of marginal practical value

but cannot be relied upon for solving large RCS problems which usually result in application.

2.5 Complexity Theory

Mathematical programming problems are often classified based on their complexity. The worst-case time complexity of a solution algorithm describes the maximum number of basic computer operations that would be required to solve a particular problem type (7:140). "The class NP is essentially the set of all problems for which solution algorithms with exponential time complexity have been found" (7:145). Therefore, for problems in the class NP, the number of operations needed to solve the problem can increase exponentially in relation to a linear increase in the problem size. The term NP-complete further classifies a problem as being among the most difficult to solve within the class NP (7:148). Scheduling problems which contain both precedence constraints and resource constraints are categorized as NP-complete problems (14:1204).

For problems that are NP-complete, which includes most arising in scheduling, there are at present no easy solutions. Furthermore, if informed mathematical opinion is correct, there never will be any easy solutions. The only methods available are those of implicit (or explicit) enumeration, which may take a prohibitive amount of computation. Certainly, large NP-complete scheduling problems are for all practical purposes insoluble (7:155).

Since RCS problems are placed in the class of NP-complete problems, heuristic methods are the most practical approach to solving real-size resource scheduling problems (14:1204). It

should be noted, the classification of a problem as NP-complete is not sufficient reason to resort to heuristic methods. The problem must also be large enough that enumerative methods are unmanageable and/or computationally impractical (7:156).

2.6 Heuristic Methods for Solving RCS Problems

In order to obtain a feasible and possibly optimal schedule, heuristic methods are often employed in solving RCS problems. Heuristic methods use fairly simple scheduling rules with the objective of producing reasonably good suboptimal schedules in a reasonable amount of time (1:279). Heuristic methods achieve their efficiency at the expense of not being able to guarantee schedule optimality (or even feasibility in some cases).

In general, heuristic methods apply knowledge of and experience with a particular type of problem to obtain a solution. Therefore, heuristic approaches for solving RCS problems are as numerous as RCS problems. In the literature, the majority of heuristic methods used in solving RCS problems involve two primary steps. The first step is to find an initial feasible solution (12:564; 14:1211). Just finding a feasible solution for a given set of constraints can often be a very difficult problem. The second step applies a solution-improving heuristic that takes a feasible solution and systematically attempts to obtain a better-quality solution while maintaining feasibility (12:564). In reference to the TPS weekly flight

schedule, two aspects of feasibility must be maintained; 1) the precedence of activities, and 2) the availability of resources.

Since the best schedules for RCS problems are usually related to the most efficient use of resources, a commonly used heuristic approach gives priority to those activities requiring the most limited resources. The heuristic then incorporates priority rule methods in which the higher priority activities have precedence over the other activities which also need to be completed (14:1207). Using this type of an approach, activities requiring the most limited resources would be scheduled first, and lower priority activities would be scheduled around the high-priority activities.

Heuristic methods must also take into consideration the scheduling performance measure to be optimized. In the case of the TPS weekly flight schedule, priority may be given to those activities (flights) that are close to violating a quality of training requirement. A heuristic could also incorporate a combination of priorities based on different criteria. For instance, a possible heuristic algorithm may incorporate a modification of a scheduling algorithm which minimizes the number of late jobs with an algorithm which seeks to level the utilization of the various resources. With regard to the TPS weekly flight schedule, such an algorithm would minimize the number of violations to the quality of training requirements and at the same time level the workload for instructors.

A two-criteria heuristic algorithm for scheduling resource-constrained activities was developed by Norbis and Smith (14:1208). Their heuristic algorithm combined critical path and resource utilization criteria into a two-level priority scheme which applied each criterion at different steps in the algorithm. The critical path criterion of their heuristic algorithm is based on activity ready dates and processing times. Due dates are only indirectly taken into account. The criteria are used to determine a priority level for each activity. Activities are iteratively scheduled based on their assigned priority until all activities have been scheduled. The resulting schedule is reviewed for the tardiness of each activity. Those activities with the large tardiness values are reset with a higher priority and the schedule is re-done. This heuristic algorithm by Norbis and Smith could possibly be modified for application to the TPS weekly flight scheduling problem.

To determine the efficiency of different heuristic approaches, heuristic approaches are often compared and evaluated on many different levels using a wide variety of different criteria (19:413). Most heuristic algorithms are rated based on computational efficiency and analytic effectiveness. Computational efficiency relates to the amount of computing resources required to attain specific results. Analytic effectiveness refers to how near a heuristic's solution is to the optimal solution (12:571). If a heuristic is to be used, the goal is to use a heuristic which is within the computational

resource limits of the user and adequately satisfies the performance goals of the given problem.

3. Problem Formulation

The focus of this chapter is to formulate the TPS weekly flight scheduling problem as a binary integer programming (BIP) problem. A BIP approach was chosen over a network formulation primarily because binary integer programming formulations are usually more straight-forward. Hence, the manner in which the TPS weekly flight scheduling problem is represented by binary variables and their corresponding constraint sets is easier to characterize, discuss, and understand. In addition, a small example problem is formulated and solved.

3.1 Model Formulation

The BIP formulation used in this section is representative of the traditional method in which each possible scheduling assignment is represented by a zero-one (binary) variable. In this problem, activities (or jobs) are training missions that must be flown by student test pilots, engineers, and navigators. The sequence in which the training missions are flown is dependent on many factors. For instance, the availability of instructor pilots, students, and aircraft greatly impact the order in which the training missions can be scheduled. Furthermore, this RCS problem is further complicated by the

inclusion of precedence constraints -- those which specify certain missions be flown before others.

The formulation in this chapter is based on a typical flight week during the period of the TPS program when two classes are present which is the most difficult scheduling period. In application, if the TPS weekly flight scheduling problem was formulated as a BIP, the formulation would vary from week to week due to fluctuating resource levels and mission requirements. In addition, some mission constraints (such as precedence and academic test days) may or may not apply for some flight weeks.

When two classes are present, on the average, 75 student training missions are flown per week. The overall resource types and numbers of a typical TPS flight week are:

- 1) 50 students (2 classes of 25)
- 2) 25 instructor pilots
- 3) 12 aircraft types.

The availability of the resources depends on many factors. For instance, the availability of individual students and instructor pilots varies based on such events as leave, TDY, and illness. The availability of aircraft types varies based on the number of each aircraft type requested and the ability of the 6510th Test Group to support the such requests.

Only student flight missions are scheduled in this formulation (instructor proficiency missions and instructor chase support missions are not included). Students fly only in the first two flight periods of the day. Some exceptions exist for special mission requirements or daily TPS agenda; however, such

exceptions are rare and are excluded from the formulation. Hence, given a five-day flight week, there are ten possible flight periods per week in which students fly their missions (two flight periods per day).

As discussed earlier, the flight period in which a given mission can be flown depends on the availability of the student, instructor pilot, and aircraft type needed by the mission. There may be several flight periods within the flight week in which all the required resources are available; however, each mission is to be flown (scheduled) only once. In addition, some missions may often not be flown until later in the flight week due to academic prerequisites. Academic prerequisites pertaining to a given mission type must be completed prior to the mission being flown.

Each given mission requires a specific student, an instructor pilot with a specific qualification, and a specific aircraft type. Instructor pilots are qualified in different aircraft types as well as different flight techniques within an aircraft type. Therefore, a mission can be satisfied with any instructor pilot and aircraft type that meet the requirement and qualification needs of the mission. When zero-one variables are defined, only feasible combinations of mission types, students, instructor pilots, and aircraft types are included. Hence, the number of zero-one variables needed to represent each mission is the product of the number of different instructor pilots that are qualified to fly the mission and the number of flight periods in the week.

3.1.1 Decision Variable Set. Each mission assignment is represented by a binary variable. If a given binary variable is set to one, the activity (mission) and its corresponding time period and resources represented by the given variable should be scheduled. Consider the zero-one variable, X_{mtsia}

where: m represents the mission type
 t represents the flight period
 s represents the student
 i represents the instructor pilot
 a represents the aircraft type.

Based on the characteristics of the TPS resources and operational environment, the subscripts have the following corresponding ranges:

$m = 1, \dots, \# \text{ of missions types (for given flight week)}$
 $t = 1, \dots, 10$
 $s = 1, \dots, 50$
 $i = 1, \dots, 25$
 $a = 1, \dots, 12.$

If it were assumed that all combinations of students, instructors, and aircraft satisfied the needs of a given mission type, the number of zero-one variables needed to formulate this problem would be 150,000 times the number of mission types (10 flight periods * 50 students * 25 instructors * 12 aircraft types). If ten mission types were scheduled to be flown during a given week, 1.5 million zero-one variables would be needed to formulate the problem. However, by preprocessing the data to include only feasible combinations of mission types, instructors, students, and aircraft, the number of zero-one variables needed is drastically reduced. On the average (based on TPS mission descriptions, instructor qualifications, and individual student

curriculum), the requirements of each mission type can be satisfied by five different instructor pilots and one aircraft type. Furthermore, approximately ten different mission types are scheduled to be flown each week, and each mission type is usually flown by eight different students. Therefore, on the average, approximately 4000 zero-one variables ($10 \text{ mission types} * 5 \text{ qualified instructors} * 8 \text{ students} * 1 \text{ aircraft type} * 10 \text{ flight periods}$) would be needed to formulate the weekly flight scheduling problem as a BIP. Although such a reduction in the problem size is significant, the resulting BIP problem is still very large.

3.1.2 Objective Function. As discussed in section 2.1.2, since the TPS program duration is fixed, the objective in solving the TPS weekly flight scheduling problem is not to minimize the time needed to complete all flights. Rather, the goal of the weekly flight scheduling problem is to maintain the pace of the integrated academics and operations schedule, which identifies missions planned to be flown each week of the TPS program. In order to maintain the pace of the integrated academics and operations schedule, the flight scheduling goal should be to schedule as many of the planned missions as possible, given the availability of resources (aircraft, instructor, and student) during the corresponding flight week. Such an objective function would be:

$$\text{MAX } \sum_{t=1}^{10} X_{mtsia} \quad \forall \text{ feasible combinations of } m, s, i, a$$

Using this objective function, the BIP model would maximize the number of missions flown during the week, subject to the following constraint sets.

3.1.3 Constraint Sets. The following constraint sets represent the resource limitations and standard flight operations that exist at the TPS. In the listed constraints, only feasible combinations of missions (m), instructor pilots (i), students (s), and aircraft types (a) are considered.

Instructor Pilot Availability: A given instructor pilot can only fly one mission per flight period, assuming the instructor pilot is available. Therefore, a set of constraints must be formulated to represent the availability of each instructor pilot (i) during each of the ten flight periods (t). This can be represented by the constraints

$$\sum X_{mtsia} \leq b_{ti} \quad \forall \text{ feasible } m, s, a; \text{ given } t, i$$

$$\text{where: } b_{ti} = \begin{cases} 1 & \text{if instructor } i \text{ available in period } t \\ 0 & \text{if instructor } i \text{ not available in period } t. \end{cases}$$

Since there are 25 instructors and 10 flight periods, this set consists of 250 constraints.

Student Availability: As in the case of the instructor pilot, a student can only fly one mission per flight period,

assuming the student is available. Therefore, a set of constraints must also be formulated to represent the availability of each student (s) during each of the ten flight periods (t). This can be represented by the constraints

$$\sum X_{mtsia} \leq b_{ts} \quad \forall \text{ feasible } m, i, a; \text{ given } t, s$$

$$\text{where: } b_{ts} = \begin{cases} 1 & \text{if student } s \text{ available in period } t \\ 0 & \text{if student } s \text{ not available in period } t. \end{cases}$$

Since there are 50 students and 10 flight periods, this set consists of 500 constraints.

Aircraft Availability: Aircraft are typically the most schedule-restricting resource in the TPS flight scheduling problem. Therefore, a set of constraints must be formulated to represent the availability of each aircraft type (a) during each of the ten flight periods (t). This can be represented by the constraints

$$\sum X_{mtsia} \leq b_{ta} \quad \forall \text{ feasible } m, s, i; \text{ given } t, a$$

$$\text{where: } b_{ta} = \text{the number of aircraft type } a \text{ available in period } t.$$

Since there are 12 aircraft types and 10 flight periods, this set consists of 120 constraints.

Fly Missions Only Once: Each mission is to be flown only once; therefore, there must be a set of constraints which

precludes the scheduling of each mission more than once. A mission is defined as the combination of a mission type (m) and specific student (s). Such constraints are represented by

$$\sum X_{mtsia} \leq 1 \quad \forall \text{ feasible } i, a; \text{ given } m, s.$$

The number of constraints in this set depends on the number of students required to fly a given mission type and the number of different mission types. Assuming that, on the average, ten different mission types are planned to be flown in a given week, and that each mission type is typically flown by eight different students, this set would consist of 80 constraints (on the average).

Mission Precedence: Frequently, it is required that one mission be flown by a student before a more advanced mission can be flown by the same student. For cases such as this, a set of constraints must be formulated in order to enforce this restriction (if required given the planned missions). Assuming mission m_x must be flown by student s before mission m_y is flown by the same student s , a constraint must be repeated for each student s . This can be represented by the constraints

$$\sum_{t=1}^{10} t * X_{m_x t s i a} - \sum_{t=1}^{10} t * X_{m_y t s i a} \leq -1 \quad \forall \text{ feasible } i, a; \text{ given } m_x, m_y, s.$$

The number of constraints in this set depends on the number of mission types that have a mission precedence requirement, and the

number of students required to fly such mission types. For example, assuming two different precedence mission sets are planned to be flown in a given week, and that each mission type is flown by eight different students, this set would consist of 16 constraints.

Academic Test Day: On academic test days, students are only allowed to fly one mission. Therefore, a set of constraints must be formulated in order to enforce this restriction (if an academic test is planned for the given week). Assuming the academic test day is represented by k , and the corresponding first flight period of this test day is represented by t_k , the following single constraint must be repeated for each student s within the class having the academic test. This can be represented by the constraints

$$\sum X_{mt_k sia} + \sum X_{mt_{k+1} sia} \leq 1 \quad \forall \text{ feasible } m, i, a; \text{ given } s.$$

Since students are divided into two classes, the number of constraints in this set depends on the number of students in each class. If a test is planned for only one class, this set contains up to 25 constraints (one constraint for each of the 25 students per class). If a test is planned for both classes, this set contains up to 50 constraints.

Mission Ready Dates: Missions cannot be flown until after academic prerequisites have been completed for the given mission

type. Therefore, a set of constraints must be formulated to ensure that missions are not scheduled to be flown before their corresponding academic prerequisites are completed. Assuming the academic prerequisite is to be completed at the end of day c , and the corresponding second flight period of this day is represented by t_c , the following constraint must be repeated for each combination of student that has not completed the academic prerequisite for the given mission type. This can be represented by the constraints

$$\sum_{t=1}^{t_c} X_{mtsia} \leq 0 \quad \forall \text{ feasible } i, a; \text{ given } m, s.$$

The number of constraints in this set depends on the number of students that have not completed their academic prerequisites for a given mission type, and the number of different mission types for which such a situation exists.

In summary, based on a typical flight week, a BIP formulation of the flight scheduling problem at the TPS consists of approximately 4000 binary variables and 1000 constraints. The complete formulation is:

Objective Function:

$$\text{MAX } \sum_{t=1}^{10} X_{mtsia} \quad \forall \text{ feasible combinations of } m, s, i, \text{ and } a$$

Subject to:

Instructor Pilot Availability:

$$\sum X_{mtsia} \leq b_{ti} \quad \forall \text{ feasible } m, s, a; \text{ given } t, i$$

Student Availability:

$$\sum X_{mtsia} \leq b_{ts} \quad \forall \text{ feasible } m, i, a; \text{ given } t, s$$

Aircraft Availability:

$$\sum X_{mtsia} \leq b_{ta} \quad \forall \text{ feasible } m, s, i; \text{ given } t, a$$

Fly Missions Only Once:

$$\sum X_{mtsia} \leq 1 \quad \forall \text{ feasible } i, a; \text{ given } m, s$$

Mission Precedence:

$$\sum_{t=1}^{10} t * X_{m_x tsia} - \sum_{t=1}^{10} t * X_{m_y tsia} \leq -1 \quad \forall \text{ feasible } i, a; \text{ given } m_x, m_y, s$$

Academic Test Day:

$$\sum X_{mt_k sia} + \sum X_{mt_{k+1} sia} \leq 1 \quad \forall \text{ feasible } m, i, a; \text{ given } s$$

Mission Ready Dates:

$$\sum_{t=1}^{t_c} X_{mtsia} \leq 0 \quad \forall \text{ feasible } i, a; \text{ given } m, s$$

Variable Type Restrictions:

$$X_{mtsia} \in B^n$$

3.1.4 Workload Leveling. The preceding formulation may be enhanced so as to incorporate a sub-objective of leveling the workload of the different instructor pilots. To accomplish this,

continuous variables are added to the formulation. The objective function is altered and a constraint set is added to the formulation in such a manner that the value of the objective function decreases if the workload of any given instructor pilot(s) exceeds the given target (or goal) instructor pilot mission workload level. The objective function becomes

$$\text{MAX } \sum_{t=1}^{10} X_{mtsia} - \sum_{i=1}^{25} p * Y_i \quad \forall \text{ feasible } m, s, i, \text{ and } a$$

where: Y_i = *the the number of missions instructor i is scheduled to fly in excess of the workload goal number of missions.*

This objective function assumes a benefit (or cost coefficient) of one for missions scheduled, X_{mtsia} , and a penalty of p for excess workloads, Y_i . These values represent the importance of scheduling missions versus leveling instructor pilot workloads. If the TPS is willing to schedule instructor pilots to fly more missions than desired in order to fly more student training missions, then $0 < p < 1$. On the other hand, if instructors are not to be overworked, then $p > 1$.

In addition to altering the objective function, another constraint set would have to be added to the formulation in order to control the values of the Y_i in the objective function. As shown, the value of a given Y_i represents the amount that the corresponding instructor pilot exceeds the goal workload level. Therefore, a constraint must be added for each instructor pilot

(i) to represent this relationship. This can be represented by the constraints

$$\sum_{t=1}^{10} X_{mtsia} - Y_i \leq b_{wk} \quad \forall \text{ feasible } m, s, a; \text{ given } i$$

where: b_{wk} = the goal instructor pilot mission workload.

In short, the addition of this sub-objective would add 25 ordinary continuous variables and their corresponding constraints to the formulation. With the addition of these variables, the problem formulation would no longer be a pure BIP. Furthermore, although the Y_i are continuous variables, their solution values are always integer (assuming b_{wk} is integer) because they are dependent upon the sum of binary variables.

3.2 A Small Example Problem

The purpose of this example problem is to formulate and solve a realistic, but small, flight scheduling problem. This small example problem is formulated as a BIP and solved using the SAS linear programming software package on a VAX 8550 mainframe computer.

3.2.1 Example Problem Description. This problem addresses the scheduling of 17 missions during a given flight week. The primary objective is to determine a schedule which maximizes the number of training missions flown during the week. A secondary

objective to level the workload of the instructor pilots is also applied. The schedule must also be within the given operating guidelines and resource constraints. There are seven different students ($s = 1, \dots, 7$) that need training missions. In addition, there are seven different mission types ($m = 1, \dots, 7$). Three instructor pilots are available for use ($i = 1, 2, 3$), and each instructor has the following different aircraft qualifications:

<u>Instructor</u>	<u>Qualifications</u>
1	C-23, T-38
2	C-23, T-38, F-4
3	T-38

In addition, the availability of instructors varies throughout the week due to medical/dental appointments, TDY, leave, etc. The availability of the three instructor pilots for this example problem is shown in Table 3.1.

TABLE 3.1
EXAMPLE PROBLEM INSTRUCTOR PILOT AVAILABILITY

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
PERIOD 1	1 2 3	1 2	1 3	1 2 3	1 2 3
PERIOD 2	2 3	1 2 3	2 3	1 2 3	1 3

The availability of students must also be considered; however, for this example problem, it is assumed that all students are available during every flight period throughout the

week. There are only two flight periods per day in which student training missions can be flown ($t = 1, \dots, 10$). Three different aircraft types are flown ($a = 1, \dots, 3$). Table 3.2 depicts the expected aircraft availability for the week.

TABLE 3.2
EXAMPLE PROBLEM AIRCRAFT AVAILABILITY

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
PERIOD 1	2 T-38 1 F-4	1 T-38 1 C-23	1 T-38 1 C-23 1 F-4	2 T-38 1 F-4	1 T-38 1 F-4
PERIOD 2	1 T-38 1 C-23	1 T-38 1 C-23 1 F-4	1 T-38 1 F-4	1 T-38 1 C-23	1 T-38 1 C-23

(1=T-38, 2=C-23, 3=F-4)

Each of the missions and their corresponding requirements are depicted in Table 3.3. Each mission is to be flown only once. Furthermore, each of the missions has different requirements regarding aircraft type, instructor and student. In addition, certain missions must be flown before others. Check flights for a given aircraft must be flown first. Also, for a given flight topic (such as LS in the MISSION TYPE column of Table 3.3) containing both demo and data flights, the demo flights must be flown first. Data flights do not require an instructor pilot.

TABLE 3.3

EXAMPLE PROBLEM MISSION LIST

STUDENT (s)	IP (i) REQUIRED	AIRCRAFT TYPE (a) REQUIRED	MISSION TYPE (m)
1	1 or 2	C-23 (2)	check flight (1)
2	1 or 2	C-23 (2)	check flight (1)
1	1 or 2	C-23 (2)	PERF demo (2)
2	1 or 2	C-23 (2)	PERF demo (2)
3	1, 2, or 3	T-38 (1)	LS demo (3)
4	1, 2, or 3	T-38 (1)	LS demo (3)
5	1, 2, or 3	T-38 (1)	LS demo (3)
3	N/A	T-38 (1)	LS data (4)
4	N/A	T-38 (1)	LS data (4)
5	N/A	T-38 (1)	LS data (4)
3	1, 2, or 3	T-38 (1)	range demo (5)
4	1, 2, or 3	T-38 (1)	range demo (5)
5	1, 2, or 3	T-38 (1)	range demo (5)
6	2	F-4 (3)	structures (6)
7	2	F-4 (3)	structures (6)
6	2	F-4 (3)	propulsion (7)
7	2	F-4 (3)	propulsion (7)

3.2.2 Example Problem Formulation. Each candidate mission must be described by a binary variable. Using the problem description, the number of binary variables needed to represent each mission (combination of mission type and specific student) is the product of the number of different instructor pilots and aircraft types that satisfy the requirements of the mission type,

and the number of flight periods in the week. For instance, the number of variables needed to represent the first mission listed is (2 instructors) * (1 aircraft type) * (10 flight periods) or 20. Applying this method to each mission, 330 binary variables are needed to represent all of the possible feasible missions. Consider the zero-one variable X_{mtsia} . For this given example problem, the subscripts have the following corresponding ranges:

$$\begin{aligned} m &= 1, \dots, 7 \\ t &= 1, \dots, 10 \\ s &= 1, \dots, 7 \\ i &= 1, \dots, 4 \quad (i=4 \text{ if no instructor needed}) \\ a &= 1, \dots, 3 \end{aligned}$$

The variable, $X_{1,3,1,2,2}$, represents a possible scheduling of the first mission listed in Table 3.3. This variable specifically represents the scheduling of the first mission type ($m=1$), in the third flight period ($t=3$), with student #1 ($s=1$), and instructor #2 ($i=2$), in a C-23 ($a=2$).

3.2.3 Example Problem Objective Function. The primary objective is to determine a schedule which maximizes the number of training missions flown during the week. However, a secondary goal of leveling the workloads of the instructor pilots is also included in the objective function. To accomplish this, three additional continuous variables, Y_i ($i = 1, 2, 3$), representing the excess workloads for each of the three instructor pilots is added to the formulation. The objective function is

$$\text{MAX} \sum_{t=1}^{10} X_{mtsia} - \sum_{i=1}^3 0.9 * Y_i \quad \forall \text{ feasible } m, s, i, a.$$

This objective function assumes a benefit equal to one for missions scheduled (X_{mtsia}), and a workload penalty, p , equal to 0.9 for the amount the workload goal is exceeded (Y_i). Since $0 < p < 1$, instructor pilots might exceed the workload goal in order to fly more missions (the benefit of flying an additional mission is greater than the cost of an instructor pilot flying one more missions than desired). For this example problem, 14 missions require an instructor pilot. Since there are only 3 instructor pilots, the goal instructor pilot workload level, b_{wk} , is set at 5 missions (approximately 14 missions / 3 instructors).

3.2.4 Example Problem Constraint Sets.

Instructor Pilot Availability:

$$\sum X_{mtsia} \leq b_{ti} \quad \forall \text{ feasible } m, s, a; \text{ given } t, i$$

$$\text{where: } b_{ti} = \begin{cases} 1 & \text{if instructor } i \text{ available in period } t \\ 0 & \text{if instructor } i \text{ not available in period } t \end{cases}$$

Since there are 3 instructor pilots and 10 flight periods, this set consists of 30 constraints. The values of each of the b_{ti} can be obtained using the instructor pilot availability data contained in Table 3.1.

Student Availability:

$$\sum X_{mtsia} \leq 1 \quad \forall \text{ feasible } m, i, a; \text{ given } t, s$$

Since there are 7 students and 10 flight periods, this set consists of 70 constraints. In addition, since it was assumed that every student is available every flight period of the week, the right hand side, b_{ts} , of each of the constraints is always 1.

Aircraft Availability:

$$\sum X_{mtsia} \leq b_{ta} \quad \forall \text{ feasible } m, s, i; \text{ given } t, a$$

where: b_{ta} = the number of aircraft type a available in period t

Since there are 3 aircraft types and 10 flight periods, this set consists of 30 constraints. The values of each of the b_{ta} can be obtained from the aircraft availability data contained in Table 3.2.

Fly Missions Only Once:

$$\sum X_{mtsia} \leq 1 \quad \forall \text{ feasible } i, a; \text{ given } m, s$$

Since there are 17 combinations of mission types (m) and students (s), this set consists of 17 constraints.

Mission Precedence: In this example problem, certain mission types must be flown before others. Before the C-23 PERF demo can be flown by a student, that same student must first fly the C-23 check flight. A similar situation is true for students flying the T-38. The T-38 LS demo flight must be flown before the T-38

LS data flight. The set of constraints below enforces these precedence relationships.

C-23 flights (with students $s = 1$ and 2):

$$\sum_{t=1}^{10} t * X_{1tsi2} - \sum_{t=1}^{10} t * X_{2tsi2} \leq -1 \quad \forall \text{ feasible } i$$

T-38 LS flights (with students $s = 3, 4,$ and 5):

$$\sum_{t=1}^{10} t * X_{3tsi1} - \sum_{t=1}^{10} t * X_{4tsi1} \leq -1 \quad \forall \text{ feasible } i$$

Two different precedence mission sets are planned to be flown, and one mission set (C-23) is flown by two different students and the other mission set (T-38) by three different students; therefore, this set consists of 5 constraints.

Academic Test Day: An academic test is scheduled for Wednesday, students can fly a maximum of one mission on Wednesday. Since the test day is Wednesday, $t_k = 5$ (the first flight period of the test day). The following constraint set enforces this restriction.

$$\sum X_{m5sia} + \sum X_{m6sia} \leq 1 \quad \forall \text{ feasible } m, i, a; \text{ given } s$$

Given that there are 7 students, and all are in the same class, this set contains 7 constraints (one for each student).

Instructor Pilot Workload:

$$\sum_{t=1}^{10} X_{mtsia} - Y_i \leq 5 \quad \forall \text{ feasible } m, s, a; \text{ given } i$$

The goal instructor pilot workload level, b_{wk} , is set at 5 missions (approximately 14 missions / 3 instructors). Since there are only 3 instructor pilots, this set consists of 3 constraints.

Mission Ready Dates: For this example problem, mission type 7 cannot be flown until Thursday of the given flight week because the academics for this mission type are not completed until Wednesday. Hence, $t_c=6$ and the resulting constraint set is the following.

For students $s = 6$ and 7 :

$$\sum_{t=1}^6 X_{7tsia} \leq 0 \quad \forall \text{ feasible } i, a; \text{ given } s$$

Given that there are only 2 students flying mission type 7, this set consists of 2 constraints.

In summary, this relatively small example problem, when formulated as a mixed integer programming problem (MIP), consists of 330 binary variables, 3 continuous variables, and 164 constraints. The complete formulation is:

Objective Function:

$$\text{MAX } \sum_{t=1}^{10} X_{mtsia} - \sum_{i=1}^3 0.9 * Y_i \quad \forall \text{ feasible } m, s, i, a$$

Subject to:

Instructor Pilot Availability:

$$\sum X_{mtsia} \leq b_{ti} \quad \forall \text{ feasible } m, s, a; \text{ given } t, i$$

Student Availability:

$$\sum X_{mtsia} \leq 1 \quad \forall \text{ feasible } m, i, a; \text{ given } t, s$$

Aircraft Availability:

$$\sum X_{mtsia} \leq b_{ta} \quad \forall \text{ feasible } m, s, i; \text{ given } t, a$$

Fly Missions Only Once:

$$\sum X_{mtsia} \leq 1 \quad \forall \text{ feasible } i, a; \text{ given } m, s$$

Mission Precedence for C-23 flights with students $s = 1, 2$:

$$\sum_{t=1}^{10} t * X_{1tsi2} - \sum_{t=1}^{10} t * X_{2tsi2} \leq -1 \quad \forall \text{ feasible } i$$

and for T-38 LS flights with students $s = 3, 4$, and 5:

$$\sum_{t=1}^{10} t * X_{3tsi1} - \sum_{t=1}^{10} t * X_{4tsi1} \leq -1 \quad \forall \text{ feasible } i$$

Academic Test Day:

$$\sum X_{m5sia} + \sum X_{m6sia} \leq 1 \quad \forall \text{ feasible } m, i, a; \text{ given } s$$

Instructor Pilot Workload:

$$\sum_{t=1}^{10} X_{mtsia} - Y_i \leq 5 \quad \forall \text{ feasible } m, s, a; \text{ given } i$$

Mission Ready Date for mission type $m = 7$ and students $s = 6, 7$:

$$\sum_{t=1}^6 X_{7tsia} \leq 0 \quad \forall \text{ feasible } i, a; \text{ given } s$$

and

$$X_{mtsia} \in B^{330} \quad Y_i \geq 0$$

3.2.5 Example Problem Solution. The formulation for the small sample problem was solved using the SAS/OR integer program solver (reference SAS/OR LP Manual) on a Digital Equipment Corporation (DEC) VAX 8550 with 64 megabytes of main memory. The problem required approximately 5 CPU minutes to obtain an optimal solution. The objective function value was 17, with every training mission successfully scheduled during the week and no instructor pilot exceeding the workload goal. Table 3.4 depicts the corresponding schedule. By a review of the schedule, it can be seen that all the constraints have been satisfied.

TABLE 3.4

**EXAMPLE PROBLEM SOLUTION SCHEDULE
(Mixed Integer Program Formulation)**

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
	s m a i	s m a i	s m a i	s m a i	s m a i
PERIOD 1	4 5 T-38 i	2 1 C-23 1	3 5 T-38 3	3 3 T-38 3 5 4 T-38 4 6 7 F-4 2	4 3 T-38 3 7 7 F-4 2
PERIOD 2	5 5 T-38 2	2 2 C-23 1 7 6 F-4 2	5 3 T-38 3 6 6 F-4 2	1 1 C-23 1 3 4 T-38 4	1 2 C-23 1 4 4 T-38 4

(s = student, m = mission type, a = aircraft type, i = instructor pilot)

The solution method applied in this example was a branch and bound approach with LP-relaxation and required 5 CPU minutes to obtain the optimal solution. In addition, since resource-constrained scheduling problems such as this are in the class of NP-complete problems, all known solution algorithms require, in

the worst case, an amount of time which is an exponential function of the size of the problem. Earlier in this chapter, the problem size for a typical TPS flight week was shown to be approximately 4000 variables and 1000 constraints, while the example problem size was 333 variables and 162 constraints. Consequently, since the actual TPS weekly flight scheduling problem would be approximately 10 to 20 times larger than the example problem, a polynomial-time heuristic approach is warranted.

4. *Heuristic Approach*

This chapter presents a heuristic approach to solving the TPS weekly flight scheduling problem. The heuristic approach presented in this chapter exploits the characteristics of the TPS weekly flight scheduling problem in order to obtain reasonable schedules in a relatively small amount of time. A detailed description of the heuristic is presented. In addition, to give a comparison of scheduling approaches, a schedule for the small example problem solved in Chapter 3 is developed using the heuristic approach.

4.1 *Heuristic Approach Background*

As mentioned in Chapter 3, the major difficulty in obtaining a weekly flight schedule for the TPS stems from the large size of the problem when formulated as a mixed integer programming (MIP) problem. Although a mixed integer programming formulation is a legitimate approach to develop good flight schedules, good flight schedules can also be obtained on a regular basis by exploiting characteristics of the weekly flight scheduling problem and applying a relatively simple heuristic algorithm. The heuristic algorithm presented in this chapter is essentially a computerized or automated replication of the procedures performed by the TPS flight scheduler as performed on a daily basis. Such an approach

has not been used at the weekly level because a human scheduler becomes overwhelmed with information when developing a flight schedule for an entire week. By automating the procedures used by the TPS flight scheduler, a schedule can be developed in seconds. The algorithm exploits this speed by developing multiple schedules in order to improve its ability to make good schedules. Each time a schedule is developed, the algorithm learns information (such as which missions require resources that are the most limiting) about the given flight week scheduling problem. After each schedule is developed, the best schedule developed thus far is saved and the algorithm applies the information learned to develop the next schedule. At each schedule iteration, missions are re-prioritized in the scheduling algorithm based on what has been learned. Missions that are more difficult to schedule gain higher priority. After several scheduling iterations, the best schedule developed is output.

4.1.1 Flight Scheduling Problem Characteristics. Although formulating and solving the TPS weekly flight scheduling problem as a MIP problem may be an impractical approach to solving this problem, the MIP formulation did provide useful insights regarding the characteristics of this specific resource-constrained scheduling problem. In addition, further insights were obtained by observing the TPS scheduler develop a daily flight schedule.

The most significant characteristic of the TPS weekly flight scheduling problem is that aircraft are the most schedule-restricting resources. Typically, if a mission is not flown in a given week, it is due to shortages in aircraft. If an aircraft is available, it is almost always used by the TPS. What this means in terms of the scheduling problem is that if every available aircraft has been assigned to a mission, then the maximum possible number of missions has been scheduled for that flight week. A mission cannot be scheduled (flown) without an aircraft. On the other hand, student and instructor pilots are usually available. Obviously, their availability must be checked, but usually there is enough flexibility in student and instructor pilot availability to develop a schedule which also levels instructor pilot workload.

Another important characteristic of the TPS weekly flight scheduling problem is the number of missions eligible to be flown in any given week. When a mission type is scheduled to be flown, it must be flown by several students. There is usually a two-week window within which a mission type must be flown. It is not expected that every student required to fly a given mission type completes that mission in the first flight week. If a mission type is not flown by a particular student in the first week, it is to be flown in the next. The integrated academics and operations schedule structures the flights in this manner. Consequently, unlike the case presented in the small example problem where there were more aircraft available than missions to

fly them, usually more missions are eligible to be flown in a given flight week than could possibly be flown, given limited aircraft availability for the week. In other words, for most weeks, the TPS weekly flight scheduling problem is mission rich, meaning that there are significantly more missions eligible to be flown than there are aircraft to support them. In addition, several instructor pilots are usually qualified to fly each given mission type; therefore, instructor pilot availability rarely prohibits a given mission type from being flown. Also, since several instructor pilots are usually qualified to fly any particular mission type, instructor pilot workload can more easily be leveled.

4.1.2 Flight Scheduling Algorithm Goals. Based on comments from the TPS scheduling staff, a goal of this scheduling algorithm should be to determine the schedule which utilizes instructor pilots as evenly as possible in addition to maximizing the number of missions flown during a given flight week. Since aircraft are the most scheduling-restrictive resource and student and instructor pilot availability typically allows for an abundance of different scheduling assignments, multiple schedules that maximize the number of missions flown (every aircraft utilized) can usually be developed. Although, an algorithm that develops such schedules could be performed by a human, it would be very time consuming due to the large amounts of data that must be evaluated at each step. Therefore, in order for an algorithm

to be useful it must be automated and the weekly flight scheduling information must be contained in a database. By manipulating the database in a logical, systematic manner, a schedule can be developed which seeks to maximize the number of sorties flown and level the workload of the instructor pilots.

Since the TPS weekly flight scheduling problem must be solved each week, data entry must be kept to a minimum. Therefore, the manner in which the data corresponding to a given flight week is entered into the database is of primary concern. The goal of this thesis is not just to develop a solution approach to the weekly flight scheduling problem, but to also present the TPS scheduling staff with a software package that would reduce their workload in producing a weekly flight schedule. The algorithm was designed to read data from external data files which can be updated to represent a given flight week's characteristics (resource availability, eligible missions, etc.) in less than one hour.

4.2 Scheduling Algorithm Description

This section provides an overview of the primary steps contained in the weekly flight scheduling algorithm. Flow charts representing the algorithm flow are provided in Figures 4.1 and 4.2. Figure 4.1 represents an overview of the overall flow and Figure 4.2 represents a more specific overview of the resource allocation flow.

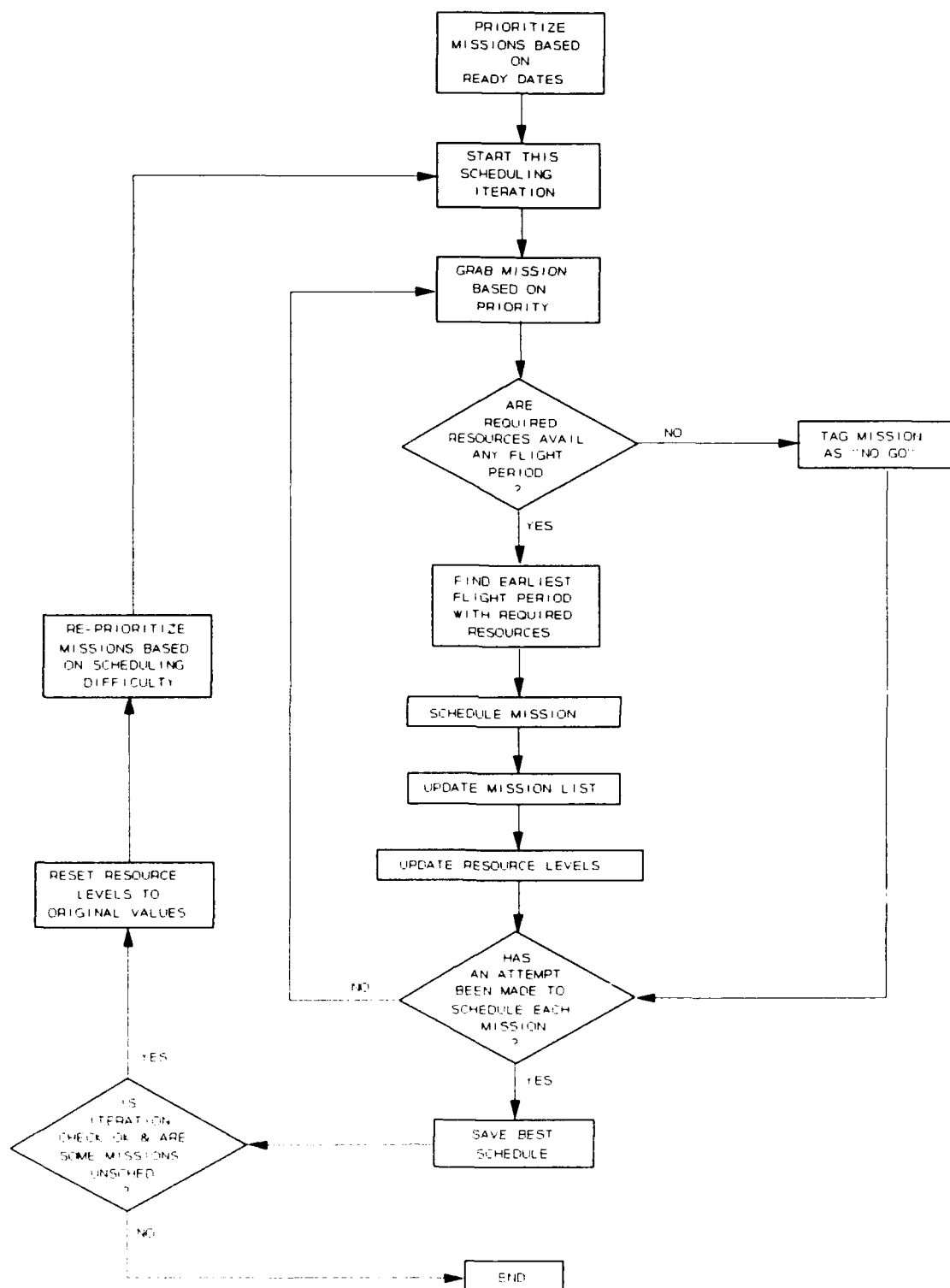


Figure 4.1 Scheduling Algorithm Flow

Step 1 (Initialization): Once all flight information is available, missions are ranked based on their ready dates, the earlier a mission's ready date, the higher its ranking. Missions which are bound by precedence constraints (that is, they require a specific mission to be flown before they can be flown) are ranked last.

Step 2 (Determine Resource Requirements): Once missions are prioritized (or re-prioritized), the algorithm attempts to schedule each mission based on its ranking. Once a specific mission is selected for scheduling, the algorithm determines the resources required for that mission to be flown. These resources include the specific aircraft type and a list containing each instructor pilot that is qualified to fly the specific mission and aircraft type. Additionally, the algorithm checks if the given mission type requires a specific mission to be flown before it. If so, the required precedence mission must already be scheduled in order for this mission to be scheduled. If the precedence mission has been scheduled, the follow-on mission is assigned a ready date for the day after the flight period in which the precedence mission was scheduled.

Step 3 (Mission Scheduling): Once a mission's resource requirements are determined, the algorithm attempts to schedule the mission in the earliest flight period in which all of its required resources are still available. Figure 4.2 provides an overview of the flow for this step of the algorithm.

The first step (**step 3a**) is to determine the first flight period after the given mission's ready date in which the required aircraft type is available. If the required aircraft type is not available in any of those flight periods, the mission cannot be scheduled and the algorithm returns to step 2 to select the next mission for scheduling. Since aircraft are the most restricting resource, aircraft availability is checked first to minimize the number of steps before it is determined that resources are not available for the mission to be flown. If the aircraft type is available, the earliest flight period in which the given aircraft type is available is chosen.

Given this flight period, the next step (**step 3b**) is to select an instructor pilot (if needed). Using the list of qualified instructor pilots developed in step 2, the algorithm selects the instructor pilot with the lowest workload who is available in the given flight period. If no qualified instructor pilots are available in this specific period, the mission's ready date is shifted to one flight period later in the week and the algorithm returns to step 3a.

Once an instructor pilot is assigned (or if an instructor pilot is not needed for the given mission), the next step (**step 3c**) is to determine if the required student is available in the given flight period. It may seem more logical to check to see if the student is available first since the student is directly tied to the mission. However, this is not the case for two reasons; 1) aircraft and instructor pilot availability drive the schedule,

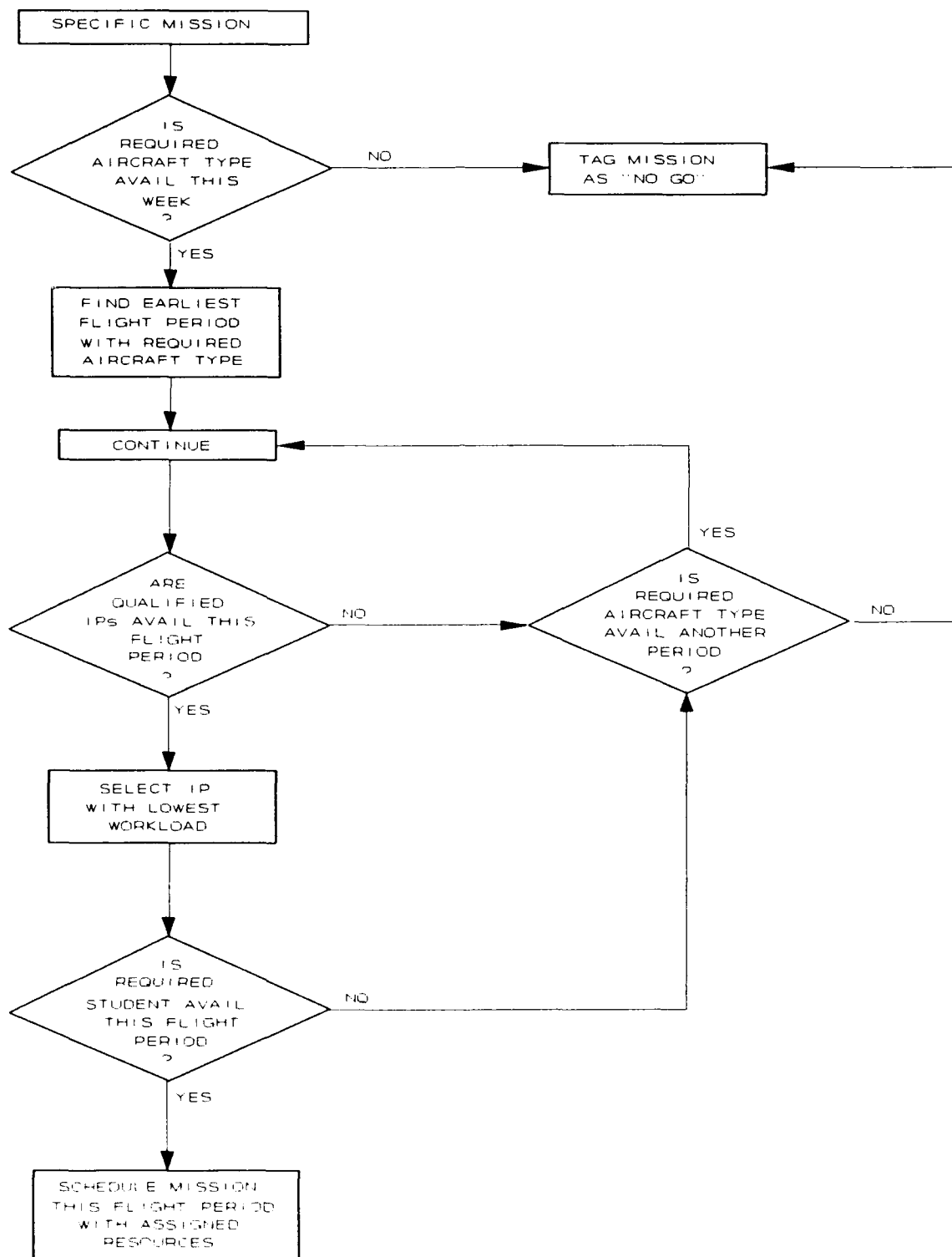


Figure 4.2 Resource Allocation Flow

and until a flight period is determined in which both aircraft and instructor are available, the flight period in which to check the student's availability is not known, and 2) students are usually available. In addition, if the given flight period is on an academic test day for the given student, the algorithm checks to see if the student has already been scheduled to fly another mission in the other flight period of that same day. If this is the case, the mission cannot be scheduled this flight period because on an academic test days, students are allowed to fly only one mission. As in the case of step 3b, if the student is not available in this specific period, the mission's ready date is shifted to one flight period later in the week and the algorithm returns to step 3a.

When a flight period in which all the required resources are available is found, the next step (**step 3d**) is to adjust all the resource levels to represent their usage in scheduling the specific mission in the given flight period. The algorithm then returns to step 2 to select the next mission for scheduling. When a scheduling attempt has been made for every mission, the algorithm goes to step 4.

Step 4 (Save the Best Schedule): At the completion of a given iteration (an iteration is complete when an attempt has been made to schedule every mission), the algorithm saves the best schedule developed thus far. The criterion for selecting the best schedule is based on 1) the number of missions scheduled and 2) the variance of instructor pilot workloads. The schedule

that produces the maximum number of scheduled missions is always saved. If two schedules result in having the same number of missions scheduled, the schedule with the smallest variance in instructor pilot workloads is saved. The variance of the instructor pilot workloads is calculated using the standard statistical formula for population variance.

Step 5 (Iteration Check): The entire process is repeated until the maximum number of scheduling iterations has been completed, or all the missions have been scheduled. Based on multiple trials using full-size TPS weekly flight scheduling scenarios, the algorithm rarely found a better schedule beyond the fifth iteration. Currently, the coded algorithm is set to perform a maximum of seven iterations. The algorithm does not guarantee a better schedule after each iteration. During test runs, better schedules were not always obtained from subsequent scheduling iterations. Therefore, unless an iteration produces a schedule in which all the missions are scheduled, the algorithm always completes the maximum number of iterations, even if subsequent iterations do not improve the schedule. Additional iterations will not be accomplished solely to even out instructor pilot workloads. Typically, since there are usually more missions eligible to be scheduled than there are aircraft available, all the missions cannot be scheduled, and therefore, the algorithm usually performs the maximum number of iterations before terminating.

Step 6 (Reset Original Resource Levels): If another scheduling iteration is to be performed, resource levels are re-initialized to their original levels.

Step 7 (Re-prioritization of Missions): After the resource levels are re-initialized, the missions are re-ranked based on information obtained from the previous developed schedules. Missions are re-ranked based on three parameters; 1) original mission ready dates, 2) number of qualified instructor pilots for each specific mission, and 3) the number of times each mission is not scheduled. The last two parameters are used to incorporate a measure of scheduling difficulty into the ranking. Missions that are more difficult to schedule due to limited flexibility in their resource requirements are given priority. For example, missions that have fewer qualified instructor pilots are likely to be more difficult to schedule. Furthermore, after several schedules have been developed, the number of times a specific mission was not scheduled gives the algorithm a measure of how difficult it is to schedule that particular mission. The algorithm uses a counter to keep track of the number of times each mission was not scheduled. Each time a mission is not scheduled, its counter is incremented by one. Thus, a high counter value represents a mission that is difficult to schedule. For missions in which a precedence relationship exists, if a follow-on mission was not scheduled for a given student, its corresponding precedence mission has one added to its counter.

Missions are re-prioritized based on the following rules; 1) missions not scheduled the most are given the highest priority, 2) if two missions are not scheduled the same number of times, the mission with the earlier ready date has priority, and 3) if there is still a tie, the mission with fewer qualified instructor pilots has priority. Table 4.1 presents an example of how 10 missions would be ranked for scheduling based on their corresponding parameter values. Once the missions have been re-prioritized, the algorithm returns to step 2 and starts to develop another schedule.

TABLE 4.1
EXAMPLE OF RE-PRIORITIZING MISSIONS

MISSION	READY DATE	NUMBER OF QUALIFIED INSTRUCTOR PILOTS	NUMBER OF TIMES MISSION NOT SCHEDULED	MISSION RANKING
A	01/03/92	3	0	5
B	01/03/92	2	0	4
C	01/03/92	1	1	2
D	01/07/92	3	0	7
E	01/08/92	3	0	10
F	01/08/92	2	0	8*
G	01/08/92	2	1	3
H	01/07/92	1	2	1
I	01/08/92	2	0	9*
J	01/07/92	2	0	6

* When there is a prioritization tie between missions (such as with missions F and I), priority is given to the mission listed first (F).

Figure 4.3 depicts an example of how the algorithm would progress through a very small flight scheduling problem. In this example, each of the three missions (each with a different student who is always available) to be scheduled requires the same aircraft type; however, their instructor pilot (IP) requirements differ. There are only two flight periods in this small example. In iteration 1, mission rankings are based only on ready dates. As the scheduling process progresses and missions are re-prioritized, the resources that are assigned to each mission vary based on the mission's scheduling priority (rank). As depicted in Figure 4.3, the second and third scheduling iterations provide no improvement to the schedule. However, on the fourth scheduling iteration, every mission is scheduled and instructor pilot workloads are equalized. Notice, that if the ready dates were all the same, the schedule that was produced in the fourth iteration would have been produced in the second iteration.

4.3 Coded Algorithm Development and Description

The heuristic scheduling algorithm was coded and compiled using *Microsoft FORTRAN 5.0*. In developing a weekly flight schedule, the program accesses 22 different external data files which contain the flight scheduling information for the given flight week. A further description of and operating instructions for the coded algorithm are provided in Appendix C. The source code for the heuristic algorithm is provided in Appendix D.

MISSION (MSN) INFORMATION

MSN	READY DATE	REQUIRED IP	# QUAL IP
A	01/01/92	1, 2, OR 3	3
B	01/02/92	1 OR 2	2
C	01/03/92	1	1

RESOURCE AVAILABILITY

	PERIOD 1	PERIOD 2
IP #1	YES	NO
IP #2	YES	YES
IP #3	NO	YES
AIRCRAFT	1	2

ITERATION #1

MSN	# TIMES UNSCHED	RANK	SCHED PERIOD	ASSIGNED IP
A	0	1	1	1
B	0	2	2	2
C	0	3	-	-

NUMBER OF MISSIONS SCHEDULED 2
 VARIANCE OF IP WORKLOAD 0.47
 SCHEDULE SAVED

ITERATION #2

MSN	# TIMES UNSCHED	RANK	SCHED PERIOD	ASSIGNED IP
A	0	2	2	2
B	0	3	-	-
C	1	1	1	1

NUMBER OF MISSIONS SCHEDULED 2
 VARIANCE OF IP WORKLOAD 0.47
 SCHEDULE NOT SAVED

ITERATION #3

MSN	# TIMES UNSCHED	RANK	SCHED PERIOD	ASSIGNED IP
A	0	3	2	2
B	1	1	1	1
C	1	2	-	-

NUMBER OF MISSIONS SCHEDULED 2
 VARIANCE OF IP WORKLOAD 0.47
 SCHEDULE NOT SAVED

ITERATION #4

MSN	# TIMES UNSCHED	RANK	SCHED PERIOD	ASSIGNED IP
A	0	3	2	3
B	1	2	2	2
C	2	1	1	1

NUMBER OF MISSIONS SCHEDULED 3
 VARIANCE OF IP WORKLOAD 0.0
 SCHEDULE SAVED
 ALGORITHM COMPLETE

Figure 4.3 Example of Algorithm Progression

4.3.1 Algorithm Testing. The coded heuristic algorithm was tested using both representative flight scheduling data and operational TPS flight scheduling data. Approximately 20 tests were conducted using flight scheduling data that was representative of typical TPS flight weeks. To a large extent, these initial tests were more developmental than operational, meaning that they were used more to develop the algorithm to its present state than test the effectiveness of the heuristic. Results from these tests were used to adjust and refine the heuristic algorithm in order to obtain better results (schedules). These tests were also used to verify that the computer code accurately performed the steps of the heuristic algorithm. Listings of the flight scheduling data files and the resulting flight schedule output file for a sample full-size TPS weekly flight scheduling problem are provided in Appendix B.

Once the developmental tests were complete, the coded heuristic algorithm was operationally tested at the TPS using real flight scheduling data. The quality of the resulting weekly flight schedule was judged good by the TPS scheduling staff; however, the manner in which the flight scheduling data was updated for use by the algorithm was considered cumbersome. This deficiency has since been corrected by editing and exporting files using the spreadsheet software *QUATTRO PRO*.

4.3.2 Testing Limitations. Although the quality of the weekly flight schedule produced in the operational test was rated

good by the TPS staff, this test was limited for three primary reasons. 1) Due to the limited availability of actual flight scheduling data and scenarios, only one real world weekly flight schedule was produced. 2) Flight schedules for an entire flight week have never been developed at the TPS; therefore, there was no historical data in which the weekly flight schedule produced by the algorithm could be objectively compared against. 3) Full-scale TPS weekly flight scheduling problems have never been solved using techniques that guarantee optimal solutions. Therefore, the ability of this heuristic algorithm to guarantee schedules within a measure of optimality could not be determined. To give a limited comparison of the scheduling effectiveness of the heuristic algorithm, a schedule for the small example problem solved in Chapter 3 using mixed integer programming is also developed using the coded heuristic algorithm (reference section 4.3.4). Unfortunately, the same effectiveness as demonstrated in the small scheduling example problem cannot be guaranteed in a full-scale application. As a result of the three limitations cited above, the analytical effectiveness of the heuristic algorithm was only subjectively rated.

4.3.2 Coded Algorithm Output. The output from the coded algorithm provides a variety of information to the TPS scheduling staff. Most important is the schedule itself. For each flight period, a listing of each mission scheduled including mission description, instructor pilot, student, aircraft type, and

students class is provided. In addition, at the request of the TPS scheduling staff, aircraft and instructor pilot resources that are still available in each flight period are also displayed. Such information makes it easier to further manipulate the schedule. The schedule produced by the algorithm is just an initial schedule for the flight week. It may be changed for reasons such as weather, illness, and aircraft in maintenance. Aside from the actual schedule, the algorithm also identifies missions not scheduled during the flight week and missions that violated quality of training completion deadlines. Missions that violate quality of training completion deadlines may be scheduled; however, flagging such missions allows the TPS scheduling staff to make the decision as to either waive the completion deadline requirement or to schedule make-up training.

4.3.4 Example Problem Solution. A schedule for the small example flight scheduling problem formulated and solved in Chapter 3 was obtained using the coded heuristic algorithm implemented on a 16 MHz 80286-based desktop computer with 1 megabyte of memory. To obtain a solution, the problem required approximately 10 CPU seconds. In terms of the objective function for the mixed integer programming scheduling approach presented in Chapter 3, the schedule produced by the heuristic algorithm was not optimal. Its corresponding objective function value would have been only 16.1 (versus the optimal value of 17). Although, every training mission was successfully scheduled

during the week, one instructor pilot (#2) exceeded the workload goal by one work unit. Table 4.2 depicts the corresponding schedule. Algorithm input and output files for this example problem are provided in Appendix A. By a review of the schedule presented in Table 4.2, it can be seen that all resource and operational restrictions have been satisfied.

TABLE 4.2
EXAMPLE PROBLEM SOLUTION SCHEDULE
(Heuristic Algorithm Approach)

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
	s m a i	s m a i	s m a i	s m a i	s m a i
PERIOD 1	3 5 T-38 3 4 5 T-38 1 6 6 F-4 2	2 1 C-23 1 3 3 T-38 2	1 2 C-23 1 5 3 T-38 3	4 4 T-38 4 5 4 T-38 4 6 7 F-4 2	7 7 F-4 2
PERIOD 2	1 1 C-23 2 5 5 T-38 3	4 3 T-38 1 7 6 F-4 2	3 4 T-38 4	2 2 C-23 1	

(s = student, m = mission type, a = aircraft type, i = instructor pilot)

4.4 Full-Scale Application

The most positive aspect of the heuristic algorithm approach is its ability to produce acceptable weekly flight schedules in relatively little time. The computerized algorithm has been used to obtain schedules for multiple representative full-scale TPS weekly flight scheduling problems. A full-scale problem is defined as including 50 students, 25 instructor pilots, 10 aircraft types, and more than 60 missions. Based on the results of these tests, on the average, good schedules were obtained in

approximately 2 minutes of CPU time on a 16 MHz 80286-based desktop computer with 1 megabyte of memory. Estimating that it takes slightly under one hour to update the scheduling data files for a given flight week, good weekly flight schedules can be obtained in approximately one hour, using the computerized heuristic algorithm.

5. Conclusions and Suggestions

5.1 Conclusions

Due to the large quantities of scheduling data that must be processed, flight schedules representing an entire flight week cannot be developed at the USAF Test Pilot School (TPS) unless the process is automated. Currently, the TPS manually develops flight schedules on a daily basis. It usually requires the TPS flight scheduler one to two hours to manually develop a daily flight schedule. Although a daily flight scheduling approach is more suitable for handling last minute changes due to unforeseen events, its lack of foresight often leads to bottlenecks and unbalanced resource utilization, both of which could have been avoided if the entire flight week were scheduled in a single process. The objective of this thesis was to investigate and develop a technique to automate the flight scheduling process at the TPS in order to improve the flow of missions scheduled throughout the week and reduce the amount of TPS staff time dedicated to flight scheduling. In addition, by posting a tentative flight schedule for the entire week, communication between the TPS scheduling staff and TPS personnel (students and instructors) can be improved.

The application of the heuristic algorithm presented in this thesis demonstrated that reasonable flight schedules representing an entire flight week can be developed in a short amount of time.

Weekly flight schedules were developed for multiple flight scheduling problems having the size and characteristics of a typical TPS flight week. Tests using these different flight scheduling scenarios were used to improve, verify, and validate the heuristic algorithm.

In addition, an operational test of the heuristic algorithm was conducted at the TPS using actual missions, mission resource requirements, and weekly resource (student, instructor pilot, and aircraft) availability. The quality of the schedule produced in this sample was acceptable to the TPS scheduling staff. Unfortunately, due to the limited availability of actual flight scheduling data and scenarios, only one real-world TPS weekly flight schedule was produced.

Based on the results of actual and representative weekly flight scheduling scenarios, good weekly flight schedules can be produced by the heuristic algorithm in less than ten minutes of computer time. Estimating that it takes approximately one hour to update the data files containing the scheduling information for a given flight week, an acceptable flight schedule for an entire week can be developed in about the same amount of time it currently requires the TPS flight scheduler to manually develop a daily flight schedule. Furthermore, once an initial weekly flight schedule is obtained, it provides a good starting point in which manual adjustments can be made by the flight scheduler in reaction to unpredictable events (weather, aircraft maintenance, illness, etc.).

Aside from the heuristic approach developed and implemented in this thesis, a mixed integer programming (MIP) approach for solving the TPS weekly flight scheduling problem was also investigated. Although a MIP approach may have been able to produce "better" weekly flight schedules, such an approach was impractical due to the resulting large problem size.

5.2 Suggestions for Further Work

Even though the algorithm has demonstrated its ability to produce good weekly flight schedules in a short period of time, further investigation should be conducted to improve the presented algorithm or to develop another algorithm in order to further improve the quality of weekly flight schedules. One possible enhancement might be an interchange scheme to improve the current heuristic algorithm's ability to level workloads of the instructor pilots. Such an enhancement would search through the schedule to check if missions supported by over-worked instructor pilots can be reassigned to under-worked instructor pilots. In addition, an approach for handling special case missions (such as missions that must be supported by a chase or target aircraft) by the current scheduling algorithm should also be developed. In a MIP approach, a requirement of this type can easily be handled with the addition of a constraint; however, the current scheduling approach requires the flight scheduler to manually handle special mission requirements.

Notwithstanding the above possible enhancements, the current heuristic algorithm appears to be capable of producing reasonable weekly flight schedules in a practical amount of time. The current coded heuristic algorithm should be further tested by the TPS scheduling staff to identify any additional desired enhancements or possible limitations and to fully assess its potential in automating the flight scheduling process at the TPS.

Appendix A:
Heuristic Algorithm Applied to the Example Problem

Student Pilot Availability Data

	MON1	MON2	TUE1	TUE2	WED1	WED2	THR1	THR2	FRI1	FRI2
ST #1	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ST #2	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ST #3	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ST #4	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ST #5	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ST #6	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ST #7	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
END										

Instructor Pilot Availability Data

	MON1	MON2	TUE1	TUE2	WED1	WED2	THR1	THR2	FRI1	FRI2	WORK
IP #1	Y	N	Y	Y	Y	N	Y	Y	Y	Y	0
IP #2	Y	Y	Y	Y	N	Y	Y	Y	Y	N	0
IP #3	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	0
END											

Aircraft Availability Data

	MON1	MON2	TUE1	TUE2	WED1	WED2	THR1	THR2	FRI1	FRI2
F-4	1	0	0	1	1	1	1	0	1	0
T-38	2	1	1	1	1	1	2	1	1	1
C-23	0	1	1	1	1	0	0	1	0	1
END										

Training Mission Data Sets

	C-23 CF	C-23 PERF DEMO	T-38 RANGE DEMO	
ST #1		10191	-10191	0
ST #2		10191	-10191	0
ST #3		0	0	10191
ST #4		0	0	10191
ST #5		0	0	10191
ST #6		0	0	0
ST #7		0	0	0
END				

	F-4 STRUCTURES	T-38 LS DEMO	T-38 LONG STAT DATA	
ST #1		0	0	0
ST #2		0	0	0
ST #3		0	10191	-10191
ST #4		0	10191	-10191
ST #5		0	10191	-10191
ST #6		10191	0	0
ST #7		10191	0	0
END				

	F-4 PROPULSION	EXTRA	EXTRA	
ST #1		0	0	0
ST #2		0	0	0
ST #3		0	0	0
ST #4		0	0	0
ST #5		0	0	0
ST #6		10491	0	0
ST #7		10491	0	0
END				

Instructor Pilot Qualification Data Sets

T-38	IP	TPS	L/D	CHSE	TGT	FE	FCF	SOF	EXTRA
IP #1	X	X	X	X	X			X	
IP #2	X	X	X	X	X			X	
IP #3	X	X	X	X					
END									

F-4	IP	TPS	STRC	PROP	AS/S	TGT	FE	FCF	SOF
IP #2	X	X	X	X		X			X
END									

C-23	IP	TPS	FE	FCF	EXTRA	EXTRA	EXTRA	EXTRA	EXTRA
IP #1	X	X	X						
IP #2	X	X							
END									

MSN_TYPE	AC_TYPE	QUAL
T-38 LS DEMO	T-38	TPS
T-38 LS DATA	T-38	N/A
T-38 RANGE DEMO	T-38	TPS
F-4 PROPULSION	F-4	PROP
F-4 STRUCTURES	F-4	STRC
C-23 CHECK FLIGHT	C-23	TPS
C-23 PERF DEMO	C-23	TPS
END		

Algorithm Output Schedule

MONDAY 1st Flight Period:

```
*****
MISSION          AIRCRAFT      INSTRUCTOR    STUDENT      CLASS
*****
T-38 RANGE DEMO   T-38          IP #3         ST #3        B
T-38 RANGE DEMO   T-38          IP #1         ST #4        B
F-4 STRUCTURES    F-4           IP #2         ST #6        B
```

AVAILABLE RESOURCES

```
*****
INSTRUCTOR        AIRCRAFT      AMOUNT
*****
```

MONDAY 2nd Flight Period:

```
*****
MISSION          AIRCRAFT      INSTRUCTOR    STUDENT      CLASS
*****
C-23 CF          C-23          IP #2         ST #1        B
T-38 RANGE DEMO   T-38          IP #3         ST #5        B
```

AVAILABLE RESOURCES

```
*****
INSTRUCTOR        AIRCRAFT      AMOUNT
*****
```

TUESDAY 1st Flight Period:

```
*****
MISSION          AIRCRAFT      INSTRUCTOR    STUDENT      CLASS
*****
C-23 CF          C-23          IP #1         ST #2        B
T-38 LS DEMO     T-38          IP #2         ST #3        B
```

AVAILABLE RESOURCES

```
*****
INSTRUCTOR        AIRCRAFT      AMOUNT
*****
```

TUESDAY 2nd Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
T-38 LS DEMO	T-38	IP #1	ST #4	B
F-4 STRUCTURES	F-4	IP #2	ST #7	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
IP #3	C-23	1

WEDNESDAY 1st Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
C-23 PERF DEMO	C-23	IP #1	ST #1	B
T-38 LS DEMO	T-38	IP #3	ST #5	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
	F-4	1

WEDNESDAY 2nd Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
T-38 LONG STAT DATA	T-38	N/A	ST #3	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
IP #2	F-4	1
IP #3		

THURSDAY 1st Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
T-38 LONG STAT DATA	T-38	N/A	ST #4	B
T-38 LONG STAT DATA	T-38	N/A	ST #5	B
F-4 PROPULSION	F-4	IP #2	ST #6	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
IP #1		
IP #3		

THURSDAY 2nd Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
C-23 PERF DEMO	C-23	IP #1	ST #2	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
IP #2	T-38	1
IP #3		

FRIDAY 1st Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
F-4 PROPULSION	F-4	IP #2	ST #7	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
IP #1	T-38	1
IP #3		

FRIDAY 2nd Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT

IP #1	T-38	1
IP #3	C-23	1

MISSIONS NOT SCHEDULED THIS WEEK:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS

MISSIONS THAT VIOLATE QOT DEADLINES:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS

Appendix B:

Heuristic Algorithm Applied to a Full-Size Problem

Student Pilot Availability Data (Class A)

	MON1	MON2	TUE1	TUE2	WED1	WED2	THR1	THR2	FRI1	FRI2
BAKKEN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
BALTRUSAIT	Y	Y	Y	Y	Y	Y	N	N	N	N
BLOOMFIELD	Y	Y	N	Y	Y	Y	Y	Y	Y	Y
CAREY	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
GRIFFITH	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HUNNEL	Y	Y	Y	Y	N	Y	Y	Y	Y	Y
SHORT	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SMITH	Y	Y	N	Y	Y	Y	Y	Y	Y	Y
STURKOW	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
TRAVEN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
DENOFRIO	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
DILLION	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
MURRAY	Y	Y	N	Y	N	Y	Y	Y	Y	Y
PAYNE	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SHAW	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
STAPP	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
WILHEIM	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
END										

Student Pilot Availability Data (Class B)

	MON1	MON2	TUE1	TUE2	WED1	WED2	THR1	THR2	FRI1	FRI2
ASH	Y	Y	Y	Y	Y	N	Y	Y	N	Y
FAIRBAIRN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
GRAY	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HOWELL	N	Y	Y	Y	Y	N	Y	Y	Y	Y
LEE	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
LUND	Y	Y	Y	Y	Y	Y	Y	Y	N	Y
MELROY	Y	Y	Y	N	Y	Y	Y	Y	Y	Y
ROBINSON	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
STERNBERG	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ZEHR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
BARAK	Y	Y	N	Y	Y	Y	Y	Y	Y	Y
DURON	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
KELLER	Y	Y	Y	Y	Y	Y	Y	Y	N	N
NICHOLS	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SHANKAR	Y	Y	Y	Y	Y	N	Y	Y	Y	Y
STAMBAUGH	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
VERDERAME	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
END										

Instructor Pilot Availability Data

	MON1	MON2	TUE1	TUE2	WED1	WED2	THR1	THR2	FRI1	FRI2	WORK
BENDORF	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
BONASSO	Y	N	Y	N	Y	Y	Y	Y	N	N	0
BROWN	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
CARLSON	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
DENESIK	Y	Y	Y	Y	Y	N	Y	Y	Y	N	0
GOGAN	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	0
GREEN	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	0
GRUNNALD	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
GARDNER	Y	Y	N	Y	N	Y	Y	N	Y	Y	0
HORTON	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
HUNTER	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
IMIG	Y	Y	Y	Y	N	N	N	N	N	N	0
KANA	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
LUEDKE	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
LUTZ	Y	Y	N	N	Y	Y	Y	Y	Y	Y	0
MARKOVICH	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
MOSER	N	Y	Y	Y	Y	N	Y	Y	Y	Y	0
NELSON	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	0
STOFFERAHN	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	0
STONE	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	0
E WILSON	Y	Y	Y	Y	Y	Y	N	Y	N	Y	0
R WILSON	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
WOOD	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	0
MARTIN	N	N	N	N	N	N	N	N	N	N	0
MOSS	N	N	Y	N	N	Y	N	N	Y	N	0
SMOLKA	N	Y	N	N	N	N	N	Y	N	Y	0
SOBCZAK	N	N	N	N	N	N	N	Y	N	N	0
END											

Aircraft Availability Data

	MON1	MON2	TUE1	TUE2	WED1	WED2	THR1	THR2	FRI1	FRI2
A-7	1	1	0	1	0	1	1	1	1	0
F-4	1	0	1	1	1	0	1	0	0	1
T-38	3	5	3	3	3	5	2	3	4	2
C-23	0	1	0	1	1	0	2	1	0	0
F-16	0	0	0	0	2	1	0	1	1	0
KC-135	0	0	0	0	0	0	0	0	0	0
C-130	0	0	0	0	0	0	0	0	0	0
GLIDER	0	0	0	0	0	0	0	0	0	0
C-141	1	0	1	0	1	0	1	0	1	0
F-15	0	0	0	0	0	0	0	0	0	0
A-37	1	0	2	0	2	1	1	0	0	1
ASTTA	0	0	0	0	0	0	0	0	0	0
VSS	0	0	0	0	0	0	0	0	0	0
END										

Training Mission Data Sets (Class A)

	T-38 CF	F-4 CF	T-38 LEVEL ACCEL DEM
BAKKEN	11691	0	-11691
BALTRUSAIT	11691	0	-11691
BLOOMFIELD	11691	0	-11691
CAREY	11691	0	-11691
GRIFFITH	11691	0	-11691
HUNNEL	11691	0	-11691
SHORT	11691	0	-11691
SMITH	0	11691	0
STURKOW	0	11691	0
TRAVEN	0	11691	0
DENOFRIO	0	11691	0
DILLION	11691	0	-11691
MURRAY	11691	0	-11691
PAYNE	11691	0	-11691
SHAW	11691	0	-11691
STAPP	0	11891	0
WILHEIM	0	11891	0
END			

	C-23 PERF DEMO	C-141 MULTI ENG DEMO	EXTRA
BAKKEN	0	0	0
BALTRUSAIT	0	0	0
BLOOMFIELD	0	0	0
CAREY	0	0	0
GRIFFITH	0	0	0
HUNNEL	0	0	0
SHORT	0	0	0
SMITH	10791	0	0
STURKOW	10791	0	0
TRAVEN	10891	0	0
DENOFRIO	10891	0	0
DILLION	0	0	0
MURRAY	0	0	0
PAYNE	0	0	0
SHAW	0	0	0
STAPP	0	11691	0
WILHEIM	0	11691	0
END			

Training Mission Data Sets (Class B)

	A-7 LAT DIR DATA	T-38 LOW LIFT/DRAG	T-38 MAN FLT DATA
ASH	0	11691	0
FAIRBAIRN	0	11691	10191
ROBINSON	0	11691	0
LEE	0	11691	10291
STERNBERG	0	11691	10191
HOWELL	0	11691	10391
LUND	11791	0	0
ZEHR	11791	0	0
MELROY	11791	0	0
GRAY	11891	0	0
SHANKAR	0	11791	0
DURON	0	11791	0
NICHOLS	0	11791	0
KELLER	0	11791	0
BARAK	0	11791	0
STAMBAUGH	0	11791	0
VERDERAME	0	11791	0
END			

	A-37 STALL DEMO	F-16 FTT	EXTRA
ASH	11791	0	0
FAIRBAIRN	0	0	0
ROBINSON	11791	0	0
LEE	11791	0	0
STERNBERG	11791	0	0
HOWELL	0	0	0
LUND	0	0	0
ZEHR	0	0	0
MELROY	0	0	0
GRAY	0	0	0
SHANKAR	0	11691	0
DURON	0	11691	0
NICHOLS	0	11691	0
KELLER	11791	11691	0
BARAK	11791	0	0
STAMBAUGH	11791	0	0
VERDERAME	0	0	0
END			

Instructor Pilot Qualification Data Sets

A-7	IP	TPS	DEP	TGT	WPN	FCF	FE	EXTRA	EXTRA
BENJAMIN	X	X	X						
STONE	X	X	X		X	X			
E WILSON	X	X	X						
R WILSON	X	X	X						
END									

C-23	IP	TPS	FE	FCF	EXTRA	EXTRA	EXTRA	EXTRA	EXTRA
BENDORF	X	X	X						
HUNTER	X	X							
KANA	X	X	X						
MARKOVICH	X	X	X						
END									

F-15	IP	TPS	FE	FCF	SOF	EXTRA	EXTRA	EXTRA	EXTRA
DENESIK					X				
HUNTER									
IMIG									
WOOD	X		X		X				
END									

F-16	IP	TPS	TGT	FE	FCF	SOF	EXTRA	EXTRA	EXTRA
BONASSO	X	X	X	X	X	X			
BROWN	X	X	X			X			
HORTON	X	X			X	X			
LUEDKE									
NELSON	X	X		X		X			
STOFFERAHN						X			
END									

F-4	IP	TPS	STRC	PROP	AS/S	TGT	FE	FCF	SOF
BONASSO	X	X	X	X		X			X
BROWN	X	X		X	X	X		X	X
MOSER	X	X	X	X	X				X
SOBCZAK	X	X	X	X	X				
STOFFERAHN	X	X	X	X	X		X		X
E WILSON	X	X	X		X				X
END									

GLIDER	MP	IP	SPIN	F/Q	TOW	EXTRA	EXTRA	EXTRA	EXTRA
ALDRICH	X	X	X	X					
BENJAMIN	X	X	X	X					
HUNTER	X	X	X	X					
MARKOVICH	X	X	X	X					
STONE	X								
R WILSON	X	X							
WOOD	X								
END									

T-38	IP	TPS	L/D	CHSE	TGT	FE	FCF	SOF	EXTRA
BENDORF	X	X	X	X	X			X	
BENJAMIN	X	X	X	X	X			X	
CARLSON	X	X	X	X					
DENESIK		X						X	
GARDNER	X	X	X						
GOGAN	X	X	X		X	X	X		
GRUNWALD	X	X	X	X	X			X	
IMIG	X	X	X	X	X				
KANA	X	X	X					X	
LUTZ	X	X	X	X	X	X		X	
MARTIN	X	X	X	X	X				
SMOLKA	X	X	X	X	X				
STONE	X	X	X	X	X		X		
WOOD	X	X	X	X	X			X	
END									

A-37	IP	TPS	SPIN	TGT	WPN	BAL	FE	FCF	SOF
CARLSON	X	X	X						
GREEN									X
LUEDKE									
MOSER	X	X	X						X
MOSS	X	X	X						
NELSON									X
R WILSON	X	X	X		X	X	X		
END									

C-141	IP	TPS	FE	FCF	SOF	EXTRA	EXTRA	EXTRA	EXTRA
GRUNNOLD	X	X							
GREEN	X	X							
END									

Mission Requirement Data

MSN TYPE	AC TYPE	QUAL
T-38 CF	T-38	IP
T-38 FTE/N FAM	T-38	TPS
T-38 FTE/N PERF DEMO	T-38	TPS
T-38 LEVEL ACCEL DEM	T-38	TPS
T-38 LA/CC PRAC	T-38	TPS
T-38 LEVEL ACCEL DAT	T-38	N/A
T-38 PACE LEAD	T-38	TPS
T-38 TFB/PACE	T-38	TPS
T-38 TURN PRAC	T-38	TPS
T-38 TURN DATA	T-38	N/A
T-38 MID-PHASE PRAC	T-38	TPS
T-38 CRUISE DATA	T-38	N/A
T-38 CHECKCLIMB DATA	T-38	N/A
T-38 RANGE DATA	T-38	N/A
T-38 LOW LIFT/DRAG	T-38	TPS
T-38 LS DEMO	T-38	TPS
T-38 LONG STAT DATA	T-38	N/A
T-38 MF/OPS HND	T-38	TPS
T-38 FORM	T-38	TPS
T-38 MAN FLT DATA	T-38	N/A
T-38 LAT DIR DATA	T-38	N/A
T-38 FTE/N FQ DEMO	T-38	TPS
T-38 DYN DATA	T-38	N/A
T-38 STALL DATA	T-38	N/A
T-38 OPS HNDLING DAT	T-38	N/A
T-38 TARGETS	T-38	TGT
T-38 SPIN CHASE	T-38	SPIN
T-38 FTE/N FQ CHECK	T-38	TPS
T-38 TGT W/TACAN	T-38	TPS
T-38 FTT	T-38	TPS
F-4 FTE/N FAM	F-4	TPS
F-4 CF	F-4	IP
F-4 LA/CC PRAC	F-4	TPS
F-4 LEVEL ACCEL DATA	F-4	N/A
F-4 TFB/PACE	F-4	TPS
F-4 TURN PRAC	F-4	TPS
F-4 TURN DATA	F-4	N/A
F-4 MID-PHASE CHECK	F-4	TPS
F-4 CRUISE DATA	F-4	N/A
F-4 CHECKCLIMB DATA	F-4	N/A
F-4 RANGE DATA	F-4	N/A
F-4 PROPULSION	F-4	PROP
F-4 LONG STAT DATA	F-4	N/A
F-4 MAN FLT DATA	F-4	N/A
F-4 LAT DIR DATA	F-4	N/A
F-4 DYN DATA	F-4	N/A
F-4 STALL DATA	F-4	N/A
F-4 OPS HNDLING DATA	F-4	N/A
F-4 TARGETS	F-4	TGT

F-4 PILOT FQ CHECK	F-4	TPS
F-4 FTE/N FQ CHECK	F-4	TPS
F-4 ASY STO/CAP COMP	F-4	TPS
F-4 STRUCTURES	F-4	STRC
F-4 FTT	F-4	TPS
KC-135 CRUISE/CC DAT	KC-135	N/A
KC-135 FAM	KC-135	N/A
KC-135 LA/CC PRC/DAT	KC-135	N/A
KC-135 RANGE/CC DATA	KC-135	N/A
KC-135 FTT	KC-135	N/A
C-23 FTE/N LA/TURN	C-23	TPS
C-23 FTE/N FAM	C-23	TPS
C-23 PERF DEMO	C-23	TPS
C-23 FTE/N L-S DEMO	C-23	TPS
C-23 MF/OPS HANDLING	C-23	TPS
C-23 FTE/N LAT DIR	C-23	TPS
C-23 CF	C-23	IP
C-23 FTT	C-23	TPS
A-7 CF	A-7	IP
A-7 IP CHASE	A-7	TPS
A-7 LONG STAT DATA	A-7	N/A
A-7 MAN FLT DATA	A-7	N/A
A-7 QUAL/FAM	A-7	TPS
A-7 DEPARTURE	A-7	TPS
A-7 LAT DIR DATA	A-7	N/A
A-7 DYN DATA	A-7	N/A
A-7 STALL DATA	A-7	N/A
A-7 OPS HNDLING DATA	A-7	N/A
A-7 TARGETS	A-7	TGT
A-7 FTT	A-7	TPS
A-7 SYSTEMS EVAL	A-7	TPS
C-141 CF	C-141	IP
C-141 LONG STAT DATA	C-141	TPS
C-141 MAN FLT DATA	C-141	TPS
C-141 LAT DIR DATA	C-141	TPS
C-141 MULTI ENG DEMO	C-141	TPS
C-141 ENGINE OUT	C-141	TPS
C-141 DYNAMICS DATA	C-141	TPS
C-141 STALL DATA	C-141	TPS
C-141 OPS HANDLING	C-141	TPS
C-141 FQ CHECK	C-141	TPS
C-141 FTT	C-141	TPS
F-16 CF	F-16	IP
F-16 CHECKCLIMB DEMO	F-16	TPS
F-16 TURN DEMO	F-16	TPS
F-16 FTT	F-16	TPS
F-16 SYSTEMS EVAL	F-16	TPS
HI LIFT/DRAG GLIDER	GLIDER	TPS
GLIDER FQ DEMO	GLIDER	TPS
GLIDER SPIN	GLIDER	SPIN
VSS 1	VSS	N/A
VSS 2	VSS	N/A

VSS FTE/N	VSS	N/A
VSS FCS PROJECT	VSS	N/A
VSS 3	VSS	N/A
A-37 CF	A-37	IP
A-37 LAT DIR DEMO	A-37	TPS
A-37 STALL DEMO	A-37	TPS
A-37 QUAL DEMO	A-37	TPS
A-37 SPIN 1	A-37	SPIN
A-37 SPIN 2	A-37	SPIN
A-37 SPIN FTE/N	A-37	SPIN
A-37 CONT WPN DEL	A-37	WPN
A-37 BALLISTICS	A-37	BAL
A-37 FTT	A-37	TPS
ASTTA DAY	ASTTA	N/A
ASTTA NIGHT	ASTTA	N/A
SYSTEMS QUALS (A)	NNN	N/A
SYSTEMS QUALS (B)	NNN	N/A
QUAL PROJECT FLIGHTS	NNN	N/A
SL QUAL FLIGHTS	NNN	N/A
FTN SYSTEMS EVAL	NNN	N/A
FTN SYSTEMS EVAL TGT	NNN	N/A
TMP FLIGHTS	NNN	N/A
F-15 CF	F-15	IP
F-15 DYN DEMO	F-15	TPS
F-15 FTT	F-15	TPS
C-130 PERF FINAL CHK	C-130	N/A
END	N/A	N/A

Algorithm Output Schedule and Information

MONDAY 1st Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
F-4 CF	F-4	BONASSO	SMITH	A
C-141 MULTI ENG DEMO	C-141	GRUNNARD	STAPP	A
T-38 MAN FLT DATA	T-38	N/A	FAIRBAIRN	B
T-38 MAN FLT DATA	T-38	N/A	LEE	B
T-38 MAN FLT DATA	T-38	N/A	STERNBERG	B
A-7 LAT DIR DATA	A-7	N/A	LUND	B
A-37 STALL DEMO	A-37	R WILSON	ASH	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
BROWN		
CARLSON		
DENESIK		
GOGAN		
GREEN		
GARDNER		
HORTON		
HUNTER		
IMIG		
KANA		
LUEDKE		
LUTZ		
MARKOVICH		
NELSON		
STOFFERAHN		
STONE		
E WILSON		
WOOD		

MONDAY 2nd Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
T-38 CF	T-38	CARLSON	BAKKEN	A
T-38 CF	T-38	GARDNER	BALTRUSAIT	A
T-38 CF	T-38	GOGAN	BLOOMFIELD	A
T-38 CF	T-38	IMIG	CAREY	A
C-23 PERF DEMO	C-23	BENDORF	SMITH	A
T-38 MAN FLT DATA	T-38	N/A	HOWELL	B
A-7 LAT DIR DATA	A-7	N/A	ZEHR	B

AVAILABLE RESOURCES

```

*****
INSTRUCTOR                AIRCRAFT                AMOUNT
*****
BROWN
DENESIK
GREEN
HORTON
HUNTER
KANA
LUEDKE
LUTZ
MARKOVICH
MOSER
NELSON
STOFFERAHN
STONE
E WILSON
R WILSON
WOOD
SMOLKA
  
```

TUESDAY 1st Flight Period:

```

*****
MISSION                AIRCRAFT                INSTRUCTOR        STUDENT            CLASS
*****
T-38 CF                T-38                WOOD                GRIFFITH           A
T-38 CF                T-38                BENDORF            HUNNEL             A
T-38 CF                T-38                CARLSON            SHORT              A
F-4 CF                F-4                BROWN              STURKOW            A
C-141 MULTI ENG DEMO   C-141              GRUNNALD           WILHEIM            A
A-37 STALL DEMO        A-37                MOSS                ROBINSON           B
A-37 STALL DEMO        A-37                MOSER               LEE                B
  
```

AVAILABLE RESOURCES

```

*****
INSTRUCTOR                AIRCRAFT                AMOUNT
*****
BONASSO
DENESIK
GOGAN
HORTON
HUNTER
IMIG
KANA
LUEDKE
MARKOVICH
NELSON
STOFFERAHN
  
```

TUESDAY 2nd Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
F-4 CF	F-4	MOSER	TRAVEN	A
T-38 CF	T-38	STONE	DILLION	A
T-38 CF	T-38	GARDNER	MURRAY	A
T-38 CF	T-38	GOGAN	PAYNE	A
C-23 PERF DEMO	C-23	HUNTER	STURKOW	A
A-7 LAT DIR DATA	A-7	N/A	GRAY	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
BENDORF		
BROWN		
CARLSON		
DENESIK		
GREEN		
HORTON		
IMIG		
KANA		
LUEDKE		
MARKOVICH		
NELSON		
STOFFERAHN		
E WILSON		
R WILSON		
WOOD		

WEDNESDAY 1st Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
F-4 CF	F-4	E WILSON	DENOFRIO	A
T-38 CF	T-38	LUTZ	SHAW	A
C-23 PERF DEMO	C-23	KANA	TRAVEN	A
T-38 LOW LIFT/DRAG	T-38	DENESIK	ASH	B
T-38 LOW LIFT/DRAG	T-38	STONE	FAIRBAIRN	B
A-37 STALL DEMO	A-37	R WILSON	STERNBERG	B
F-16 FTT	F-16	HORTON	SHANKAR	B
F-16 FTT	F-16	NELSON	DURON	B
A-37 STALL DEMO	A-37	MOSER	KELLER	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
BENDORF	C-141	1
BONASSO		
BROWN		
CARLSON		
GREEN		
GRUNNALD		
HUNTER		
LUEDKE		
MARKOVICH		
WOOD		

WEDNESDAY 2nd Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
T-38 LOW LIFT/DRAG	T-38	KANA	ROBINSON	B
T-38 LOW LIFT/DRAG	T-38	LUTZ	LEE	B
T-38 LOW LIFT/DRAG	T-38	WOOD	STERNBERG	B
A-7 LAT DIR DATA	A-7	N/A	MELROY	B
T-38 LOW LIFT/DRAG	T-38	CARLSON	DURON	B
T-38 LOW LIFT/DRAG	T-38	GARDNER	KELLER	B
F-16 FTT	F-16	BONASSO	NICHOLS	B
A-37 STALL DEMO	A-37	MOSS	BARAK	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
BENDORF		
BROWN		
GOGAN		
GREEN		
GRUNNALD		
HORTON		
HUNTER		
LUEDKE		
MARKOVICH		
STOFFERAHN		
STONE		
E WILSON		
R WILSON		

THURSDAY 1st Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
F-4 CF	F-4	STOFFERAHN	STAPP	A
C-23 PERF DEMO	C-23	MARKOVICH	DENOFRIO	A
T-38 LOW LIFT/DRAG	T-38	DENESIK	HOWELL	B
T-38 LOW LIFT/DRAG	T-38	BENDORF	SHANKAR	B
A-37 STALL DEMO	A-37	R WILSON	STAMBAUGH	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
BONASSO	A-7	1
BROWN	C-23	1
CARLSON	C-141	1
GOGAN		
GREEN		
GRUNNALD		
GARDNER		
HORTON		
HUNTER		
KANA		
LUEDKE		
LUTZ		
MOSER		
NELSON		
STONE		
WOOD		

THURSDAY 2nd Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
T-38 LOW LIFT/DRAG	T-38	SMOLKA	NICHOLS	B
T-38 LOW LIFT/DRAG	T-38	DENESIK	BARAK	B
T-38 LOW LIFT/DRAG	T-38	GOGAN	STAMBAUGH	B
F-16 FTT	F-16	BROWN	KELLER	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
BENDORF	A-7	1
BONASSO	C-23	1
CARLSON		
GREEN		
GRUNNALD		
HORTON		
HUNTER		
KANA		
LUEDKE		
LUTZ		
MARKOVICH		
MOSER		
NELSON		
STOFFERAHN		
STONE		
E WILSON		
R WILSON		
WOOD		
SOBCZAK		

FRIDAY 1st Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
T-38 LEVEL ACCEL DEM	T-38	LUTZ	BAKKEN	A
T-38 LEVEL ACCEL DEM	T-38	STONE	BLOOMFIELD	A
T-38 LEVEL ACCEL DEM	T-38	WOOD	CAREY	A
T-38 LOW LIFT/DRAG	T-38	KANA	VERDERAME	B

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
BENDORF	A-7	1
BROWN	F-16	1
CARLSON	C-141	1
DENESIK		
GOGAN		
GREEN		
GRUNNALD		
GARDNER		
HORTON		
HUNTER		
LUEDKE		
MARKOVICH		
MOSER		

FRIDAY 2nd Flight Period:

MISSION	AIRCRAFT	INSTRUCTOR	STUDENT	CLASS
T-38 LEVEL ACCEL DEM	T-38	SMOLKA	GRIFFITH	A
T-38 LEVEL ACCEL DEM	T-38	BENDORF	HUNNEL	A
F-4 CF	F-4	STOFFERAHN	WILHEIM	A

AVAILABLE RESOURCES

INSTRUCTOR	AIRCRAFT	AMOUNT
BROWN	A-37	1
CARLSON		
GOGAN		
GREEN		
GRUNNARD		
GARDNER		
HORTON		
HUNTER		
KANA		
LUEDKE		
LUTZ		
MARKOVICH		
MOSER		
NELSON		
STONE		
E WILSON		
R WILSON		
WOOD		

MISSIONS NOT SCHEDULED THIS WEEK:

MISSION	AIRCRAFT	STUDENT	CLASS
T-38 LEVEL ACCEL DEM	T-38	BALTRUSAIT	A
T-38 LEVEL ACCEL DEM	T-38	SHORT	A
T-38 LEVEL ACCEL DEM	T-38	DILLION	A
T-38 LEVEL ACCEL DEM	T-38	MURRAY	A
T-38 LEVEL ACCEL DEM	T-38	PAYNE	A
T-38 LEVEL ACCEL DEM	T-38	SHAW	A

MISSIONS THAT VIOLATE QOT DEADLINES:

MISSION	AIRCRAFT	STUDENT	CLASS
T-38 MAN FLT DATA	T-38	FAIRBAIRN	B
T-38 MAN FLT DATA	T-38	LEE	B
T-38 MAN FLT DATA	T-38	STERNBERG	B

Appendix C:

TPS Flight Scheduling Program Operating Instructions

Scheduling Program Background:

The program is written and compiled using Microsoft PC FORTRAN. It is designed to be used on personal computers. The following procedures should be used to run the model. The program accesses 22 different external data files. The information contained in these data files describes the environment of the TPS flight situation for any given week. The scheduling program uses this information to develop the flight schedule for the corresponding week.

Description of External Data Files:

Flight Mission Data Files: There are 8 different flight mission files. They contain information pertaining to the student missions that are eligible to be flown (scheduled) during the given week. Information includes student name, mission description, and the date at which the mission can first be flown by the student. Each of the two student classes has four mission data files dedicated to represent their missions. Furthermore, each file can contain up to three different mission types. Hence, up to 12 different mission types can be scheduled per class. The eight mission files are the following.

MSN1A.DAT	Class A student missions (1-3)
MSN2A.DAT	Class A student missions (4-6)
MSN3A.DAT	Class A student missions (7-9)
MSN4A.DAT	Class A student missions (10-12)
MSN1B.DAT	Class B student missions (1-3)
MSN2B.DAT	Class B student missions (4-6)
MSN3B.DAT	Class B student missions (7-9)
MSN4B.DAT	Class B student missions (10-12)

NOTE: Precedence information forcing a certain mission (such as a checkflight) to be flown before another mission by the same student is also contained in these files. This is accomplished by entering the negative of the ready date of the mission that must be flown first in the place corresponding to the ready date of the mission that can only be flown after the first mission.

Resource Availability Data Files: There are four different data files that represent the availability of the three key resources (aircraft, instructors, and students). Information contained in these files includes student name, instructor name, or aircraft type, and the corresponding availability status for the two morning flight periods for each of the flight days. The four availability data files are the following.

A_AVAIL.DAT	Class A student availability data
B_AVAIL.DAT	Class B student availability data
IP_AVAIL.DAT	Instructor pilot availability data
AC_AVAIL.DAT	Aircraft availability data

Mission Requirements Data File: The information contained in this file lists the aircraft type and instructor pilot qualification required by each mission type. This data file is named MSN_REQMT.DAT.

Instructor Pilot Qualification (Letter of X's) Data Files: There are nine data files of this type -- one for each aircraft type in the TPS Letter of X's Notebook. Information contained in these files includes instructor pilot name and qualifications attained by that instructor in a given aircraft. The nine data files are the following.

T-38_XS.DAT	Instructor qualifications for T-38
A-37_XS.DAT	Instructor qualifications for A-37
A-7_XS.DAT	Instructor qualifications for A-7
C-23_XS.DAT	Instructor qualifications for C-23
C-141_XS.DAT	Instructor qualifications for C-141
F-4_XS.DAT	Instructor qualifications for F-4
F-16_XS.DAT	Instructor qualifications for F-16
F-15_XS.DAT	Instructor qualifications for F-15
GLIDER_XS.DAT	Instructor qualifications for Glider

NOTE: The names of instructors, students, aircraft, mission descriptions, and qualifications must be entered by the same name in all data files. Otherwise, the scheduling program will not be able to recognize and equate them.

TPS Scheduling Program Hard Disk Installation:

If the TPS scheduling program is installed on a hard disk, the program and its associated files should have their own directory. To create a directory and copy files into the created directory, consult your DOS manual.

Updating External Data Files:

Each of the above external data files has a corresponding QUATTRO PRO spreadsheet file with a .WK file extension instead of the .DAT file extension. These spreadsheet files are formatted such that the data file formats required by the TPS scheduling program are maintained. Furthermore, the spreadsheet styl of editing provided by QUATTRO PRO allows for quick and efficient editing. *NOTE: It is not necessary to use the QUATTRO PRO spreadsheet files to update the .DAT files. The spreadsheet files are provided for ease and efficiency. The .DAT files can be updated in any manner desired by the user; however, their formats must be maintained or the TPS scheduling program will not be able to read them. To update the files using QUATTRO PRO, the following steps must be accomplished (for a complete guide for using QUATTRO PRO, consult your QUATTRO PRO manual).*

- 1) Enter the directory containing the TPS scheduling program files.
- 2) Enter the QUATTRO PRO program.
- 3) Load in the file to be updated. Type **/FR** and select the .WK file corresponding to the .DAT to be updated.
- 4) Make desired changes.
- 5) Save the updated file as a .DAT file (ASCII). Type **/PDF, filename.DAT**, select the **REPLACE** option, ensure the full file is blocked by QUATTRO PRO by using the **BLOCK** option, and then select the **SPREADSHEET PRINT** option.
- 6) The QUATTRO PRO spreadsheet file can also be save if desired -- it is not necessary.
- 7) Repeat steps 3-6 for all files to be updated.
- 8) Exit QUATTRO PRO.

Running the TPS Scheduling Program:

To have the TPS scheduling program produce a schedule for the flight week corresponding to the information contained in the external data files, perform the following steps.

- 1) Enter the directory containing the TPS scheduling program files.
- 2) Delete the old schedule. Type **del SCHED.OUT <ENTER>**
- 3) Run the program. Type **TPS <ENTER>**
- 4) The user will be prompted to enter Monday's date of the flight week and (if applicable) the dates of the academic test days for the student classes during the flight week. All dates must be entered in the format **MMDDYY**.
- 5) When the program terminates, the resulting schedule and related information will be provided in the file **SCHED.OUT**.

Appendix D:

TPS Flight Scheduling Program Source Code

```

CC*****
CC  SCHEDULE.FOR - TPS WEEKLY FLIGHT SCHEDULING PROGRAM          *
CC                                                                *
CC  References:  MS THESIS, AFIT/ENS GOR 92M Capt Gary Foster    *
CC*****
CC*****
CC  KEY VARIABLE LIST WITH DESCRIPTION:                          *
CC                                                                *
CC  num_A          = number of students in class A              *
CC  num_B          = number of students in class B              *
CC  num_IP         = number of instructor pilots                *
CC  num_AC         = number of aircraft types                   *
CC  num_FTT        = number of different mission types          *
CC  n_msns         = number of missions to be scheduled         *
CC  msn_list(*)    = mission description (type)                 *
CC  msndat(*,1)    = name of student to fly mission             *
CC  msndat(*,2)    = class of student                           *
CC  msndat(*,3)    = name of instructor to fly mission          *
CC  msndat(*,4)    = aircraft type scheduled for mission        *
CC  FTT_need(*,1)  = mission type requirement data              *
CC  FTT_need(*,2)  = required aircraft type for mission         *
CC  FTT_need(*,3)  = required instructor qualification for mission *
CC  rdate(*,1)    = mission ready date (adjustable)             *
CC  rdate(*,2)    = flight period that mission is scheduled    *
CC  rdate(*,3)    = scheduled flight period of best schedule   *
CC  rdate(*,4)    = priority rank of mission                    *
CC  rdate(*,5)    = number of times mission not scheduled       *
CC  rdate(*,6)    = mission ready date (fixed)                  *
CC  rdate(*,7)    = number of IPs qualified to fly given mission *
CC*****

```

PROGRAM tps

```

CC ** common block variables ***
CC  COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
CC    &,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
CC    &,msn_list(150), best_list(150)
CC  COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
CC    &,n_AC_per(20,10), n_msns, num_AC, num_FTT
CC    &,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
CC    &,best_AC(20,10), best_IP(30,10)
CC  CHARACTER*20 msn_list, FTT_need, best_list
CC  CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
CC    &,best_schd
CC  INTEGER rdate, workload, n_AC_per, n_msns, A_avail
CC    &,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
CC    &,best_AC, best_IP

```

```

CC ** main variables ***
    INTEGER i, j, n_cap_ip, schd_prd, no_go, start_date
    INTEGER a_test, b_test, counter, msn_num, rank
    CHARACTER*20 msn_type
    CHARACTER*10 ac_type, ip_cap_list(30), stdt_name
    &,ip_name, qual_need
    CHARACTER class
    LOGICAL found, test_day, avail

CC ** obtain date for the start day of the flight week to be scheduled **
13  WRITE (*,8000) ' Enter the flight week start date (MMDDYY): '
    READ  (*,*) start_date
CC ** convert date into day of year
    IF (start_date.GE.0.AND.start_date.LE.123200) THEN
        start_date = N_day(start_date)
    ELSE
        GOTO 13
    END IF

CC ** obtain class test days for week **
11  WRITE (*,8000) ' Enter class A test date (MMDDYY or 0 if none): '
    READ  (*,*) a_test
    IF (a_test.GE.0.AND.a_test.LE.123200) THEN
        IF (a_test.NE.0) THEN
            a_test = N_day(a_test)
            a_test = 2 * (a_test - start_date) + 1
        END IF
    ELSE
        GOTO 11
    END IF

12  WRITE (*,8000) ' Enter class B test date (MMDDYY or 0 if none): '
    READ  (*,*) b_test
    IF (b_test.GE.0.AND.b_test.LE.123200) THEN
        IF (b_test.NE.0) THEN
            b_test = N_day(b_test)
            b_test = 2 * (b_test - start_date) + 1
        END IF
    ELSE
        GOTO 12
    END IF

CC ** read in data (mission, resource, capability, etc) ***
CC ** pertaining to flight week to be scheduled ***
    PRINT*,' '
    PRINT*,'LOADING DATA FILES'
    CALL get_msns
    CALL ip_capblty
    CALL res_avail

```

```

CC ** initialize number of times mission not scheduled and attempt s**
DO i = 1, n_msns
    rdate(n_msns,5) = 0
END DO
counter = 1

CC ** initially rank missions according to ready dates ***
CALL msn_rank
PRINT*, 'PRIORITIZE MISSIONS'

CC ** BEGIN SCHEDULING MISSIONS **
CC ** schedule missions based on rank and resource availability **
70  no_go = 0
    PRINT*, 'SCHEDULING MISSIONS.  ITERATION # ', counter
    PRINT*, ' '
    DO 60 rank = 1, n_msns

CC ** find mission (msn_num) with priority (rank) **
        found = .FALSE.
        msn_num = 0
        DO 75 WHILE(.NOT.found.AND.msn_num.LT.n_msns)
            msn_num = msn_num + 1
            IF (rdate(msn_num,4).EQ.rank) THEN
                found = .TRUE.
            END IF
75      CONTINUE

CC ** reset/clear capable IP list, aircraft type, and mission
        n_cap_ip = 0
        DO j = 1, num_IP
            ip_cap_list(j) = 'temp'
        END DO
        msn_type = msn_list(msn_num)
        stdt_name = msndat(msn_num,1)
        class = msndat(msn_num,2)
        ac_type = ' '
        qual_need = ' '

CC ** find aircraft type and IP qualification needed for mission ***
        CALL ac_match(msn_type, ac_type, qual_need)
        msndat(msn_num,4) = ac_type

CC ** make list of IPs capable of flying mission ***
        CALL ip_qual(ac_type, qual_need, n_cap_ip, ip_cap_list)
        rdate(msn_num,7) = n_cap_ip

CC ** check to see if required preceding mission scheduled **
CC ** if not, this mission cannot be scheduled **
        IF (rdate(msn_num,1).LT.0) GOTO 60

```



```

CC ** determine first PERIOD that mission is eligible to be flown ***
CC ** based on its ready date ***
      schd_prd = 2 * (rdate(msn_num,1) - start_date)

CC ** if ready date from previous week, set 1st period to 1 ***
      IF (schd_prd.LT.0) THEN
        schd_prd = 0
      END IF

CC ** find period in which aircraft is available for mission **
90      CALL ac_period(ac_type, schd_prd)

      IF (schd_prd.LT.0) THEN
CC ** if aircraft not avail, tag msn as infeasible, and get next msn **
        no_go = no_go + 1
        rdate(msn_num,2) = 0
        GOTO 60
      END IF

CC ** if sched period is on test day, check to see if student is **
CC ** already scheduled to fly a mission that day. (QOT requirement) **
      test_day = .FALSE.
      IF ((class.EQ.'A'.AND.a_test.NE.0).OR.
&      (class.EQ.'B'.AND.b_test.NE.0)) THEN
&      CALL check_day(schd_prd, stdt_name, class, a_test, b_test
&      ,test_day, rank)
      END IF
      IF (test_day) GOTO 90

CC ** check IP availability and workload for that period and select IP ***
      IF (ip_cap_list(1).EQ.'N/A') THEN
        ip_name = 'N/A'
      ELSE
        ip_name = 'NONE'
        CALL ip_schd(schd_prd, n_cap_ip, ip_cap_list, ip_name)
      END IF

CC ** if no IP available, check aircraft availability for another period **
      IF (ip_name.EQ.'NONE') GOTO 90

CC ** check student availability ***
      avail = .TRUE.
      CALL stdt_check(schd_prd, stdt_name, class, avail)
CC ** if student not avail, check aircraft avail for another period **
      IF (.NOT.avail) GOTO 90

CC ** schedule mission, adjust resource levels, check for precedence msn **
      msndat(msn_num,3) = ip_name
      rdate(msn_num,2) = schd_prd
      CALL res_adjst(schd_prd, class, ip_name, ac_type, stdt_name)
      CALL prec_msn(msn_num, start_date)

```

```

CC ** schedule next mission ***
  60  CONTINUE

CC ** if less than max number of iterations and
CC ** there are any unscheduled missions ***
      IF (counter.LT.7.AND.no_go.GT.0) THEN
CC ** save best schedule, reset resource levels, and try again **
          CALL save_best(counter)
          CALL res_avail
          PRINT*, 'RE-INITIALIZING RESOURCE LEVELS'
          counter = counter + 1
CC ** reset mission ready dates to original values **
          DO j = 1, n_msns
              rdate(j,1) = rdate(j,6)
          END DO
CC ** re-rank missions based on ready dates and **
CC ** number of times not scheduled **
          CALL msn_rank
          PRINT*, 'RE-PRIORITIZING MISSIONS'
          GOTO 70
      ELSE
CC ** release flight schedule for the week **
          CALL save_best(counter)
          CALL schd_out(start_date)
          PRINT*, ' '
          PRINT*, 'SCHEDULING ALGORITHM COMPLETE.'
          PRINT*, 'OUTPUT IS CONTAINED IN THE FILE "SCHED.OUT"'
      END IF

8000 FORMAT( // A \ )

      STOP
      END

```

SUBROUTINE get_msns

CC ** common block variables ***

```
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP
```

CC This subroutines reads mission data from external data files
 CC This mission data pertains to missions planned to be flown during
 CC the given week. Information includes: mission description, student
 CC name, and mission ready date.

```
INTEGER i, j, date(15,15), temp, unum
CHARACTER*20 msn_type(150)
CHARACTER*10 astdnt(30), bstdnt(30)
```

```
OPEN(UNIT=1,FILE='msn1a.dat',STATUS='OLD')
OPEN(UNIT=2,FILE='msn2a.dat',STATUS='OLD')
OPEN(UNIT=3,FILE='msn3a.dat',STATUS='OLD')
OPEN(UNIT=4,FILE='msn4a.dat',STATUS='OLD')
OPEN(UNIT=5,FILE='msn1b.dat',STATUS='OLD')
OPEN(UNIT=6,FILE='msn2b.dat',STATUS='OLD')
OPEN(UNIT=7,FILE='msn3b.dat',STATUS='OLD')
OPEN(UNIT=8,FILE='msn4b.dat',STATUS='OLD')
```

```
n_msns = 0
unum = 0
```

CC read in and echo weekly mission data for class A

```
DO unum = 1, 4
  READ(UNIT=unum,FMT=9100) (msn_type(j), j = 1, 3)
  i = 0
888  i = i + 1
  READ(UNIT=unum,FMT=9200) astdnt(i),(date(i,j), j = 1, 3)
  DO j = 1, 3
    IF (date(i,j).NE.0) THEN
      n_msns = n_msns + 1
      msn_list(n_msns) = msn_type(j)
      msndat(n_msns,1) = astdnt(i)
      msndat(n_msns,2) = 'A'
      msndat(n_msns,3) = ' '
      msndat(n_msns,4) = ' '
```

```

        temp = date(i,j)
        rdate(n_msns,1) = N_day(temp)
        rdate(n_msns,6) = rdate(n_msns,1)
        rdate(n_msns,4) = n_msns
        rdate(n_msns,7) = 0
    END IF
END DO
    IF (astdnt(i).NE.'END'.AND.i.LT.30) GOTO 888
END DO

CC  read in and echo weekly mission data for class B
    D: unum = 5, 8
    READ(UNIT=unum,FMT=9100) (msn_type(j), j = 1, 3)
    i = 0
777  i = i + 1
    READ(UNIT=unum,FMT=9200) bstdnt(i),(date(i,j), j = 1, 3)
    DO j = 1, 3
        IF (date(i,j).NE.0) THEN
            n_msns = n_msns + 1
            msn_list(n_msns) = msn_type(j)
            msndat(n_msns,1) = bstdnt(i)
            msndat(n_msns,2) = 'B'
            msndat(n_msns,3) = ' '
            msndat(n_msns,4) = ' '
            temp = date(i,j)
            rdate(n_msns,1) = N_day(temp)
            rdate(n_msns,6) = rdate(n_msns,1)
            rdate(n_msns,4) = n_msns
            rdate(n_msns,7) = 0
        END IF
    END DO
    IF (bstdnt(i).NE.'END'.AND.i.LT.30) GOTO 777
END DO

CLOSE(UNIT=1)
CLOSE(UNIT=2)
CLOSE(UNIT=3)
CLOSE(UNIT=4)
CLOSE(UNIT=5)
CLOSE(UNIT=6)
CLOSE(UNIT=7)
CLOSE(UNIT=8)

9100 FORMAT(11X,3(2X,A20))
9200 FORMAT(1X,A10,3(2X,I20))
9250 FORMAT(1X,A20,2X,A8,3X,A1,3X,I4)
RETURN
END

```

SUBROUTINE ip_capblty

CC ** common block variables ***

```

COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

```

CC This subroutines reads IP capability data from an external data file
CC This data pertains to IP qualifications to fly
CC a given FTT mission. Information includes: mission description,
CC aircraft type, and FTT mission qualification status.

```

INTEGER i
CHARACTER*20 head1
CHARACTER*10 head2, head3

```

```

OPEN(UNIT=13,FILE='msn_reqmt.dat',STATUS='OLD')

```

```

READ(UNIT=13,FMT=9300) head1, head2, head3

```

```

i = 0

```

```

222 i = i + 1

```

```

    READ(UNIT=13,FMT=9300) FTT_need(i,1),FTT_need(i,2),
&                          FTT_need(i,3)
    IF (FTT_need(i,1).NE.'END'.AND.i.LT.140) GOTO 222
    num_FTT = i - 1

```

```

CLOSE(UNIT=13)

```

```

9300 FORMAT(1X,A20,4X,A10,4X,A10)

```

```

RETURN
END

```

SUBROUTINE res_avail

CC ** common block variables ***

```

COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

```

CC This subroutines reads in resource availability data for the week
CC from external data files. This data pertains to each resources
CC (Aircraft, IPs, and students from both classes A and B) availability
CC for the week.

```

INTEGER i, j
CHARACTER*10 header, class, period(10)
CHARACTER temp(10)

```

```

OPEN(UNIT=9,FILE='ac_avail.dat',STATUS='OLD')
OPEN(UNIT=10,FILE='ip_avail.dat',STATUS='OLD')
OPEN(UNIT=11,FILE='A_avail.dat',STATUS='OLD')
OPEN(UNIT=12,FILE='B_avail.dat',STATUS='OLD')

```

CC ** read in expected aircraft availability for given flight week ***

```

READ(UNIT=9,FMT=9400) (period(j), j = 1, 10)
i = 0

```

111 i = i + 1

```

READ(UNIT=9,FMT=9500) AC_list(i), (n_AC_per(i,j), j = 1, 10)
IF (AC_list(i).NE.'END'.AND.i.LT.20) GOTO 111
num_AC = i - 1

```

CC ** read in IP availability for given flight week **

```

READ(UNIT=10,FMT=9450) (period(j), j = 1, 10), header
i = 0

```

333 i = i + 1

```

READ(UNIT=10,FMT=9600) IP_list(i), (temp(j),j = 1, 10)
&,workload(i)
DO j = 1, 10
  IF (temp(j).EQ.'Y') THEN
    IP_avail(i,j) = 1
  ELSE
    IP_avail(i,j) = 0
  END IF

```

```

        END DO
        IF (IP_list(i).NE.'END'.AND.i.LT.30) GOTO 333
        num_IP = i - 1

```

```

CC  ** read in class A student availability for given flight week **
    READ(UNIT=11,FMT=9750) class, (period(j), j = 1, 10)
    i = 0
444  i = i + 1
    READ(UNIT=11,FMT=9700) A_list(i), (temp(j), j = 1, 10)
    DO j = 1, 10
        IF (temp(j).EQ.'Y') THEN
            A_avail(i,j) = 1
        ELSE
            A_avail(i,j) = 0
        END IF
    END DO
    IF (A_list(i).NE.'END'.AND.i.LT.30) GOTO 444
    num_A = i - 1

```

```

CC  ** read in class B student availability for given flight week **

    READ(UNIT=12,FMT=9750) class, (period(j), j = 1, 10)
    i = 0
555  i = i + 1
    READ(UNIT=12,FMT=9700) B_list(i), (temp(j), j = 1, 10)
    DO j = 1, 10
        IF (temp(j).EQ.'Y') THEN
            B_avail(i,j) = 1
        ELSE
            B_avail(i,j) = 0
        END IF
    END DO
    IF (B_list(i).NE.'END'.AND.i.LT.30) GOTO 555
    num_B = i - 1

```

```

    CLOSE(UNIT=9)
    CLOSE(UNIT=10)
    CLOSE(UNIT=11)
    CLOSE(UNIT=12)

```

```

9400 FORMAT(8X,10A6)
9450 FORMAT(13X,11A5)
9500 FORMAT(1X,A7,10I6)
9600 FORMAT(1X,A10,2X,10(A1,4X),I5)
9700 FORMAT(1X,A10,2X,10(A1,4X))
9750 FORMAT(1X,A10,2X,10A5)

```

```

    RETURN
    END

```

```

        INTEGER FUNCTION n_day (ndate)
CC
CC  Converts ready date into day of year
CC
        INTEGER ndate, flag

CC *** IF DATE IS NEGATIVE (DEPENDENT ON ANOTHER MISSION), FLAG IT **
        flag = 1
        IF (ndate.LT.0) THEN
            flag = -1
            ndate = ndate * flag
        END IF

CC ** convert date into day of year ****
        IF (ndate.LT.20000) THEN
            n_day = INT((ndate - 10000)/100)
        ELSE IF (ndate.LT.30000) THEN
            n_day = 31 + INT((ndate - 20000)/100)
        ELSE IF (ndate.LT.40000) THEN
            n_day = 59 + INT((ndate - 30000)/100)
        ELSE IF (ndate.LT.50000) THEN
            n_day = 90 + INT((ndate - 40000)/100)
        ELSE IF (ndate.LT.60000) THEN
            n_day = 120 + INT((ndate - 50000)/100)
        ELSE IF (ndate.LT.70000) THEN
            n_day = 151 + INT((ndate - 60000)/100)
        ELSE IF (ndate.LT.80000) THEN
            n_day = 181 + INT((ndate - 70000)/100)
        ELSE IF (ndate.LT.90000) THEN
            n_day = 212 + INT((ndate - 80000)/100)
        ELSE IF (ndate.LT.100000) THEN
            n_day = 243 + INT((ndate - 90000)/100)
        ELSE IF (ndate.LT.110000) THEN
            n_day = 273 + INT((ndate - 100000)/100)
        ELSE IF (ndate.LT.120000) THEN
            n_day = 304 + INT((ndate - 110000)/100)
        ELSE
            n_day = 334 + INT((ndate - 120000)/100)
        END IF
        n_day = n_day * flag

        RETURN
    END

```


SUBROUTINE msn_rank

CC ** common block variables **

```
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP
```

CC *** This subroutine ranks the missions in order of earliest ***
 CC *** ready dates. Flights that cannot be flown prior to the completion
 CC *** of another flight are ranked last. Their ranking is updated once
 CC *** the required flight is completed. ****

```
INTEGER i, dummy(150,4), temp(4), msn_num
LOGICAL found, sorted, ranked
```

CC ** set up dummy arrays **

```
DO i = 1, n_msns
  dummy(i,1) = rdate(i,6)
  dummy(i,2) = i
  dummy(i,3) = rdate(i,5)
  dummy(i,4) = rdate(i,7)
END DO
```

CC *** perform the sort until mission ranked by ready dates ***
 CC *** and IPs qual to fly mission type ***

```
sorted = .FALSE.
```

```
20 IF (.NOT.sorted) THEN
```

```
  sorted = .TRUE.
```

```
  DO 10 i = 1, n_msns-1
```

CC *** IF MISSION HAS LATER READY DATE OR DEPENDENT READY DATE ***

```
  IF ((dummy(i,1).GT.dummy(i+1,1).AND.dummy(i+1,1).GT.0)
&      .OR.(dummy(i+1,4).LT.dummy(i,4)
&      .AND.dummy(i+1,1).EQ.dummy(i,1).AND.dummy(i+1,1).GE.0)
&      .OR.(dummy(i,1).LT.0.AND.dummy(i+1,1).GE.0)) THEN
```

CC *** EXCHANGE OUT OF RANK MISSION PRIORITIES *****

```
    temp(1) = dummy(i,1)
    temp(2) = dummy(i,2)
    temp(3) = dummy(i,3)
    temp(4) = dummy(i,4)
    dummy(i,1) = dummy(i+1,1)
    dummy(i,2) = dummy(i+1,2)
    dummy(i,3) = dummy(i+1,3)
```

```

        dummy(i,4) = dummy(i+1,4)
        dummy(i+1,1) = temp(1)
        dummy(i+1,2) = temp(2)
        dummy(i+1,3) = temp(3)
        dummy(i+1,4) = temp(4)
        sorted = .false.
    END IF
10    CONTINUE
    GOTO 20
END IF

CC *** sort based on number of times mission not scheduled ***
    ranked = .FALSE.
25    IF (.NOT.ranked) THEN
        ranked = .TRUE.
        DO 15 i = 1, n_msns-1
CC *** if mission not scheduled more than other mission ***
            IF (dummy(i+1,3).GT.dummy(i,3).AND.dummy(i+1,1).GE.0) THEN
CC *** EXCHANGE OUT OF RANK MISSIONS *****
                temp(1) = dummy(i,1)
                temp(2) = dummy(i,2)
                temp(3) = dummy(i,3)
                temp(4) = dummy(i,4)
                dummy(i,1) = dummy(i+1,1)
                dummy(i,2) = dummy(i+1,2)
                dummy(i,3) = dummy(i+1,3)
                dummy(i,4) = dummy(i+1,4)
                dummy(i+1,1) = temp(1)
                dummy(i+1,2) = temp(2)
                dummy(i+1,3) = temp(3)
                dummy(i+1,4) = temp(4)
                ranked = .false.
            END IF
15    CONTINUE
    GOTO 25
END IF

CC ** re-assign rank to real variables **
    DO i = 1, n_msns
CC ** find mission (msn_num) with priority (rank) **
        found = .FALSE.
        msn_num = 0
        DO 77 WHILE(.NOT.found.AND.msn_num.LT.n_msns)
            msn_num = msn_num + 1
            IF (dummy(msn_num,2).EQ.i) THEN
                rdate(i,4) = msn_num
                found = .TRUE.
            END IF
77    CONTINUE
    END DO

8250 FORMAT(1X,A20,2X,A8,3X,A1,3X,4I4)

```

RETURN
END

SUBROUTINE ac_match (msn_type, ac_type, qual_need)

CC ** common block variables ***

```
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP
```

CC ** This subroutine finds the aircraft type and IP ****
CC ** qualification needed for a given FTT mission. ***

```
CHARACTER*20 msn_type
CHARACTER*10 ac_type, qual_need
```

```
INTEGER row
LOGICAL found
```

CC ** search aircraft type for match, tag row number of aircraft ***
found = .FALSE.

row = 0

DO 55 WHILE(.NOT.found.AND.row.LT.num_FTT)

row = row + 1

IF (msn_type.EQ.FTT_need(row,1)) THEN

found = .TRUE.

ac_type = FTT_need(row,2)

qual_need = FTT_need(row,3)

END IF

55 CONTINUE

IF (.NOT.found) THEN

PRINT*,'ERROR -- MISSION TYPE NOT FOUND IN SUB ac_match.'

PRINT*,'CHECK INPUT FILES FOR: ',msn_type

END IF

RETURN
END

```

SUBROUTINE ip_qual (ac_type, qual_need, n_cap_ip, ip_cap_list)

CC ** common block variables ***
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

CC ** This subroutine determines how many and which IPs ****
CC ** are qualified to fly a given mission type. ***

INTEGER i, j, col, unum, n_cap_ip
CHARACTER*10 ac_type, qual_need, ip_cap_list(30), space
&,quals(9), name_ip

CC ** originally assumes no IP required for mission ***
unum = 0
n_cap_ip = 1
ip_cap_list(1) = 'N/A'

IF (qual_need.NE.'N/A') THEN
CC ** open file corresponding to required aircraft type **
IF (ac_type.EQ.'T-38') THEN
unum = 14
OPEN(UNIT=unum,FILE='T-38_Xs.dat',STATUS='OLD')
ELSE IF (ac_type.EQ.'A-7') THEN
unum = 15
OPEN(UNIT=unum,FILE='A-7_Xs.dat',STATUS='OLD')
ELSE IF (ac_type.EQ.'A-37') THEN
unum = 16
OPEN(UNIT=unum,FILE='A-37_Xs.dat',STATUS='OLD')
ELSE IF (ac_type.EQ.'F-4') THEN
unum = 17
OPEN(UNIT=unum,FILE='F-4_Xs.dat',STATUS='OLD')
ELSE IF (ac_type.EQ.'C-23') THEN
unum = 18
OPEN(UNIT=unum,FILE='C-23_Xs.dat',STATUS='OLD')
ELSE IF (ac_type.EQ.'GLIDER') THEN
unum = 19
OPEN(UNIT=unum,FILE='GLIDER_Xs.dat',STATUS='OLD')
ELSE IF (ac_type.EQ.'C-141') THEN
unum = 20
OPEN(UNIT=unum,FILE='C-141_Xs.dat',STATUS='OLD')

```

```

ELSE IF (ac_type.EQ.'F-16') THEN
    unum = 21
    OPEN(UNIT=unum,FILE='F-16_Xs.dat',STATUS='OLD')
ELSE IF (ac_type.EQ.'F-15') THEN
    unum = 22
    OPEN(UNIT=unum,FILE='F-15_Xs.dat',STATUS='OLD')
END IF
END IF

CC ** find column of required IP qualification ****
IF (unum.NE.0) THEN
    READ(UNIT=unum,FMT=8800) space, (quals(j), j = 1,9)

    col = 10
    DO i = 1, 9
        IF (qual_need.EQ.quals(i)) THEN
            col = i
        END IF
    END DO
    IF (col.EQ.10) THEN
        PRINT*, 'IP QUALIFICATION NOT FOUND. AIRCRAFT =', ac_type
        PRINT*, 'QUALIFICATION TYPE = ', qual_need
        PRINT*, 'CHECK INPUT FILES FOR INPUT ERROR.'
    END IF

CC ** make list of qualified IPs ***
    n_cap_ip = 0
    ip_cap_list(1) = 'temp'
767 READ(UNIT=unum,FMT=8800) name_ip, (quals(j), j = 1,9)
    IF (quals(col).EQ.'X') THEN
        n_cap_ip = n_cap_ip + 1
        ip_cap_list(n_cap_ip) = name_ip
    END IF
    IF (name_ip.NE.'END') GOTO 767
    CLOSE(UNIT=unum)
END IF

8800 FORMAT(1X,A10,2X,9A6)

RETURN
END

```

```

SUBROUTINE ac_period (ac_type, schd_prd)

CC ** common block variables ***
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

CC ** This subroutine finds the earliest flight period in which ***
CC ** the needed aircraft type is available for a mission ***
INTEGER schd_prd, row, col
CHARACTER*10 ac_type
LOGICAL found

CC ** search aircraft type for match, tag row number of aircraft ***
found = .FALSE.
row = 0
DO 25 WHILE(.NOT.found.AND.row.LT.num_AC)
    row = row + 1
    IF (ac_type.EQ.AC_list(row)) THEN
        found = .TRUE.
    END IF
25 CONTINUE
IF (.NOT.found) THEN
    PRINT*, 'ERROR -- AIRCRAFT TYPE NOT FOUND in SUB AC_period.'
    PRINT*, 'CHECK INPUT FILES FOR: ',ac_type
END IF

CC ** search aircraft availability to find earliest period in which **
CC ** needed aircraft is available after current period (schd_prd) **
col = schd_prd
found = .FALSE.
DO 30 WHILE(.NOT.found.AND.col.LT.10)
    col = col + 1
    IF (n_AC_per(row,col).GT.0) THEN
        found = .TRUE.
        schd_prd = col
    END IF
30 CONTINUE
IF (.NOT.found) THEN
    schd_prd = -18
END IF
RETURN
END

```

```

SUBROUTINE check_day(schd_prd, stdt_name, class, a_test, b_test
&, test_day, rank)

```

```

CC ** common block variables ***

```

```

COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

```

```

CC ** This subroutine checks to see if the student has a test scheduled **
CC ** on the same day as the candidate flight period. If so, it checks **
CC ** to see if the student has already been scheduled for a mission **
CC ** that day. QOT reqmts limit students to one mission on test days **

```

```

INTEGER schd_prd, a_test, b_test, rank
CHARACTER*10 stdt_name
CHARACTER class
LOGICAL test_day

```

```

INTEGER i, msn_tag
LOGICAL found

```

```

CC ** period on test day of student's class **

```

```

IF (((schd_prd.EQ.a_test.OR.schd_prd.EQ.a_test+1).AND.
& class.EQ.'A').OR.
& ((schd_prd.EQ.b_test.OR.schd_prd.EQ.b_test+1).AND.
& class.EQ.'B')) THEN

```

```

CC ** search through missions scheduled so far in this iteration **

```

```

DO i = 1, rank-1
found = .FALSE.
msn_tag = 0
DO 66 WHILE(.NOT.found.AND.msn_tag.LT.n_msns)
msn_tag = msn_tag + 1
IF (rdate(msn_tag,4).EQ.i) THEN
found = .TRUE.

```

```

CC ** if student name and class match of scheduled mission **

```

```

IF ((stdt_name.EQ.msndat(msn_tag,1)).AND.
& (class.EQ.msndat(msn_tag,2))) THEN

```

```

CC ** and if mission scheduled in 2nd period of test day **

```

```

IF ((MOD(schd_prd,2).EQ.0).AND.
& ((schd_prd.EQ.rdate(msn_tag,2)).OR.
& (schd_prd.EQ.(rdate(msn_tag,2)+1)))) THEN

```

```

                                test_day = .TRUE.
CC ** else if mission scheduled in 1st period of test day **
                                ELSE IF ((MOD(schd_prd,2).NE.0).AND.
&                                ((schd_prd.EQ.rdate(msn_tag,2)).OR.
&                                (schd_prd.EQ.(rdate(msn_tag,2)-1)))) THEN
                                schd_prd = schd_prd + 1
                                test_day = .TRUE.
CC ** otherwise no conflict **
                                ELSE
                                test_day = .FALSE.
                                END IF
                                END IF
                                END IF
66                                CONTINUE
                                END DO
                                END IF

                                RETURN
                                END

```



```

SUBROUTINE ip_schd(schd_prd, n_cap_ip, ip_cap_list, ip_name)

CC ** common block variables ***
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

CC ** This subroutine checks IP availability and workload for ***
CC ** the candidate flight period. If IP resources are available **
CC ** it selects the IP with the lowest workload ***

INTEGER schd_prd, n_cap_ip
CHARACTER*10 ip_cap_list(30), ip_name

INTEGER i, j, best

best = 0
DO i = 1, n_cap_ip
  DO j = 1, num_IP
    IF (ip_cap_list(i).EQ.IP_list(j)
    & .AND.IP_avail(j,schd_prd).EQ.1) THEN
CC ** if IP both available and capable to instruct mission ***
      IF (best.EQ.0.OR.workload(j).LT.workload(best)) THEN
CC ** select one with least workload **
        best = j
        ip_name = IP_list(best)
      END IF
    END IF
  END DO
END DO

RETURN
END

```

```

SUBROUTINE stdt_check(schd_prd, stdt_name, class, avail)

CC ** common block variables ***
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

CC ** This subroutine checks to see if the student is available **

INTEGER schd_prd
CHARACTER*10 stdt_name
CHARACTER class
LOGICAL avail

INTEGER row
LOGICAL found

found = .FALSE.
row = 0
CC ** branch based on class A or class B ***
IF (class.EQ.'A') THEN
CC ** search class A student name for match ***
DO 42 WHILE(.NOT.found.AND.row.LT.num_A)
row = row + 1
IF (stdt_name.EQ.A_list(row)) THEN
found = .TRUE.
END IF
42 CONTINUE
IF (.NOT.found) THEN
PRINT*, 'ERROR -- STUDENT NAME NOT FOUND.'
PRINT*, 'CHECK INPUT FILES FOR: ',stdt_name
CC ** if student not avail, tag as none **
ELSE IF (A_avail(row,schd_prd).EQ.0) THEN
avail = .FALSE.
END IF
ELSE

```

```

CC ** search class B student name for match ***
      DO 43 WHILE(.NOT.found.AND.row.LT.num_B)
        row = row + 1
        IF (stdt_name.EQ.B_list(row)) THEN
          found = .TRUE.
        END IF
43      CONTINUE
        IF (.NOT.found) THEN
          PRINT*, 'ERROR -- STUDENT NAME NOT FOUND.'
          PRINT*, 'CHECK INPUT FILES FOR: ', stdt_name
CC ** if student not avail, tag as none **
          ELSE IF (B_avail(row, schd_prd).EQ.0) THEN
            avail = .FALSE.
          END IF
        END IF

      RETURN
      END

```

```

SUBROUTINE res_adjst(schd_prd, class, ip_name, ac_type, stdt_name)

CC ** common block variables ***
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

CC ** This subroutine adjusts the availability levels of the different ***
CC ** resources once a mission is scheduled. ****

INTEGER schd_prd
CHARACTER*10 ip_name, ac_type, stdt_name
CHARACTER class

INTEGER row
LOGICAL found

found = .FALSE.
row = 0
CC ** branch based on class A or class B and adjust student avail ***
IF (class.EQ.'A') THEN
CC ** search class A student name for match ***
DO 44 WHILE(.NOT.found.AND.row.LT.num_A)
row = row + 1
IF (stdt_name.EQ.A_list(row)) THEN
found = .TRUE.
A_avail(row,schd_prd) = 0
IF (ac_type.EQ.'GLIDER'.OR.ac_type.EQ.'C-141') THEN
A_avail(row,schd_prd+1) = 0
END IF
END IF
44 CONTINUE
ELSE
CC ** search class B student name for match ***
DO 45 WHILE(.NOT.found.AND.row.LT.num_B)
row = row + 1
IF (stdt_name.EQ.B_LIST(row)) THEN
found = .TRUE.
B_avail(row,schd_prd) = 0
IF (ac_type.EQ.'GLIDER'.OR.ac_type.EQ.'C-141') THEN
B_avail(row,schd_prd+1) = 0

```

```

        END IF
    END IF
45    CONTINUE
END IF

CC ** search aircraft type for match, decrease number of aircraft ***
found = .FALSE.
row = 0
DO 46 WHILE(.NOT.found.AND.row.LT.num_AC)
    row = row + 1
    IF (ac_type.EQ.AC_list(row)) THEN
        found = .TRUE.
        n_AC_per(row,schd_prd) = n_AC_per(row,schd_prd) - 1
    END IF
46 CONTINUE

CC ** search IP list for match, set IP not avail ***
found = .FALSE.
row = 0
DO 47 WHILE(.NOT.found.AND.row.LT.num_IP)
    row = row + 1
    IF (ip_name.EQ.IP_list(row)) THEN
        found = .TRUE.
        IP_avail(row,schd_prd) = 0
        IF (ac_type.EQ.'GLIDER'.OR.ac_type.EQ.'C-141') THEN
            IP_avail(row,schd_prd+1) = 0
        END IF
        workload(row) = workload(row) + 1
    END IF
47 CONTINUE

RETURN
END

```

```

SUBROUTINE prec_msn(msn_tag, start_date)

CC ** common block variables ***
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

CC ** This subroutine checks to see if the mission just scheduled **
CC ** is required to be flown before another mission to be scheduled **
CC ** If it is, the ready date for the follow on mission is adjusted **
CC ** IAW QOT guidelines to be scheduled **

INTEGER msn_tag, start_date

INTEGER i

CC ** search for follow on mission **
DO i = 1, n_msns
  IF (msndat(msn_tag,1).EQ.msndat(i,1).AND.
&    msndat(msn_tag,2).EQ.msndat(i,2).AND.
&    rdate(i,1).LT.0.AND.
&    rdate(msn_tag,1).EQ.ABS(rdate(i,1))) THEN
CC ** if found, set the ready date of the follow on mission to 1 day **
CC ** after the required mission was scheduled to be flown **
    rdate(i,1) = start_date+NINT((rdate(msn_tag,2)/2+.2))
    RETURN
  END IF
END DO

RETURN
END

```

```

SUBROUTINE save_best(iteration)

CC ** common block variables ***
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

CC *** This subroutine saves the schedule with fewest unscheduled missions
**

INTEGER i, j, k, best_miss, cur_miss, iteration
REAL cur_work_var, best_work_var, mean_work, sum_sqr_work
&,sum_work

CC ** total number of missed missions for each schedule ****
cur_miss = 0
best_miss = 0
DO i = 1, n_msns
  IF (rdate(i,2).EQ.0) THEN
    cur_miss = cur_miss + 1
CC ** count the number of times this mission not scheduled **
    IF (rdate(i,1).LT.0) THEN
CC ** if unsched msn is follow-on, search for precedent mission **
      DO k = 1, n_msns
        IF (msndat(k,1).EQ.msndat(i,1).AND.
&          msndat(k,2).EQ.msndat(i,2).AND.
&          rdate(k,1).EQ.ABS(rdate(i,1))) THEN
CC ** when found, add # of times not scheduled to precedent msn **
          rdate(k,5) = rdate(k,5) + 1
        END IF
      END DO
    ELSE
CC ** count # times not scheduled
      rdate(i,5) = rdate(i,5) + 1
    END IF
  END IF
  IF (rdate(i,3).EQ.0) THEN
    best_miss = best_miss + 1
  END IF
END DO

```

```

CC ** calculate the variance of IP workload for the current schedule **
    sum_work = 0
    sum_sqr_work = 0
    DO i = 1, num_IP
        sum_work = sum_work + workload(i)
    END DO
    mean_work = sum_work / num_IP
    DO i = 1, num_IP
        sum_sqr_work = sum_sqr_work + (workload(i) - mean_work)**2
    END DO
    cur_work_var = sum_sqr_work / num_IP

CC ** if current schedule has fewer missed missions or the same
CC ** number of missed missions with a more even workload, save it **
    IF ((cur_miss.LT.best_miss).OR.(iteration.EQ.1).OR.
    & (cur_miss.EQ.best_miss.AND.cur_work_var.LT.best_work_var)) THEN
        best_iter = iteration
        best_work_var = cur_work_var
        DO i = 1, n_msns
            rdate(i,3) = rdate(i,2)
            best_list(i) = msn_list(i)
            DO j = 1, 4
                best_schd(i,j) = msndat(i,j)
            END DO
        END DO
CC ** save the availability of aircraft and IPs of best schedule **
        DO i = 1, 10
            DO j = 1, num_AC
                best_AC(j,i) = n_AC_per(j,i)
            END DO
            DO j = 1, num_IP
                best_IP(j,i) = IP_avail(j,i)
            END DO
        END DO
    END IF

RETURN
END

```



```

SUBROUTINE schd_out(start_date)

CC ** common block variables ***
COMMON /TPS_CHAR/ msndat(150,4), FTT_need(140,3), A_list(30)
&,B_list(30), IP_list(30), AC_list(20), best_schd(150,4)
&,msn_list(150), best_list(150)
COMMON /TPS_INT/ rdate(150,7), workload(30), num_A, num_B
&,n_AC_per(20,10), n_msns, num_AC, num_FTT
&,A_avail(30,10), B_avail(30,10), IP_avail(30,10), num_IP
&,best_AC(20,10), best_IP(30,10)
CHARACTER*20 msn_list, FTT_need, best_list
CHARACTER*10 A_list, B_list, IP_list, AC_list, msndat
&,best_schd
INTEGER rdate, workload, n_AC_per, n_msns, A_avail
&,B_avail, IP_avail, num_AC, num_A, num_B, num_IP, num_FTT
&,best_AC, best_IP

CC *** This subroutine saves the schedule which was determined to ***
CC *** be the best in an external file for print out. ***

INTEGER i, j, start_date, loop, avl_IP, avl_AC, n_temp(20)
CHARACTER*20 line, head1
CHARACTER*10 head2, head3, head4, head5, blank, head6
&,temp_AC(20), temp_IP(30)

OPEN(UNIT=25,FILE='sched.out',STATUS='NEW')

line = '*****'
head1 = 'MISSION'
head2 = 'AIRCRAFT'
head3 = 'INSTRUCTOR'
head4 = 'STUDENT'
head5 = 'CLASS'
head6 = 'AMOUNT'
blank = ' '
DO i = 1, 10
  IF (i.EQ.1) THEN
    WRITE(25,*) ' MONDAY 1st Flight Period:'
  ELSE IF (i.EQ.2) THEN
    WRITE(25,*) ' MONDAY 2nd Flight Period:'
  ELSE IF (i.EQ.3) THEN
    WRITE(25,*) ' TUESDAY 1st Flight Period:'
  ELSE IF (i.EQ.4) THEN
    WRITE(25,*) ' TUESDAY 2nd Flight Period:'
  ELSE IF (i.EQ.5) THEN
    WRITE(25,*) ' WEDNESDAY 1st Flight Period:'
  ELSE IF (i.EQ.6) THEN
    WRITE(25,*) ' WEDNESDAY 2nd Flight Period:'
  ELSE IF (i.EQ.7) THEN
    WRITE(25,*) ' THURSDAY 1st Flight Period:'
  ELSE IF (i.EQ.8) THEN

```

```

        WRITE(25,*) ' THURSDAY  2nd Flight Period:'
ELSE IF (i.EQ.9) THEN
        WRITE(25,*) ' FRIDAY   1st Flight Period:'
ELSE
        WRITE(25,*) ' FRIDAY   2nd Flight Period:'
END IF
WRITE(25,7000) line,line,line,line
WRITE(25,7100) head1,head2,head3,head4,head5
WRITE(25,7000) line,line,line,line
DO j = 1, n_msns
    IF (rdate(j,3).EQ.i) THEN
        WRITE(25,7100) best_list(j), best_schd(j,4)
&         ,best_schd(j,3), best_schd(j,1), best_schd(j,2)
    END IF
END DO
WRITE(25,*)
CC ** determine IPs and aircraft still available in this period **
    avl_IP = 0
    avl_AC = 0
    DO j = 1, num_IP
        temp_IP(j) = ' '
        IF (best_IP(j,i).EQ.1) THEN
            avl_IP = avl_IP + 1
            temp_IP(avl_IP) = IP_list(j)
        END IF
    END DO
    DO j = 1, num_AC
        temp_AC(j) = ' '
        n_temp(j) = 0
        IF (best_AC(j,i).GE.1) THEN
            avl_AC = avl_AC + 1
            temp_AC(avl_AC) = AC_list(j)
            n_temp(avl_AC) = best_AC(j,i)
        END IF
    END DO
CC ** print out availability this period **
    IF (avl_IP.GT.avl_AC) THEN
        loop = avl_IP
    ELSE
        loop = avl_AC
    END IF
    WRITE(25,*) ' AVAILABLE RESOURCES'
    WRITE(25,7000) line,line,line,line
    WRITE(25,7200) head3,head2,head6
    WRITE(25,7000) line,line,line,line
    DO j = 1, loop
        IF (j.GT.20.OR.n_temp(j).EQ.0) THEN
            WRITE(25,7300) temp_IP(j)
        ELSE
            WRITE(25,7250) temp_IP(j), temp_AC(j), n_temp(j)
        END IF
    END DO
END DO

```

```

        WRITE(25,*)
        WRITE(25,*)
        WRITE(25,*)
        WRITE(25,*)
        WRITE(25,*)
    END DO

CC ** list unscheduled missions *****
    WRITE(25,*) 'MISSIONS NOT SCHEDULED THIS WEEK:'
    WRITE(25,7000) line,line,line,line
    WRITE(25,7100) head1,head2,blank,head4,head5
    WRITE(25,7000) line,line,line,line
    DO j = 1, n_msns
        IF (rdate(j,3).EQ.0) THEN
            WRITE(25,7100) best_list(j), best_schd(j,4), blank
&                ,best_schd(j,1), best_schd(j,2)
            END IF
        END DO

        WRITE(25,*)
        WRITE(25,*)
        WRITE(25,*)
        WRITE(25,*)
        WRITE(25,*)

CC ** list missions that violate QOT deadlines **
    WRITE(25,*) 'MISSIONS THAT VIOLATE QOT DEADLINES:'
    WRITE(25,7000) line,line,line,line
    WRITE(25,7100) head1,head2,blank,head4,head5
    WRITE(25,7000) line,line,line,line
    DO j = 1, n_msns
        IF (rdate(j,3).EQ.0) THEN
            day_schd = start_date + 7
        ELSE
            day_schd = start_date + INT(rdate(j,3)/2.2)
        END IF
        IF ((rdate(j,6).GE.0).AND.((day_schd-rdate(j,6)).GT.14)) THEN
            WRITE(25,7100) best_list(j), best_schd(j,4), blank
&                ,best_schd(j,1), best_schd(j,2)
            END IF
        END DO

    CLOSE(UNIT=25)

7000 FORMAT(1X,4A19)
7100 FORMAT(1X,A20,4(4X,A10))
7200 FORMAT(1X,A10,15X,2(A10,5X))
7250 FORMAT(1X,A10,15X,A10,5X,I2)
7300 FORMAT(1X,A10)

    RETURN
    END

```

Bibliography

1. Baker, Bruce N. *Introduction to Sequencing and Scheduling*. New York: John Wiley & Sons, 1974.
2. Chan, Yupo. Class Lecture in OPER 767, Networks and Combinatorial Optimization. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, August 1991.
3. Department of the Air Force. *Quality of Training Guidelines*. Edwards AFB: USAF Test Pilot School, January 1991.
4. Department of the Air Force. *Scheduling Instruction Manual*. Edwards AFB: USAF Test Pilot School, January 1991.
5. Department of the Air Force. *Scheduling Procedures for Aircraft and Air/Ground Support*. Air Force Flight Test Center (AFFTC) Regulation 55-15. Edwards AFB: HQ AFFTC, 12 November 1986.
6. Department of the Air Force. *USAF Test Pilot School Curriculum*. Edwards AFB: USAF Test Pilot School, January 1991.
7. French, Simon. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop*. London: Ellis Harwood Ltd, 1982.
8. Hassel, Captain Lisa M. *Investigation of a Zero-One Integer Programming Approach to Automating the Scheduling Process at the USAF Test Pilot School*. MS thesis, AFIT/GOR/ENS/91M-7. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1991.
9. Heald, Lt Col James R., Director, Student Training. Personal interview. USAF Test Pilot School, Edwards AFB, 13 June 1991.
10. Hillier, Fredrick S. and Gerald J. Lieberman. *Introduction to Operations Research*. Oakland, CA: Holden-Day, Inc., 1980.
11. Jain, T. C. and A. M. Hardas. "Heuristic Algorithm for Project Scheduling with Limited Resources," *Journal of Industrial Engineers of India*, 65: 6-9 (July 1984).
12. Mazzola, Joseph B. and Alan W. Neebe. "Resource-Constrained Assignment Scheduling," *Operations Research*, 34: 560-571 (July-August 1986).

13. Moore, James T. Class Notes in OPER 620, Integer Programming. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, November 1991.
14. Norbis, Mario I. and J. MacGreggor Smith. "Two Level Heuristic for the Resource Constrained Scheduling Problem," *International Journal for Production Research*, 24: 1203-1218 (September-October 1986).
15. Patterson, James H. and Glenn W. Roth. "Scheduling a Project Under Multiple Resource Constraints: A Zero-One Programming Approach," *American Institute of Industrial Engineers Transactions*, 8: 449-455 (December 1976).
16. Plebani, Louis J., Jr. "A Heuristic for Multiple Resource Constrained Scheduling," *Production and Inventory Management*, 22: 65-80 (First Quarter, 1981).
17. *SAS/OR Users Guide*. Cary, NC: SAS Institute Inc., 1989.
18. Talbot, F. Brian and James H. Patterson. "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems," *Management Science*, 24: 1163-1174 (July 1978).
19. Thesen, Arne. "Heuristic Scheduling of Activities Under Resource and Precedence Restrictions," *Management Science*, 23: 412-422 (December 1976).
20. Woolsey, E. D. Robert and Huntington S. Swanson. *Operations Research for Immediate Application: A Quick and Dirty Manual*. New York: Harper & Row, Publishers, 1975.

Vita

Captain Gary G. Foster was born July 11, 1964 in Visalia, California. He graduated from Sierra Joint Union High School in Tollhouse, California in 1982 and attended the U.S. Air Force Academy, graduating with a Bachelor of Science in Operations Research in May 1986. Upon graduation, he received a regular commission in the USAF and served his first tour of duty as a test analyst at Headquarters, Air Force Operational Test and Evaluation Center, Kirtland AFB, New Mexico. There he was responsible for directing and evaluating reliability and maintainability aspects of evolving Air Force and DOD space and aircraft systems. He entered the School of Engineering, Air Force Institute of Technology, in August 1990.

Permanent Address: 25037 Auberry Rd
 Clovis, CA 93612
 c/o Mr Charles Foster

March 1992

Master's Thesis (final)

AUTOMATING THE FLIGHT SCHEDULING PROCESS AT
THE USAF TEST PILOT SCHOOL

Gary G. Foster, Capt, USAF

Air Force Institute of Technology
Wright-Patterson AFB OH 45433

AFIT/GOR/ENS/92M-10

USAF/TPS/DOS
Edwards AFB CA 93523

Approved for public release; distribution
unlimited

This study investigated different scheduling solution approaches that could be used to determine weekly flight schedules at the USAF Test Pilot School (TPS). Currently, weekly flight schedules are not developed due to the large quantity of flight scheduling data that must be processed. A weekly flight scheduling approach would reduce the occurrence of scheduling problems (such as unbalanced resource utilization) and improve the communication between the TPS scheduling staff and flight personnel. Since resource-constrained scheduling problems are classified as NP-complete, heuristic methods are usually the most practical approach to solving real-size resource-constrained scheduling problems. This study details a heuristic flight scheduling algorithm designed specifically for the TPS. The computerized version of this heuristic algorithm has demonstrated the capability of producing flight schedules in minutes for weekly flight scheduling problems of realistic size.

resource-constrained scheduling, flight scheduling,
mixed integer programming

141

Unclassified

Unclassified

Unclassified

UL