AFIT/ENS/GOR/99M-08

An Approach for Tasking Allocated Combat Resources to Targets

THESIS

David A. Koewler, 2nd Lieutenant

AFIT/ENS/GOR/99M-08

Approved for public release; distribution unlimited

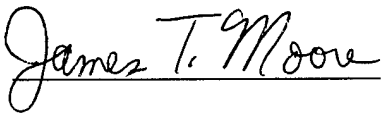
AFIT/ENS/GOR/99M-08

THESIS APPROVAL

Name: David A. Koewler, 2Lt, USAF **Class**: GOR-99M

Thesis Title: An Approach for Tasking Allocated Combat Aircraft to Targets

Defense Date: 11 March 1999

Committee:	**Name/Title/Department**	**Signature**
Advisor	Richard F. Deckro Professor of Operations Research	
Reader	James T. Moore Associate Professor of Operations Research	
Reader	John C. Van Hove, Capt., USAF Senior Logistic Analyst Air Force Logistics Management Agency	

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

AFIT/ENS/GOR/99M-08

An Approach for Tasking Allocated Combat Resources to Targets

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

David A. Koewler, B. S.
2nd Lieutenant, USAF

March, 1999

Acknowledgments

I would like to thank the following people for there help and support of this research: my thesis advisor Dr. Richard Deckro, Dr. James T. Moore, Captain John C. Van Hove, and Major Donald W. Hinton. My thanks to each of you.

David Koewler

Table of Contents

List of Figures

List of Tables

Abstract

Tasking allocated combat aircraft to strike targets is a complicated and time-consuming process for combat planners. Currently, the process of scheduling missions is a two to four day process. To be able to respond quickly to the changing conditions of the battlefield, the military needs to compress the time that this process requires. Despite efforts to develop computer-based tools to automatically plan missions, combat planners still manually perform most of the tasking and scheduling of aircraft and targets. Unfortunately some of the tools currently available are perceived to be complicated and time consuming to use by the planners. They also generate schedules that often require major manual modifications. This means the combat planners often feel it is easier and faster to schedule the missions manually.

With these issues in mind, this research created a prototype-scheduling tool that strikes a balance between ease of use, accurately defining and solving the problem, and generating solutions in an operationally useful amount of time. A combination of concepts from project scheduling, object-oriented programming, heuristics methodology, and genetic algorithms form the basis for the methodology developed in the research. This methodology is then demonstrated in a computer-based tool. While the focus of this research is combat planning, the object-oriented approach developed for project scheduling has applications in both the military and civilian sector. This includes a flexible method for modeling the availability of resources.

1. Introduction

The critical characteristics of the modern battlefield involve continuous increases in speed, range and lethality. The combination of these elements creates a fluid battlespace that is constantly changing and evolving. To achieve and maintain supremacy in this fluid battlespace, the military needs to increase the rate at which it is able to react to changing conditions. One of the critical bottlenecks in being able to respond is the ability of the military to assign aircraft to targets once the targets are identified.

1.1. Background and Purpose

Specialized software has already been developed to help combat planners better perform specific aspects of combat planning. Attempts have also been made to completely automate some aspects of the planning of combat missions with the goal of shortening the ATO generation cycle. For the most part, the automation has been done using greedy heuristics, since current optimal seeking algorithms often can not generate optimal solutions in a timeframe useful to combat planners. Greedy heuristics have the advantage of being able to develop solution alternatives quickly, the proposed solutions are not necessarily optimal. With or without optimal solutions, however, the planner often needs to make major adjustments to the solution generated as conditions change in a dynamic battlespace (Van Hove: 1).

Despite the development of these software tools, most combat planners still assign aircraft to targets without the aid of computers (Hinton, Dec 1998). There are several reasons for this, the first of which, as mentioned before, is because the schedules developed often need to be significantly altered by the planner. However, the primary

reason is because the planning tools are not perceived to be user friendly, with data entry for a single target often taking several minutes (Hinton, Dec 1998).

The purpose of this thesis is to develop a framework for a computer-based tool that helps combat planners to task allocated strike aircraft to identified targets. The developed tool is a balance between the following three goals: 1) be user-friendly, 2) accurately define the scheduling problem, and 3) generate good solutions in an operationally useful amount of time. The purpose of the computer based tool is to demonstrate the applicability of the methodology and to demonstrate to the combat planners a possible tool that can make their job easier.

1.2. Research Problem

The research in this thesis develops a framework for a user-friendly, computer-based tool to aid combat planners task combat resources to targets. The goal of the tool is to provide combat planners with a good starting solution for tasking and scheduling allocated air combat resources to targets. A significant portion of the research involves developing the methodology for generating "good" solutions for the problem of tasking allocated combat resources to targets on a target nomination list. The methodology is a combination of several concepts, including: project scheduling, object-oriented programming and genetic algorithms. The concepts are defined and explained in the literature review chapter.

1.3. Research Objectives

The research accomplishes the following objectives:

1. Formulate the combat planning problem to include some of the following:

a. Tracking aircraft availability across continuous time.

b. Explicitly incorporate wave constraints.

c. Limit the times at which squadrons can perform missions during the flying day that are specific to the individual squadrons.

d. Group the missions flown by a squadron into Goes, where the takeoffs for all missions in a Go are within a set window of time. The number of aircraft available in each Go is variable.

2. Formulate the methodology for solving the problem using a project scheduling and an object-oriented perspective, with the focus of setting up data structures for use with heuristics.
3. Use heuristics to develop starting solutions and evolutionary algorithms to improve the solutions.
4. Develop the framework for a computer-based tool that can be used by combat planners.

1.4. Assumptions

The developed methodology relies upon the following assumptions:

1. The necessary munitions are available for a combat resource.
2. It is acceptable to plan and schedule the attack missions first and to then schedule the support and refueling missions around the generated strike missions.
3. The duration of a mission, for a given squadron/target pairing, can be assigned a specific time value.
4. The airfield can accommodate the solution generated.

1.5. Approach

In order to develop a prototype computer based tool that is capable of solving combat planning problems, a methodology for solving combat planning problems is developed. First, the problem is divided into a combat planning data structure and a combat scheduling data structure. The purpose of the combat planning data structure is to facilitate the process of entering combat planning information by the combat planners, which is accomplish through a Graphical User Interface (GUI). The purpose of the combat scheduling data structure is to be able to build and evaluate different schedules that are defined by the combat planning data structure. Fundamental ideas of object-oriented programming and project scheduling are used in the development of this data structure. Instead of developing variables and equations to model the problem, various objects are developed that model the different sections of the problem. The building of a schedule then involves the manipulation of these different objects (this approach offers some distinct advantages that are discussed in Chapter 3). Next, a solution engine is developed to search the solution space for different possible schedules. Genetic algorithm and heuristic techniques are used to accomplish this.

1.6. Summary

This chapter presented an overview of the focus of the thesis. Chapter 2 presents a review of the literature pertaining to the traditional approach of modeling the problem using project management techniques. It also presents some object-oriented programming and genetic algorithms concepts that are used to develop the methodology in this research. Chapter 3 covers the development of the methodology and the prototype computer based, combat scheduling tool. Chapter 4 presents a case study to demonstrate

the capabilities of the methodology and the scheduling tool. Chapter 5 provides a summary of the research, the significant contributions, and recommendations for future work.

2. Literature Review

The purpose of this thesis is to develop a methodology for solving combat planning problems and to develop the framework for a computer-based tool that aides combat planners task allocated strike aircraft to identified targets. To this end, the following areas were researched in the literature. First, an overview is presented of the overall process within which aircraft are tasked to targets. Next, the project scheduling section shows the primary models used by Van Hove to develop his formulation of combat planning as a project scheduling problem, followed by the actual formulation developed. An overview of object-oriented programming is then presented. The final section covers genetic algorithms and some of the research on using genetic algorithms to solve project-scheduling problems.

2.1. ATO Flight Scheduling

An Air Operations Center (AOC) is the operational facility that provides planning, direction, and control over air resources committed to a theater. One of the six functional elements of an AOC is the Combat Plans Division (CPD) which daily develops the detailed plans for the application of air resources. Each daily plan is an Air Tasking Order (ATO), developed based on the Joint Force Commander (JFC) objectives and the air component commander's guidance (AFI 13: 28-29).

The planning cycle for an ATO is a three to four day process that concludes with the execution of the ATO. Often, three to four of the planning cycles are occurring at the same time, each at a different level of completion. Thus, as one ATO is being executed, there may already be two or three other ATOs at different stages of development (AFI 13: 28).

Based on campaign objectives, the CPD makes recommendations to Joint Forces Air Component Commands (JFACC) for the apportionment of air resources. The apportionment defines the percentage and/or priority that should be devoted to the different air operations for a given period of time. The JFACC in turn makes recommendations to the Joint Forces Commander (JFC), who makes the ultimate apportionment decision. The CPD, based on the finalized apportionments, allocates specific air assets by type of aircraft and the total number of sorties, to each operation or task (AFI 13: 28-30).

2.2. Project Scheduling

A project is defined to be a finite set of activities scheduled according to precedence requirements between the activities. The objective of a project-scheduling problem is usually to minimize either the cost or the completion time of the project.

2.2.1. Gantt Charts

Henry L. Gantt developed the Gantt chart during World War I. A type of bar chart, the Gantt Chart is the most widely used management tool for scheduling and control (Shtub: 322). The format of the Gantt chart is also similar to the grease boards used by combat planners to plan missions. The horizontal axis of the Gantt chart represents time, and the vertical axis represents an enumeration of the activities or some resource through which activities are completed. Figure 1 illustrates a simple example where the activities are enumerated on the vertical axis. Figure 2 illustrates an example where the vertical axis is the resource being used for the activity.

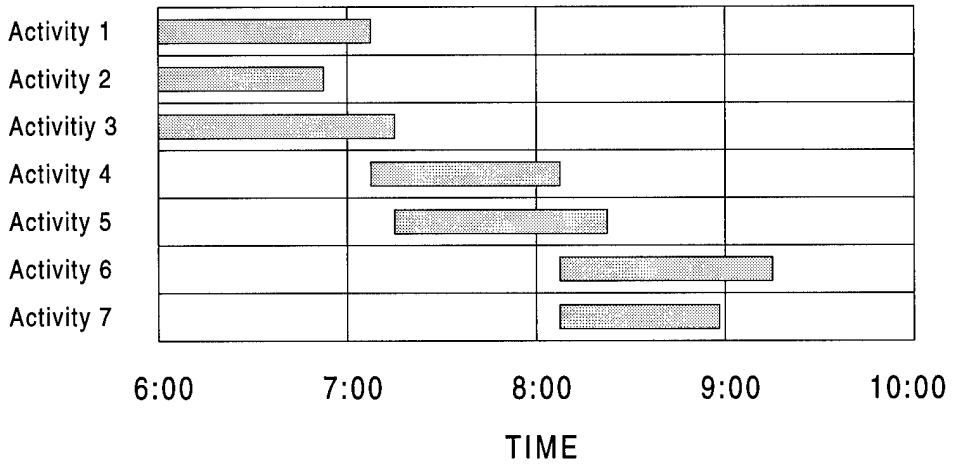


Figure 1 Gantt Chart: Example of activities on the y-axis

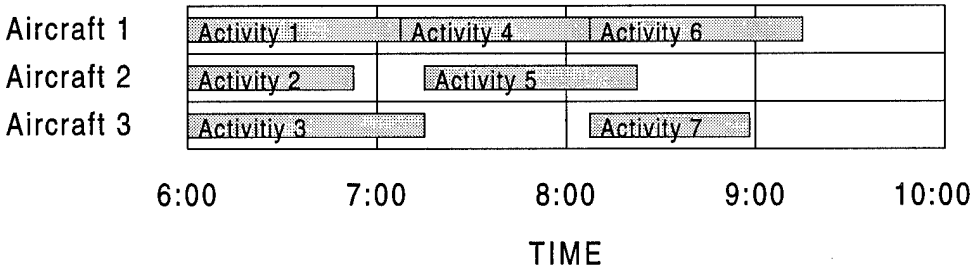


Figure 2 Gantt Chart: Example of resources on the y-axis

The bars in Figures 1 and 2 represent the different activities that are scheduled. The left end of the bar indicates the start time of the activity while the right side indicates the time the activity is finished.

2.2.2. The Resource Constrained Project Scheduling Problem

The Resource Constrained Project Scheduling Problem (RCPSP) is an expanded formulation of the classic project-scheduling model. The RCPSP accounts for limited project resources, which are consumed by the activities. The following RCPSP formulation is adapted from a formulation presented by Pritsker, Watters, and Wolfe

(Pritsker: 93-101). The parameters, variables, and equations used in the model are shown in Figure 3.

Parameters:

A	the set of all activities
K	the set of all resources
P	the set of all activity precedence pairs
g	the project deadline
τ_i	the duration of activity i
e_i	the earliest completion time for activity i
l_i	the latest completion time for activity i
r_{ik}	the amount of resource k required by activity i
R_{jk}	the amount of resource k available in period j

Variables:

x_{it} equals 1 if activity i finishes in period t; 0 otherwise

$$\text{Minimize } \sum_{t=e_n}^{l_n} t x_{nt} - \sum_{t=e_1}^{l_1} t x_{1t} \qquad (1)$$

$$\text{Subject to } \sum_{t=e_n}^{l_n} t x_{nt} - \sum_{t=e_i}^{l_i} t x_{it} \geq \tau_n \quad \forall (i,n) \in P \qquad (2)$$

$$\sum_{i \in A} \sum_{t=j}^{j+\tau_i-1} r_{ik} x_{it} \leq R_{jk} \quad \forall k \in K \text{ and } j = 1,\ldots,g \qquad (3)$$

$$\sum_{t=e_i}^{l_i} x_{it} = 1 \quad \forall i \in A \qquad (4)$$

$$x_{it} \in \{0,1\} \quad \forall i \in A \text{ and } t = 1,\ldots,g \qquad (5)$$

Figure 3 Parameters, Variables, and Equations for RCPSP

For this RCPSP formulation, as well as other traditional formulations that involve resource usage, a series of binary decision variables indicates whether or not a resource is in use over a unit of time. The addition of binary variables significantly increases the computational complexity of the problem. This is due to two main reasons. The first is that the number of variables increases as multiples of: the number of activities, and the

resolution and length of time being modeled. The second reason is that in containing binary decision variables, strictly linear programming solvers can not be used to solve the problem. In addition, the total number of possible solutions is equal to 2^n, where n is the number of binary variables.

The objective function, Equation (1), of this formulation minimizes the project completion time. Equation (2) enforces the activity-timing requirement that completes activity *i* before the start of activity *j*. Equation (3) limits the consumption of resources, and Equation (4) ensures that a resource can be tasked to only one activity in a given time period.

2.2.3. The Multi-Modal Resource Constrained Project Scheduling Problem

If an activity is multi-modal, it means that there are different options for how the activity is executed, each of which is a different mode. The mode the activity is executed in determines how many and which resources are used. The mode can also determine the duration of the activity. The multi-modal resource constrained project scheduling problem (MMRCPSP) is an expanded form of the RCPSP that allows the resources consumed by an activity to vary by the mode in which it is executed. Figure 4 lists the variables used in a formulation of the MMRCPSP, followed by the formulation itself (Boctor: 350).

Parameters

A the set of all activities
K the set of all resources
P the set of all activity precedence pairs
M_i the set of all execution modes for activity i
τ_{im} the duration of activity i in mode m
e_{im} the earliest completion time for activity i
l_{im} the latest completion time for activity i
r_{imk} the amount of resource k for activity i in mode m
R_{jk} the amount of resource k available in period j

Variables

x_{imt} = 1 if activity i finishes in period t using mode m

Minimize $$\sum_{m\in M_n}\sum_{t=e_n}^{l_n} t x_{nmt} - \sum_{m\in M_1}\sum_{t=e_1}^{l_1} t x_{1mt} \quad (6)$$

Subject to: $$\sum_{m\in Mj}\sum_{t=e_j}^{l_j}(t-\tau_{jm})x_{jmt} - \sum_{m\in M_i}\sum_{t=e_i}^{l_i} t x_{imt} \geq 0 \quad \forall(i,j)\in P \quad (7)$$

$$\sum_{i\in A}\sum_{m\in M_i}\sum_{t=j}^{j+\tau_i-1} r_{imk}x_{imt} \leq R_{jk} \quad \forall(j,k) \quad (8)$$

$$\sum_{m\in M_i}\sum_{t=e_i}^{l_i} x_{imt} = 1 \quad \forall i\in A \quad (9)$$

$$x_{imt} \in \{0,1\} \quad \forall(i,m,t) \quad (10)$$

Figure 4 Parameters, Variables, and Equations for the MMRCPSP

The binary decision variable, x_{imt}, is an expanded version of the decision variable x_{it} in RCPSP with x_{nmt} being the terminal node. The variable is expanded to account for the execution mode of the activity. The variable has a value of one if and only if activity i is executed in mode m and completed in period t. Equation (9) forces this to occur. The variable 'n' is defined as the index of the terminal activity. Equation (7) drives the pair precedence relationships between activities, and Equation (8) accounts for resource usage. The objective function, Equation (6), attempts to minimize the project's

completion time. This is done by subtracting the time the first activity is completed from the time the last activity is completed.

2.3. The Generalized Resource Constrained Project Scheduling Problem

The advantage of the Generalized Resource Constrained Project Scheduling Problem (GRCPSP) over the RCPSP is it has the ability to model a greater variety of precedence relationships. The GRCPSP uses four different constraint formulations to model all possible precedence relations between the start and completion of two activities. Figure 5 illustrates the four different types of generalized precedence constraint types, using a Gantt chart format.

H_1 start-start activity pair relations with a lag of SS_{ij}

H_2 start-finish activity pair relations with a lag of SF_{ij}

H_3 finish-start activity pair relations with a lag of FS_{ij}

H_4 finish-finish activity pair relations with a lag of FF_{ij}

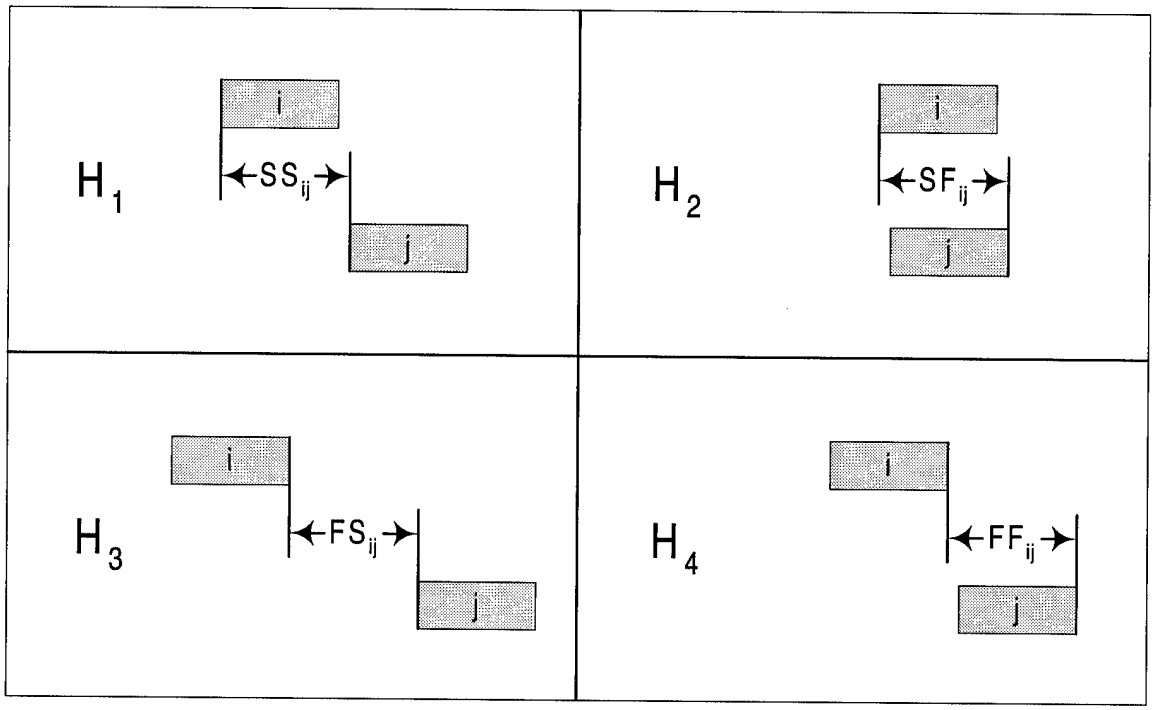


Figure 5 Generalized Precedence Types

These four types of constraints can capture any possible precedence relationship between either the start or finish of two activities. The H_1 type precedence constraint causes activity j to start some minimum amount of lag time after the start of activity i. The H_2 type allows activity j to finish a minimum lag time after the start of activity i. The H_3 type allows activity j to start some minimum lag time after activity i is completed. The H_4 type allows activity j to finish at a minimum, some lag time after activity i is completed. These four precedence relationships can also be set as maximums, in which case the lag time becomes the maximum amount of lag time allowed. The GRCPSP model in Figure 6 applies these four types of precedence relations (Chretienne: 260-262).

Minimize $y_n - y_1$ (11)

Subject to: $\sum_{t=e_i}^{l_i} t x_{it} = y_i \quad \forall i \in A$ (12)

$y_j - y_i \geq SS_{ij} + \tau_j - \tau_i \quad \forall (i,j) \in H_1$ (13)

$y_j - y_i \geq SF_{ij} - \tau_i \quad \forall (i,j) \in H_2$ (14)

$y_j - y_i \geq FS_{ij} + \tau_j \quad \forall (i,j) \in H_3$ (15)

$y_j - y_i \geq FF_{ij} \quad \forall (i,j) \in H_1$ (16)

$\sum_{i \in A} \sum_{t=j}^{j+\tau_i - 1} r_{ik} x_{it} \leq R_{jk} \quad \forall (i,j) \in \forall k \in K \text{ and } j = 1,\ldots,g$ (17)

$\sum_{t=e_i}^{l_i} x_{it} = 1 \quad \forall i \in A$ (18)

$x_{it} \in \{0,1\} \quad \forall i \in A \text{ and } t = 1,\ldots,g$ (19)

$y_i \geq 0 \quad \forall i \in A$ (20)

Figure 6 Equations of the GRCPSP

Equation (12) defines the continuous variable y_i to be the completion time of activity *i*. Equations (13-16) are included to simplify the precedence constraints. The objective function of Equation (11) minimizes the time between the completion of activity n, and the completion of the first activity.

2.4. Van Hove's Combat Planning Model Formulation

Since combat planning requires mission scheduling, mission precedence, and resource accounting, Van Hove developed a new variant of the Resource Constrained Project Scheduling Problem (RCPSP) formulation. This variant is a combination of

several other RCPSP variants. The new model formulation includes multi-modal activities, doubly constrained resources, and minimal time lag precedence constraints. It is referred to as the multi-modal generalized resource constrained project-scheduling problem (MMGRCPSP). Doubly constrained resource refers to how a resource can only be used a set number of times, and that it can be in use by at most only one activity within any unit of time.

2.4.1. Definitions

Table 1 lists the associations used to apply the project-scheduling model.

Table 1 Definitions

Project Scheduling Terms	Combat Planning Nomenclature
activities	targets
mode	weaponeering option
resource	squadrons
precedence constraint	mission timing requirements
asset	a single aircraft

2.4.2. Multi-Modal Activities

The activities for the model are the targets identified to attack during the planning period in question. The target nomination list (TNL) determines the activities in an air campaign-planning problem. For every target on the TNL, a set of weaponeering options is compiled, based on availability of aircraft, crews and munitions, as well as an acceptable level of success, or probability of kill (PK). Each option is composed of a single type of air asset. Each mode for an activity is associated with a possible squadron that could be tasked to attack the target. Modes are developed from the weaponeering

options, and since the same type of air assets can be located at more than one location, one option can correspond to two or more modes.

The duration of an activity is dependent on the location of the activity's target, and the type and location of the resources required for the mode. A simplified activity duration can be determined for each mode using the following three values: 1) The time to go from the base to the target, and back again. These times are based on the location of the base, the location of the target, and the nominal air speed of the aircraft type. 2) The window of time the air assets has exclusive access to the target. 3) The regeneration time between when an aircraft lands and when it is again ready to takeoff for a new mission. Currently at the force level, detailed routing is not performed to take into account terrain or enemy air defenses. The unit level does the detailed routing which usually involves different throttle settings to reach the target at the assigned time.

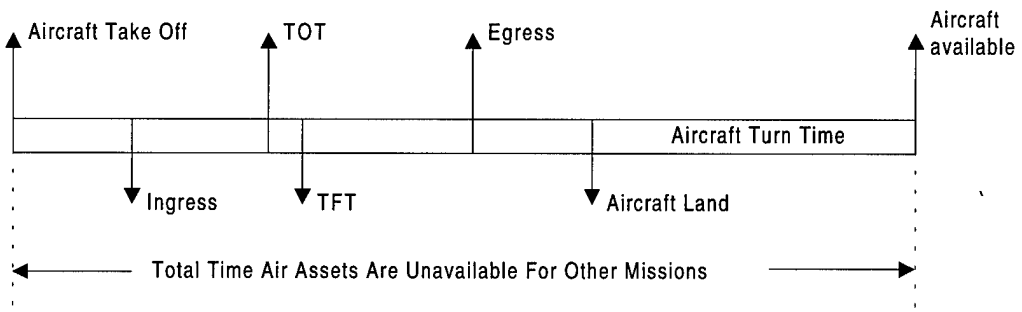


Figure 7 Activity Milestones

Figure 7 reflects the mission legs used in each activity. The following milestones mark the start and end of the different legs: 1) aircraft take off, 2) ingress of enemy territory, 3) TOT (time over target), 4) TFT (time off target), 5) egress of enemy territory, 6) aircraft lands, and 7) the aircraft is available for a new mission. As can be seen in Figure 7, an activity starts when the aircraft takes off, and is over when the aircraft is

available to take off again for another activity. The ingress and egress milestones mark when the aircraft(s) transition to and from enemy territory. Time on target (TOT) and time off target (TFT) mark a window of opportunity for the aircraft to attack the target. This window is when a mission has exclusive access to the target area without the chance of interference with other missions attacking the same target. These durations are mode dependent and all activities have the six duration components as shown in Figure 7.

2.4.3. Doubly Constrained Resources

This section presents the framework used to model air assets as resources in the MMGRCPSP formulation for campaign planning. Resources are generated by dividing aircraft by both type and location. Thus, a resource reflects a specific type of air asset at a specific location or base.

The resources are doubly constrained in that they are both a renewable resource and a nonrenewable resource. The renewable aspect of a resource is the number of that type of asset at the location and available for use during the ATO planning period. This limits the number of assets that can be tasked from the resource at any one time. The nonrenewable aspect of a resource is the maximum number of sorties the resource can complete within the planning period.

2.4.4. Generalized Precedence Constraints

Precedence constraints are added to the MMGRCPSP formulation to enforce the timing restrictions between various activities. The standard finish-to-start activity precedence relations are not flexible enough for the mission sequencing of air combat planning. This is because the standard precedence constraint between two activities

would require all air assets from the first mission to land and be serviced before the assets for the second mission could takeoff. Therefore, generalized precedence constraints are used.

Generalized precedence constraints allow a minimum and/or maximum time lag between the start of two different activities. If the constraint is based upon the start or completion of legs of the activity, then a different lag time is calculated for each different combination of modes for the two activities. A common example is if an enemy air defense site must be attacked before the aircraft for another mission are allowed to ingress enemy air space.

2.4.5. The Complete Formulation

The complete MMGRCPSP developed by Van Hove follows (Van Hove: 49).

Parameters:

A	the set of all activities
d	the index of terminal activity
e_{im}	the earliest completion time for activity *i in mode m*
l_{im}	the latest completion time for activity *i in mode m*
M_i	the set of all execution modes for activity i
τ_{im}	the duration of activity i in mode m
S_i	the set of generalized successors of activity i
Δ_{ijmn}	the minimum lag between the start time of activity i in mode m and the start time of activity $j \in S_i$ in mode n
K	the set of all double constrained resources
r_{imk}	the amount of resource k required by activity i when being executed in mode m
R_k	the per period availability of resource k
N_k	the total amount of resource k available
g	the deadline for the project under consideration

Variables:

x_{imt} = 1 if activity i starts in period t and is executed in mode m

$$\text{Minimize} \quad \sum_{m \in M_d} \sum_{t=e_{dm}}^{l_{dm}} (t + \tau_{dm}) x_{dmt} \tag{21}$$

Subject to:

$$\sum_{i \in A} \sum_{m \in M_i} \sum_{t=e_{im}}^{l_{im}} r_{imk} x_{imt} \leq N_k \quad \forall k \in K \tag{22}$$

$$\sum_{i \in A} \sum_{m \in M_i} \sum_{t=j-\tau_{im}+1}^{j} r_{imk} x_{imt} \leq R_k \quad \forall k \in K, j = 1 \ldots g \tag{23}$$

$$\sum_{n \in M_j} \sum_{t=e_{jn}}^{l_{jn}} (\Delta_{ijm'n} - t) x_{jnt} + \sum_{t=e_{im'}}^{l_{im'}} t x_{im't} \leq \sum_{m \in (M_i \setminus m')} \sum_{t=e_{im}}^{l_{im}} g x_{imt} \quad \forall i \in A, \forall j \in S_i, m' \in M_i \tag{24}$$

$$\sum_{m \in M_i} \sum_{t=e_{im}}^{l_{im}} x_{imt} = 1 \quad \forall i \in A \tag{25}$$

$$x_{imt} \in \{0,1\} \quad \forall (i,m,t) \tag{26}$$

Figure 8 Variables, Parameters, and Equations for the MMGRCPSP

The objective function, Equation (21), minimizes the make span of the generated schedule. When applied to combat planning it minimizes the length of time the air assets are over enemy territory. Equation (22) models the nonrenewable aspect of the

resources, while Equation (23) models the renewable number of aircraft. Equation (24) enforces the precedence constraints. Equation (25) drives the decision variable, x_{imt} to have a value of one for an activity, in only one mode, and for only one completion time, while Equation (26) makes the decision variables binary. To solve a combat planning problem using this formulation, Van Hove used a hybrid variant of the integer programming decomposition algorithm developed in his dissertation. The problem in his case study contained one hundred activities, four different resources, and numerous precedence constraints. Using a 167MHZ UltraSPARC, Van Hove was able to generate the ten best scheduling solutions in approximately forty-five minutes (Van Hove: 142)

2.5. Object Oriented Programming

This section presents the basic concepts of Object-Oriented Programming (OOP). These concepts form the framework for the formulation and solution of the models developed. One of the advantages of object-oriented programming is that it provides a framework by which large problems can be broken up into smaller problems. Once solved, the smaller problem solutions can be integrated to solve the original problem. However, one of the disadvantages of OOP is that it can involve a substantial increase in the amount of code required to solve the problem.

2.5.1. Concept of Objects and Classes

An object is a real world person, place, or thing which is to be represented on a computer. For OOP, an object represents what a computer needs to know and do about the person, place, or thing (Coad: 4). A class is a grouping of objects and this grouping is based on common characteristics. Thus, all objects that belong to a class have a set of

common characteristics. For example, while there are a large variety of different aircraft, they all have some of the same characteristics. Therefore, they could all belong to the same aircraft class.

2.5.2. Basic Terminology

This section explores two OOP concepts: Abstraction and Encapsulation.

2.5.2.1 Abstraction

To represent something from the real world on a computer, it is necessary to extract the essential characteristics. The extracted data is how the analyst represents the object on a computer. Abstraction is the process by which only the appropriate characteristics that describe the object, and are necessary for the task, remain (Linden: 31). The task refers to the overall purpose of the program. If the purpose of the program is inventorying cars in the sales lot, the type of relevant information is different from the information needed to keep track of the maintenance of a car.

2.5.2.2 Encapsulation

Encapsulation is the step following abstraction that accounts not only for the characteristics of an object, but also the operations performed on the object. An object contains both the data that characterizes its real world counterpart and the set of operations that the system can perform on the object (Linden: 32). Often the functions are the only way to modify and retrieve the information contained within the objects. This allows an object to control the manner in which its information is modified and retrieved.

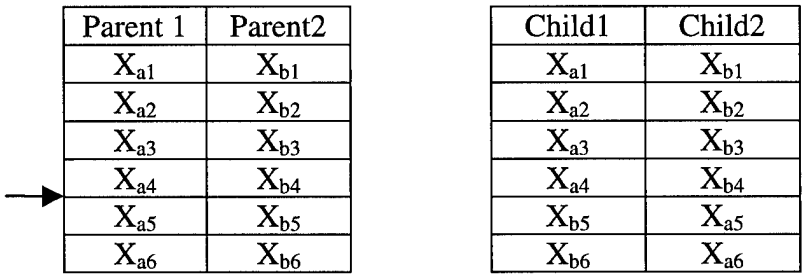
2.6. Genetic Algorithms

A genetic algorithm is a problem solving technique that is based on the ideas of genetics. Just as DNA defines the characteristics of an organism, an electronic genotype defines the characteristics of a solution to the problem being solved (Chambers:1). Either the initial population of electronic genotypes can be produced randomly or it can represent solutions generated through other solving techniques. Gene crossovers and mutations are performed on the electronic genotype to create new solutions. The fitness of the different solutions is determined through a scoring process that evaluates the solution represented by an electronic genotype. To maintain a consistent population, a selection process is then used to select the highest scoring genotypes, with the remaining genotypes being removed. The highest scoring genotypes are used to create the next generation of genotypes. The process is repeated as the highest scoring genotypes are combined using crossovers to create new genotypes. The stopping criteria for this process is either the convergence of the highest scoring genotypes across several generations or the reaching of a set time limit. The electronic genotype is traditionally represented by a string of zeros and ones, which are mapped to a possible solution.

2.6.1. Crossovers

There are many crossover strategies. Selection of a crossover strategy can be an entire study in and of itself. It is through the process of crossovers that new genotypes are produced. An example of a crossover is as follows. Two genotypes are selected, these become the parents, and a copy of each is made. At a random location along the copies, the remaining information is switched between the two genotypes. If two locations are used, then the information between the two spots is swapped. In this

manner, the two copies become the children of the two parents. Figure 9 illustrates an example of how two "parent" genotypes can be used to produce two "children" genotypes. The X variables indicate the different values stored within the elements of the genotypes. The arrow on the left side indicates the position where the crossover was applied. As can be seen on the right side of the figure, the first four elements of a "child" are from one parent, while the last two elements are from the other parent.



Parent 1	Parent2
X_{a1}	X_{b1}
X_{a2}	X_{b2}
X_{a3}	X_{b3}
X_{a4}	X_{b4}
X_{a5}	X_{b5}
X_{a6}	X_{b6}

Child1	Child2
X_{a1}	X_{b1}
X_{a2}	X_{b2}
X_{a3}	X_{b3}
X_{a4}	X_{b4}
X_{b5}	X_{a5}
X_{b6}	X_{a6}

Figure 9 Crossover Example

2.6.2. Mutations

The process of mutation is how variations are continually added to the genotype population between generations. The simplest mutation operator involves selecting an element of a genotype, and randomly changing its value. Two other mutations involve swapping two elements of a "child" genotype and removing an element from the genotype and then randomly inserting it.

2.6.3. Hartman's Approach

This section explores a genetic algorithm approach, developed by Hartman, for solving multi-mode resource constrained project scheduling problems. Hartman defined the set of genotypes to be searched to consist of the precedence feasible activity

sequences and all mode combinations (Hartman: 2). Thus, the genotype no longer consists of a sequence of ones and zeros; instead, it now consists of the order in which activities are scheduled and the associated mode of the activity. The precedence feasible aspect requires activities constrained to occur after another activity to be listed after the activity. Activities are scheduled in the order prescribed by the sequence, in the given mode, at the earliest feasible time.

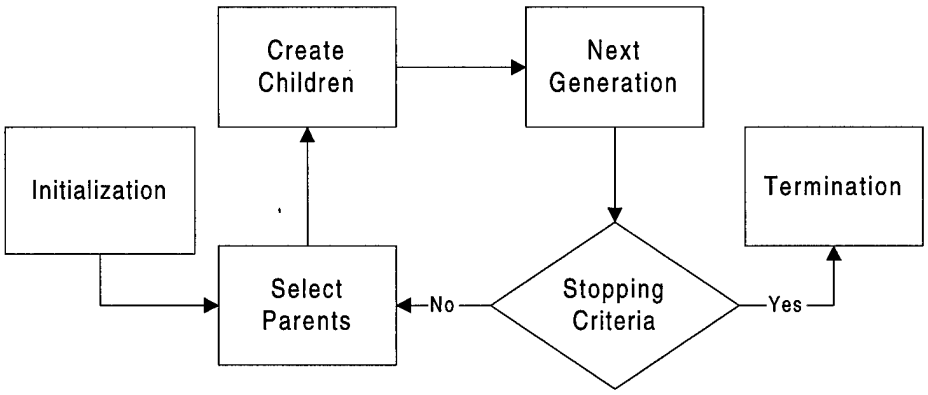


Figure 10 Flow Diagram of Hartman's Genetic Algorithm

Figure 10 shows the main steps of the genetic algorithm used by Hartman to solve scheduling problems. The "Initialization" step involved randomly generating a certain population size of individuals, and measuring the fitness of each. The "Select Parent" step involved randomly dividing the population into pairs of parents. The "Create Children" step loops through all pairs of parents and creates two children, through a crossover operator. A mutation operator is applied to each 'child.' The "Next Generation" step, involves adding the new children to the parent population, doubling its size. Next, a selection operator is applied to reduce the population back to the initial size. The "Stopping Criteria" step involves either reaching the set number of generations or a

time limit. The "Termination" step picks the individual that that has the best fitness value (Hartman: 5).

2.7. Summary

This chapter reviewed the literature that provides a background for the research in this thesis. The review focused on the areas of project scheduling, object-oriented programming, and genetic algorithms. Chapter 3 covers the development of the methodology and the prototype computer based, combat scheduling tool. Chapter 4 presents a case study to demonstrate the capabilities of the methodology and the scheduling tool. Chapter 5 provides a summary of the research, the significant contributions, and recommendations for future work.

3. Methodology

The purpose of this chapter is to provide the methodology used to accomplish the research objectives outlined in Chapter 1. The overall research goals are to develop a new methodology for solving combat planning problems, and to develop the framework for a computer-based tool that aids combat planners in tasking air resources to targets. To do this, the combat planning problem is first formulated. The next step involved determining the solution space the solver searches for scheduling solutions. A generalized framework for a combat planning tool is shown in Figure 11. Based on the framework, two different object-oriented data structures are simultaneously developed, one for combat planning, and the other for combat scheduling. The combat planning data structure models the type and format of information a combat planner uses. The data structure also includes the development of a Graphical User Interface (GUI) to allow the combat planners to enter problem data. The combat scheduling data structure models the problem of scheduling air assets to targets, using fundamental ideas of object-oriented programming and project scheduling to develop a data structure that can build and evaluate different schedules. A solution engine is then developed that is capable of generating solutions of large combat planning problems in an operationally useful amount of time. The final step of the process is to present the solutions generated by the solver in a format that is understandable to the combat planners.

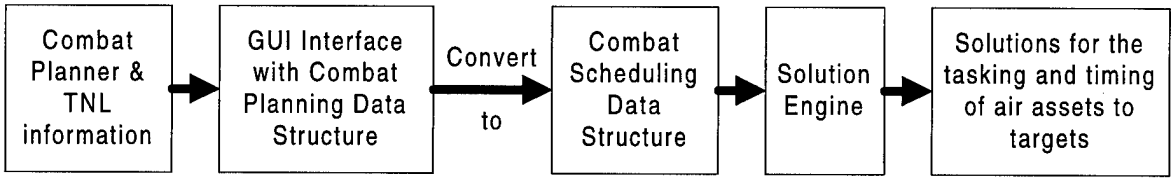


Figure 11 Combat Planning Tool

3.1. The Problem

This research examines the problem where there is a set of various targets to strike and a limited number of allocated aircraft, of different capabilities, to carry out the strike missions against these targets. As covered in Chapter 2, the term mission reflects the entire process by which aircraft attack a single target; the mission starts at the takeoff of the aircraft and ends when the aircraft are ready to takeoff for another mission. The recovery phase of a mission also captures any preparation time required before the start of the next mission. Figure 12 outlines the combat planning problem data that is captured by the formulation.

- Bases
  - Location
  - Squadrons
    - Aircraft Type
    - Number Of Aircraft
    - Flying Day
    - Goes

- Targets
  - Location
  - Weaponeering Options
  - Possible Scheduling Constraints
    - Waves
    - Individual coordinations between pairs of missions
    - Window to attack the target

Figure 12 Structure of Combat Planning Problem Data

The problem data is divided into two main sections: the squadrons of aircraft located at the bases and the targets that need to be attacked using the aircraft in the squadrons.

3.1.1. The Air Squadrons

As shown in Figure 13, there are different bases in the theater, each of which contains one or more squadrons. The location of the bases is used in part to determine the durations of the missions the squadrons are tasked to perform. Since only strike missions are being scheduled, the only aircraft included in this data set are those that are allocated specifically for strike missions and should be considered for assignment to the given targets.

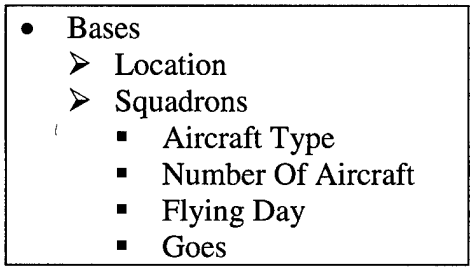


Figure 13 The Squadrons

A squadron is composed of a group of the same type of aircraft, located at the same base. Each squadron has its own unique flying day, which defines a window of time within which it flies missions. The aircraft type indicates to the combat planner what type of targets the squadron can be tasked to attack.

Squadrons fly missions that are grouped into Goes. A Go is a group of missions flown by a squadron that all start within a takeoff window, usually defined to be one hour. The start of a Go is defined to be when the first mission of the Go starts, and the Go ends when all aircraft used within the Go are again available. The first Go starts at or after the start of the flying day. The start of the remaining Goes must be on or after the time that the previous Go ends minus the takeoff window. For example, the time the first Go ends, minus the takeoff window equals the earliest time the second Go can start. This

allows any aircraft used in the previous Go to be available for the next Go. This also allows the possibility for Goes to overlap, which requires the availability of individual aircraft to be tracked by the model. If aircraft availability is not tracked, then aircraft could be scheduled for missions in the next Go, before they finish missions from the previous Go.

Due to maintenance and possible attrition, the maximum number of aircraft that are planned to be available decreases throughout the flying day. For example, a squadron of 20 aircraft might have a maximum number of 18 aircraft for the first go and 16 aircraft for the second go. Since the availability of aircraft is important, when the maximum number of aircraft decreases between two consecutive goes, the aircraft from the previous go that are available first are removed as possibilities for the next Go. The number removed is equal to the decrease between the two Goes. This provides for the worst case scenario for which airplanes are no longer available between Goes. For example, it is assumed that between the first and second Goes, there will be a decrease in availability of two aircraft. If this is the case, then the first two aircraft that return from missions in the first Go are assumed unavailable. This is the worst case scenario for losing the availability of two planes between Goes. This is because between Goes the squadron can not pick which aircraft will not be available for the next Go. If the aircraft are not picked in this manner, then a schedule could be produced that would require aircraft when none are available, even if the loss of aircraft falls within the parameters given between Goes.

3.1.2. The Targets

The allocated strike aircraft are the aircraft that are tasked to attack the targets. For the purpose of this research, a mission is defined to be composed of aircraft from

only one squadron, and the aircraft of a mission attack only one target from the target list. In addition, each target on the target list can only have one scheduled mission against it. Figure 14 contains the target information to be included in the formulation.

- Targets
  - Location
  - Weaponeering Options
  - Possible Scheduling Constraints
    - Waves
    - Individual coordinations between pairs of missions
    - Window to attack the target

Figure 14 Target Information

Weaponeering options define which type and number of aircraft are necessary to attack a target. The capabilities of the aircraft, the type of target, and the target damage required are used to determine the weaponeering options. Each target has a list of possible squadrons that are options for attacking the target, only one of which is chosen. Each possible option is considered to be a different mode for the activity. Thus, the different modes represent the different combat resources that can be used to attack the target.

Each mode has some information that is unique to it, the first of which is the squadron it represents attacking the target, and the number of aircraft needed for the mission. Earliest and latest start times for the mission can be different for each mode. The default value for the takeoff window is the flying day for the squadron the mode represents. This window is narrowed when a target needs to be attacked at a specific time or within a specific window of time. Since different squadrons can take a different

amount of time to reach the target, the set takeoff window is dependent on the weaponeering option.

Since the air plan works in coordination with other military forces not included in the schedule, specific targets often need to be attacked within specific windows. Another type of scheduling constraint is where the combat planners determine the general pattern by which air assets attack targets; for example: attacking the enemy air defenses before sending the bombers over the region. To accomplish this task, a target can also belong to a wave. Waves are used when the milestone of one group of missions needs to be accomplished before the milestone of the next group of missions can occur. An example of this is if the first wave consists of a set of missions that attack air defense targets. Once the targets are attacked, then the missions for the next wave can ingress enemy territory.

To define when a wave starts and ends, start and end milestones are defined for each wave. The wave starts when the earliest time of any start milestone occurs in any of the missions contained in the wave. The wave ends at the latest time the end milestone occurs.

For the previous example, the first wave would start at the first takeoff and end when all the air defense targets are attacked, although the Go (as defined in the Squadron section) is still going on. The second wave starts when the strike aircraft ingress enemy territory and could end when the aircraft have landed. In this way the aircraft in the second wave can start to ingress enemy territory as soon as all the air defense targets are attacked, but not before.

While the missions of different waves can overlap, the waves themselves do not overlap. The earliest a wave can start is the time that the previous wave ends. Targets that are not defined to be in any wave are not directly constrained by the waves.

To allow further coordination between other missions, precedence constraints can be defined for a mission. A mission's precedence constraints can only constrain the mission to missions that will be scheduled before it. An example is if mission #2 can only attack the target after mission #1 attacks a target. Mission #2 can not be scheduled until mission #1 is scheduled. Once mission #1 is scheduled, then mission #2 can be scheduled. The start time of mission #2 will be constrained so that it can not attack the target until after the time that mission #1 attacks the target. The precedence constraints can be set between any combination of the milestones of the two missions.

3.2. The Solution Space

The solution space this research searches is any combination of 1) All mission/mode combinations and 2) Within each squadron, all mission/Go combinations. However, to simplify the ability to represent a schedule as a list of activities and modes, the solution space consists of all the precedence feasible mission sequences, with all mode combinations, some of which map to the same schedule. A precedence feasible mission sequence constrains the order the missions can be listed, based on the precedence constraints. For example, all activities in the first wave need to be listed before any activity in the second wave can be listed. Thus, from a precedence feasible list of missions and the mode they are executed in, a schedule is built.

The process by which schedules are built from the precedence feasible list is as follows: 1) Pick the first mission and mode. 2) Schedule the mission at the earliest

feasible time. 3) If there are any more missions left on the list, go back to step one. This process continues until the entire list has been traversed once. A simple analogy is that the combat planning structure provides the building blocks and pieces for the schedule, and the precedence feasible mission-mode sequence provides the directions for putting the pieces together. The better the instructions, the better the resulting schedule.

The schedule this process generates is one where no scheduled mission can start earlier without violating the constraints. It is possible that missions not in the critical path(s) can start later without extending the span of the schedule.

Without the Go constraint for flying squadrons (not incorporated in Van Hove's Math Programming model), the optimal solution for the problem is mapped to one of the possible precedence feasible list of mission/modes. The inclusion of the Go constraint creates more theoretical problems since there can be a period of time between the missions where an aircraft is not tasked to an activity, due to the Go constraints. Part of the problem is based on the manner in which the precedence feasible requirement can force missions to be scheduled in a specific order that can be contrary to the best solution with respect to the scheduling of Goes. This comes to play when a mission needs to be scheduled with a squadron, and despite having planes available in the current Go, the takeoff window ends before the earliest time the current mission can take off. The question then arises whether or not to shift the current Go to start later, thus scheduling the current mission in the current go. Alternatively, one may consider scheduling the mission in the next Go, thus causing the mission to possibly start at a significantly later time, while not taking advantage of the available aircraft for missions in the previous Go. This problem is worked around by assuming that waves should be completed as early as

possible, and any missions causing problems with the Goes would occur near the end of Wave. Thus shifting the Go to start later would extend the wave. Therefore, if a mission can not be scheduled in a Go as it is, then scheduling is attempted in the next Go.

3.3. Object-Oriented Data Structure Development

The research develops two main object-oriented data structures, the combat planning data structure and the combat scheduling data structure. The purpose of the combat planning data structure is for ease of interaction with the combat planner. This is accomplished by connecting the data structure directly to a GUI for entering in data. The combat scheduling data structure then uses fundamental project scheduling concepts to develop a structure that facilitates a heuristic approach in generating scheduling solutions in an acceptable amount of time.

Figure 15 shows the main features of the combat planning data structure, the combat scheduling data structure, and how the input data translates between the two. For example, a squadron at a base becomes a combat resource, and the aircraft at the base become assets.

Combat Planning data structure		Combat Scheduling data structure
• Base/Squadron	→	• Combat Resource
➢ Aircraft	→	➢ Assets
➢ Flying day, number of aircraft, and aircraft availability	→	➢ Asset Availability
➢ Goes	→	➢ Goes
• Target	→	• Activity (Mission)
➢ Weaponeering Options	→	➢ Modes
▪ Target-Base locations	→	▪ Mission leg durations
➢ Possible Scheduling Constraints		
▪ Window for mission milestones	→	▪ Start time Window
▪ Individual coordinations between pairs of missions	→	➢ Precedence Constraints
▪ Waves	→	➢ Waves

Figure 15 Converting the Planning Structure to the Scheduling Structure

While the two different data structures were developed simultaneously, for the purpose of explanation, the combat planning data structure is first explained along with the GUI interface. This is followed by an explanation of the combat scheduling data structure.

3.3.1. Combat Planning Data Structure and GUI

The combat planning data structure has two main functions. The first is to contain all the information needed to be able to schedule the missions. The second, and equally important, is to be user friendly, allowing a combat planner to enter in the data as quickly and easily as possible. This is equally important because no matter how good the rest of the program is, if the interface is not easy and quick to use, the combat planners will not find it worthwhile to use this tool.

3.3.1.1 Resource Types

Since there are often multiple squadrons to be selected, with the same type of aircraft, information specific for the aircraft is stored separately. In this way the combat planner is not required to enter the same information for each squadron. Figure 16 displays the prototype GUI interface for entering in this information.

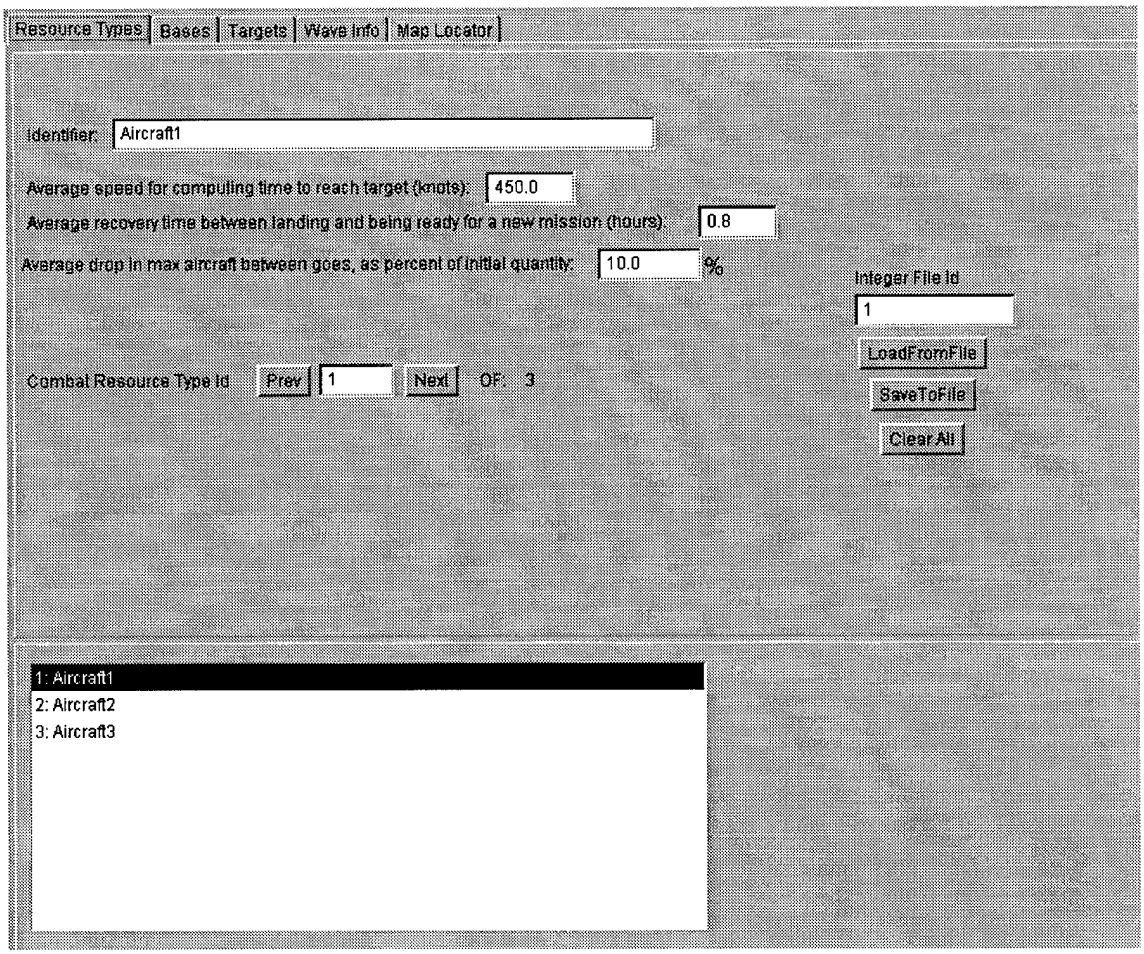


Figure 16 Window to input resource type information

The GUI for entering information is actually composed of five different panels, and the tabs at the top of the window switch to the appropriate panel. Loading and saving of the information for each panel is the same for all panels. The field above the load file and save file is used to input an integer identifier that is used to load and save the

information relevant to the panel. The list shown at the bottom of the window contains the current aircraft entered and can be used to switch between the different elements of the list. The resource Id is a unique identifier that indicates the position of the current aircraft in the list of aircraft types. The buttons on either side of the Id can be used to shift to an earlier or later data element.

The information is stored for each aircraft in a manner similar to database architecture. The *identifier* field is the name of the aircraft. The *average speed* is used to calculate the amount of time that it takes the aircraft to travel from the base to the target and back. The *recovery time* is the amount of time between when the aircraft lands and when it is projected to be again available for a mission. The *average drop in maximum aircraft between Goes, as percent of the initial quantity*, is used to provide an estimate of the maximum number of aircraft in each go. For example, if a squadron is composed of 20 aircraft, and the percent drop is 10%, 20*0.10 = drop of 2 aircraft (rounded to the nearest whole number). Thus, the first Go would be composed of 20-2=18 aircraft, and the second Go would be composed of 18-2=16 aircraft. The actual values used by the combat planners can be changed in the squadron window. The purpose of this value is to generate a rough estimate that can then be modified.

3.3.1.2 Map Locator

The purpose of this panel is to decrease the time needed to enter in the locations for different elements of the problem. The map locator panel is shown in Figure 17 (the map shown in the figure is from the following web address: http://www.lib.utexas.edu/Libs/PCL/Map_collection/Atlas_middle_east/Iraq.jpg). Different graphic files containing maps can be loaded by clicking on the *Import Map*

button located on the upper right side of the panel. This button opens a file dialogue window that allows any picture file of either .JPG or .GIF format to be loaded into the box on the upper left side of the panel. Once the map has been loaded onto the screen, the user sets two initialization points of known latitude and longitude on the map. The latitudes and longitudes for other locations on the map are determined based on these two points. The program assumes that lines of constant longitude go vertically across the map, and the lines of latitude go horizontally across the map. It is recommended that the initialization points be selected near the location relevant to the problem, with one point on the upper left side of the locations and the other on the lower right side. To do this, the user first clicks on the first radio button beneath the *Import Map* button, labeled *Init Point1*. Next, the user clicks on a location on the map which cause the location on the screen to be entered as *X* and *Y* coordinates in the fields listed below *Init Point 1* on the upper right side of the screen. Next, the user enters in the latitude and longitude for the point, in the fields following the *X* and *Y* coordinate fields. Then the user clicks on the radio button labeled *Init Point2* and follows the same procedure as before. Once this is done, whenever the mouse cursor is over the map, the latitude and longitude labels beneath the map will display the cursor's corresponding location on the map.

The Forward Line of Troops (FLOT) is used to determine at what point an aircraft crosses from friendly to enemy territory. The first time an aircraft crosses a FLOT line, it is assumed over enemy territory until it crosses the same point when it returns from the target. The FLOT is represented by a series of locations on the map where the consecutive points are connected by the straight lines. It is important to note that the first and last points are not connected. Therefore, if the FLOT encircles an area of the map,

the first location on the FLOT list should be the same as the last elements on the list. The FLOT points are stored along with the base information.

The points of the FLOT can be either entered manually, by typing in the coordinates in *Latitude* and *Longitude* fields located beneath the FLOT heading, or by the following method. First, the FLOT radio point is selected, now when the user clicks on locations on the map, a point is added to the list of FLOT points, with the latitude and longitude corresponding to the location on the map. Then the user just needs to click on the appropriate locations on the map, and the FLOT points are added to the list. The selection of base and target locations is done is a similar manner. There are also separate panels for entering in the other problem information needed on the different bases and targets.

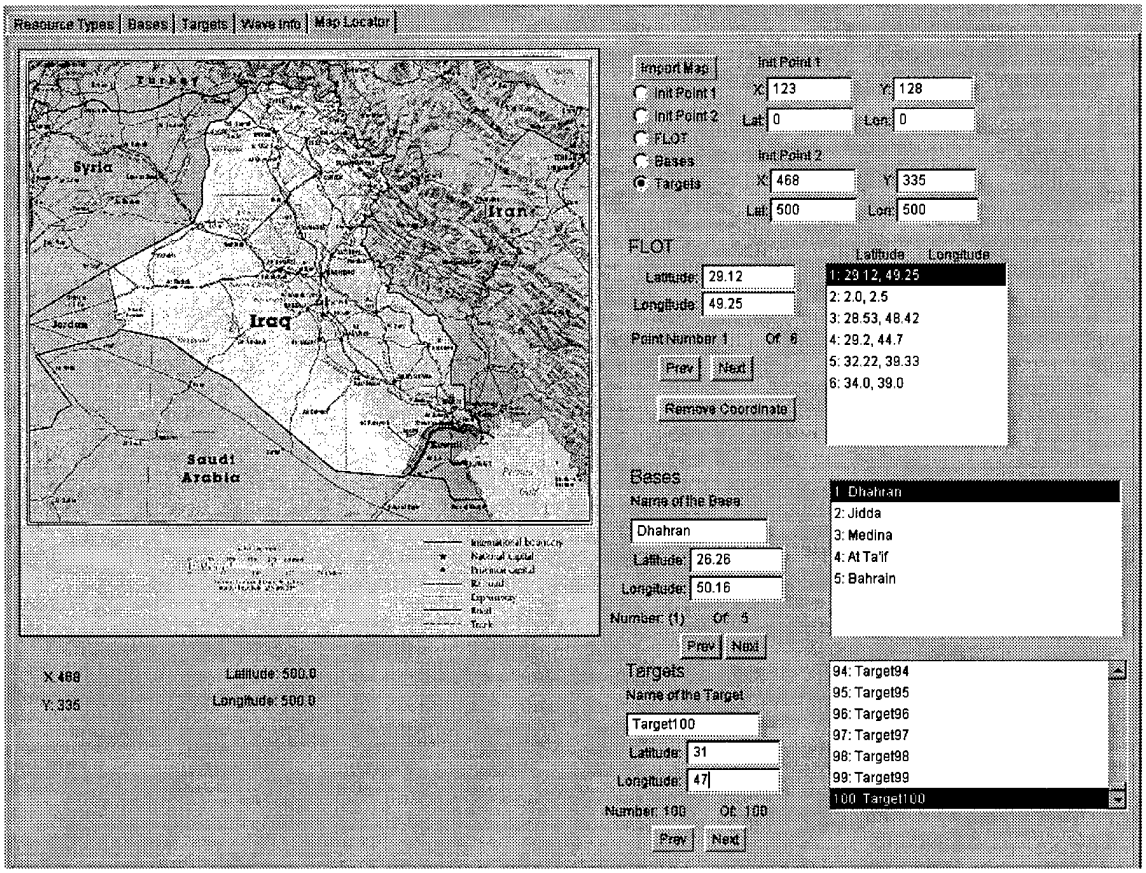


Figure 17 Window to Enter Locations via a Map

3.3.1.3 Bases & Squadrons

Currently the only information specific to a base is its name (*identifier*), its location, and the squadrons located at the base. Each base has its own unique list of squadrons, a squadron must be located at a base and can be located at no more than one base. Information pertaining to the base is on the left half of the window. The information pertaining to the squadrons located at the selected base is on the right side of the window. Figure 18 displays the prototype GUI interface for entering in base and squadron information.

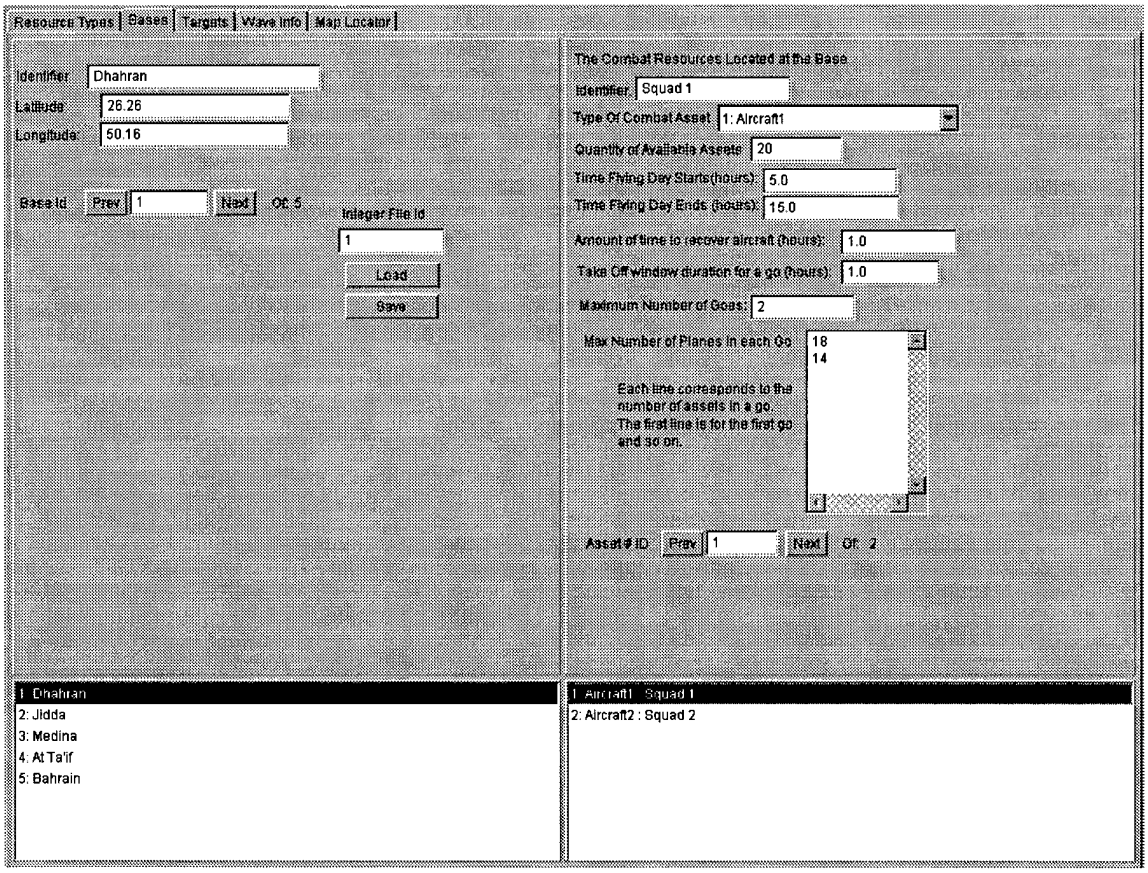


Figure 18 Window to input base and squadron information

As before, the *identifier* field for both the base and squadron are provided so the user can easily identify the base or squadron, instead of just using the integer Id value. The *Latitude* and *Longitude* fields are used to locate the base with respect to the other bases, the FLOT and targets. The different bases can be added either through this panel or through the Map Locator panel.

For the squadron, the *Type of Combat Asset* refers to the type of aircraft that make up the squadron. The user is limited to selecting one of the resource types entered into the first panel. Next, the user enters in how many aircraft are in the squadron, then the times that indicate the start and end of the squadron's flying day. Any missions assigned to the squadron are restricted to start on or after the start of the flying day. No mission

can be started that does not allow the aircraft to be fully recovered before the end of the flying day. While the term "day" is used, the value is not restricted to be between zero and twenty-four; rather, the number can be as large as is necessary to capture the problem. For example, since strikes can be flown at night, the flying day could start at hour 20, and continue to hour 30. The *Amount of Time to Recover the Aircraft* is the amount of time between when an aircraft lands and when it can takeoff for a new mission. The *Takeoff Window Duration for a Go* is the amount of time within which all missions for a Go need to be started. Next is the *Maximum Number of Goes* the squadron can fly, followed by the maximum number of aircraft allowed in each Go. The flying day and the length of the missions also limit the number of Goes actually flown by the squadron. Thus, a new Go can not start if it extends beyond the squadron's flying day.

3.3.1.4 Targets

The target panel, shown in Figure 19, has three different sections. The upper left section is used for entering and selecting different targets. The upper right section is where the different weaponeering options are entered for each target. The bottom section is for entering the specific mission pair constraints. The information in the weaponeering and constraint sections is target specific and is unique for each target.

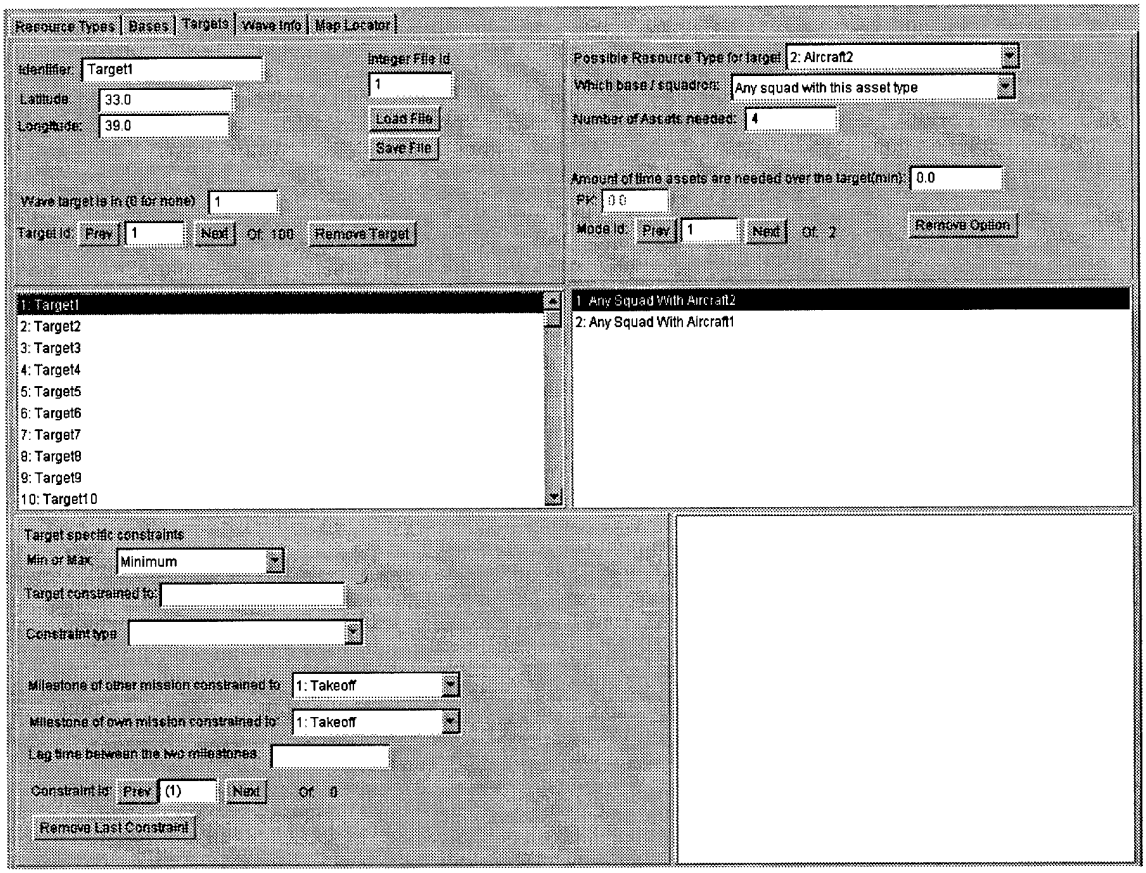


Figure 19 Window to input target information

The upper left section is used for selecting and entering new targets. The identifier field is used when printing out reports and for displaying information specific to the target. The *latitude* and *longitude* fields provide the location of the target. The information in the next two sections is unique to each target. Currently different priority classes are not incorporated for targets. However, the current implementation of the scheduling engine will in general tend to schedule missions for the targets that are listed earlier in the list more often then the missions listed later in the list. With this in mind, the combat planner should enter in the higher priority targets before the targets of lesser priority.

The upper right section of the panel is used for entering in the different weaponeering options for each target. The first field allows the planner to select one of the aircraft types. This selection is used to limit the choices in the base/squadron field to only the squadrons composed of that aircraft type, as well as an option to use any squadron with this aircraft type. The *number of assets* needed is the number of aircraft required from the squadron in order to fly the mission.

The bottom section of the window is used for entering in pair constraint information that allows the combat planner to set up coordinations between the strike missions. Currently only minimum constraints are allowed to be specified in the *Min or Max* field. The target is constrained to the other mission that this mission is constrained to. The other mission must be scheduled prior to allowing this mission to being scheduled. The *Constraint Type* field is not implemented (the idea behind this field is that it would give the user several options that would automatically set the next two fields according to the type of constraint). The next two fields allow the user to choose between which milestone the constraint attaches to any two related activities. For example, the mission for target #2 is constrained not to ingress until the mission #1 is finished attacking the target. The constraint would be under target #2 and the other target would be target #1. The *other milestone* would be milestone four, and the *own milestone* would be set to milestone two. The lag time would be set to zero.

3.3.1.5 Waves

The final panel is used for entering in which milestones are used to signify the start and end of each wave. The final panel is displayed in Figure 20. The waves do not overlap, so when all of the missions that belong in a wave have the end milestone occur,

then the missions in the next wave can have their start milestone occur. For example, if the first wave starts with milestone one, and end with milestone four, and the second wave starts with milestone two, and ends with milestone seven. This means that the first wave starts at the takeoff of the missions and ends after all of the missions have completed their time over target milestone. The second wave starts at the ingress of enemy territory, and ends when the aircraft are again available. Once the missions in the first wave have completed their time over target, the missions in the second wave can ingress enemy territory.

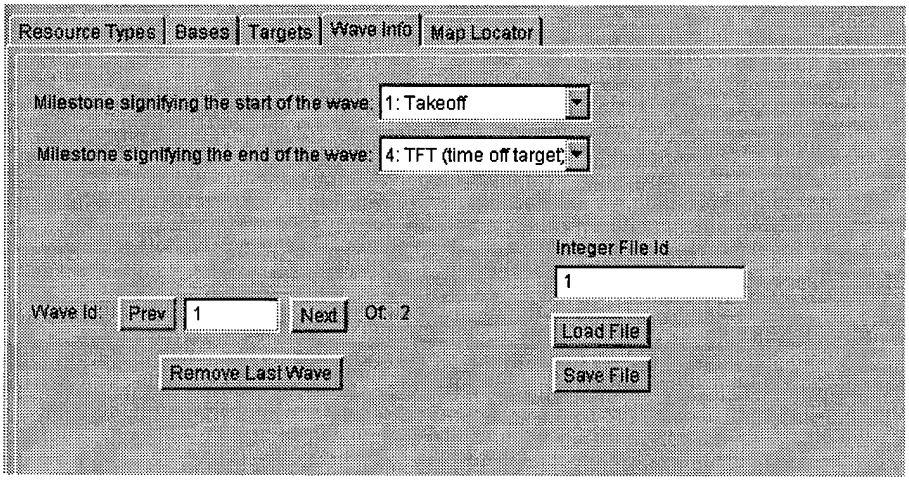


Figure 20 Window to input window information

3.3.2. Combat Scheduling Data Structure

The combat scheduling data structure has two main purposes. The first is to contain the problem information in a format conducive to building the schedule in a project scheduling style format. As mentioned previously in Section 3.2 above (on page 33), a precedence feasible list of missions and modes is used to build a schedule. Therefore, the second purpose of the combat scheduling data structure is to contain the

functions that allow a schedule to be built from a feasible list within the constraint of the problem.

Since a project management style framework is being used, project-scheduling terms are used instead of the combat planning nomenclature. The terms used are those presented in Table 1 (page 16). The next section discusses how the combat planning terms relate to a project scheduling style.

3.3.2.1 Nomenclature Associations

Project scheduling deals with the scheduling of activities, subject to both activity and resource constraints. In the case of combat planning, the missions to attack targets are activities and the different squadrons that carry out the missions are resources. Each squadron is composed of one or more aircraft and these aircraft are the assets of the resource. The weaponeering options determine the different squadrons that can be used to conduct a mission; these become the different modes of the activity. Each mode determines the resource needed to accomplish the activity, the number of assets required, and the window of time within which the activity needs to start.

3.3.2.2 Significant Features of the Scheduling Process

There are two significant features of combat scheduling data structure. The first is a flexible method of tracking asset availability across time, with time being continuous and not broken into discrete units. The second is the explicit modeling of waves, instead of implicitly capturing waves through precedence constraints.

3.3.2.2.1 Asset Availability Across Continuous Time

Tracking the use of resources across time, using mathematical programming techniques, is traditionally done by dividing time into discrete time intervals. A binary decision variable is associated with the availability of a single resource over a single interval of time. These decision variables assure that a single resource is not assigned more than one task in any unit of time. The problem with this approach is the large number of decision variables necessary to model anything beyond small problems. For example, to track the availability of 10 resources, during 10 hours, with a resolution of one minute, could require up to 10*10*60=6000 binary decision variables.

Instead of using an approach of discrete time intervals, a methodology utilizing some of the principles of Gantt charts (Figure 1 and Figure 2 on page 9) is developed. Rather than considering time as discrete intervals, the model in this study represents time as a series of concurrent and non-overlapping windows of time. Within each of these windows, the status of the resource is constant. To represent this information in a computer, a status window object is created. Each object stores the start time of the window, the status of the resource, and other relevant information. The end of a window is simply the start time of the next window. A dynamic list of the status window objects then stores the availability of a resource, with the number of objects in the list growing as the resource is tasked to different activities.

Often, the availability of the resource is modeled across a specific interval of time. If this is the case, the first element and the last element of the dynamic list represents the start and end of the interval of time. However, due to the flexibility of this approach, a more open-ended model is possible. Instead of representing the earliest and latest times a resource is available, the first element could represent the earliest time the

resource is currently tasked, and the end element could represent the latest time the resource is currently tasked. If this approach is taken, a specific interval of resource availability does not need to be determined before hand. The only limiting factor of this approach is the precision of the variables used to store the time values. For the purposes of this research, the approach of setting a specific interval of resource availability is used. This is because for this research the availability of the resources is defined by the flying day of the squadron.

For this research, each resource is composed of one or more assets. Since the availability of each of these assets needs to be tracked, a dynamic list of windows is used for each asset. The method of tasking the activity involves traversing the list of windows until a large enough free window is found to accommodate the activity, or the end of the list is reached. An activity Id, the earliest and latest start times, as well as the duration of the activity, are the inputs used to determine when and if an asset can be tasked.

Searching the dynamic list of windows for a resource uses the following procedure. First, the dynamic list is traversed until the first window that ends after the earliest start time is reached. Starting with this window, the list is sequentially searched through the windows that start before the latest start time, to find the windows of time that the asset is free. Each free window is checked to see if it is large enough to schedule the activity for the given duration, and within the earliest and latest start times. If a window is found, either the earliest time the asset can task the activity is returned, or the asset can be immediately tasked to the activity. If the size of the free window coincides with the time that the resource is to be tasked, then the status of the window is changed accordingly. However, if they do not match up, then one or two status window objects

are inserted into the list, and values are adjusted accordingly. The process used to task the asset within a window is as follows.

Figure 21 is used to illustrate how the availability of the resource is updated to accommodate the tasking of an activity. For this example, it has already been determined that the resource can be scheduled within a window of time that the resource is available, represented by timeline **A**. This window is represented by the elements W_0 and W_1. The status of W_0 signifies that it is the start of an available window. The status of W_1 does not matter; this element is only significant for the current situation in that it indicates the end of an available window of time for the resource. The window starts at time t_0 and ends at time t_1. This window could be the part of a larger list of window elements. The activity to be tasked starts at time t_a and ends at time t_b.

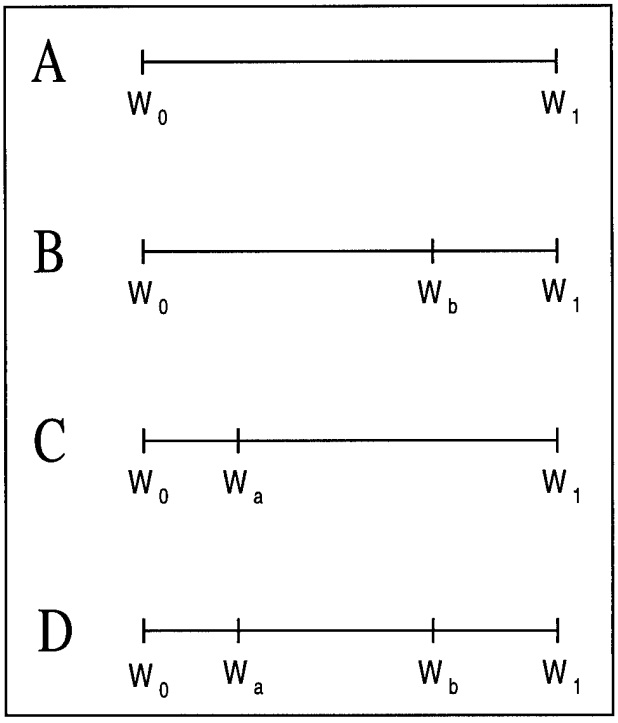


Figure 21 Possibilities for Tasking a Resource

Following are the four possible outcomes:

1. $t_a = t_0$ and $t_b = t_1$ - Timeline A, the activity starts at the start of the available window and ends when the available window ends. For this case, the status of element W_0 is changed to indicate the start of a window of time where the resource is not available, and it has been tasked to the appropriate activity.
2. $t_a = t_0$ and $t_b < t_1$ - Timeline B, the activity starts at the start of the available window and ends before the available window ends. For this case, the status of element W_0 is changed to indicate that the activity has been tasked to start at this time. However, now there is still some available time after the activity is finished with the resource. Therefore, element W_b is inserted into the list, between elements W_0 and W_1. Element W_b occurs at time t_b, and it has a status of available.
3. $t_a > t_0$ and $t_b = t_1$ - Timeline C, the activity starts after the start of the available window and ends when the available window ends. For this case, a window of available time remains before the activity starts. Therefore, element W_a is inserted between elements W_0 and W_1. Element W_a occurs at time t_a, and it has a status of being unavailable.
4. $t_a > t_0$ and $t_b < t_1$ - Timeline D, the activity starts after the start of the available window and ends before the available window ends. For this case, a window of available time remains before the activity starts, and after it ends. Therefore, element W_a and W_b are inserted between elements W_0 and W_1. Element W_a occurs at time t_a, and has a status of being tasked

to the activity. Element W_b occurs at time t_b and has a status of being available.

One of the constraints of the research problem is that multiple assets can be required from a resource, over identical windows of time. A simplified method for solving this is as follows (a more complicated and faster method is used in the program and is explained later in the chapter; however, this approach gives the general idea). 1) All of the assets are searched to find the earliest time an asset can be tasked, within the start window and for the given duration, and the quantity of assets that can accomplish the activity at this time. If an asset can not schedule an activity at the best time, the second best time the asset can schedule the activity is also found. 2) If enough assets can be scheduled at the earliest time, then schedule the appropriate number of assets and exit the process as a success. 3) If not enough assets can be scheduled at the earliest start time and there is a second earliest start time, then change the start window being searched to start at the second earliest start time, and go to step 1. 4) If no assets can be scheduled at any time, or not enough assets can be scheduled at the earliest start time and there is no second earliest time, then exit the routine as a failure.

3.3.2.2.2 Waves

Pair precedence constraints can be used to implicitly model the waves defined by the wave constraints defined in the project-planning model. Instead, this research explicitly models the waves in that the latest time the end milestone occurs is tracked and this value is then used as the earliest time the start milestone can occur in the subsequent wave.

Figure 22 illustrates the concept of modeling waves. The timelines represent activities contained within the waves. The first three timelines represent missions in the first wave; the "X" on each of the three timelines indicates the time at which the end milestone occurs during each mission. The latest time any of the end milestones occur in the first wave is represented by t_0. Once all of the activities in the first wave are scheduled, then the activities in the next wave can be scheduled, as represented by the second set of three timelines. The stored value of time t_0 is used to determine the earliest time the start milestone can occur for missions in the second wave. The "^" character represents the time the start milestone occurs in each of these missions.

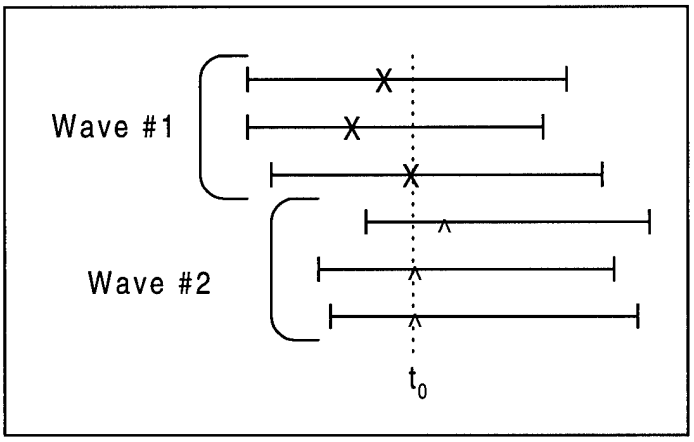


Figure 22 Wave Example

3.3.2.3 Converting Locations into Durations

One of the differences between the planning and the scheduling data structures is that the planning structure dealt with geographical locations, while the scheduling structure deals with the length of time to complete the mission. The equation shown in Figure 23 is used to calculate the angular difference (D) between site *a* and *b*, where their locations are expressed as a latitude and a longitude value (Bailey).

$$D = \cos^{-1}\left[\sin(lat_a)*\sin(lat_b)+\cos(lat_a)*\cos(lat_b)*\cos(|lon_a - lon_b|)\right]$$

Figure 23 Equation for the Angular Difference between Locations

If the result D is in terms of radians, then it is converted to degrees by multiplying it by 57.2958 degrees per radian. Each degree of angular difference is than multiplied by 60 nautical miles per degree. The time to cross this distance is found by dividing the distance by the speed in knots of the aircraft.

To determine when during a mission the aircraft will ingress and egress enemy territory, the following process is used. First, the minimum distance is set to the distance between the base and the target, and the location is set to the target. Next, the function loops through the lines that connect the consecutive points. For each line, the function calculates where it intersects the line connecting the base and the squadron. If the point where the FLOT line is intersected is between the two defining points, then the function calculates the distance between the base and the intersection. If this value is less than the minimum distance, the new distance and location is stored as the intersection point. Once all of the lines are calculated, then the distance between the location of the minimum and the target is calculated. The distance between the base and the intersection is used to approximate the time to fly from the base to the ingress point and from the egress point to landing. The distance between the intersection and the target is used to determine the time to fly from the point the aircraft enters enemy territory to the time it is over the target, and from the time it is off the target to when it exits enemy territory.

3.3.2.4 *Hierarchy*

The chart in Figure 24 shows the hierarchy of the objects used in the combat scheduling data structure. For example, the *All Resource Constraints* object contains a list of one or more *Resource Constraint* objects. Each of the *Resource Constraint* objects contains multiple dynamic lists of status windows used to track asset availability, as well as a list of *Go Constraints*.

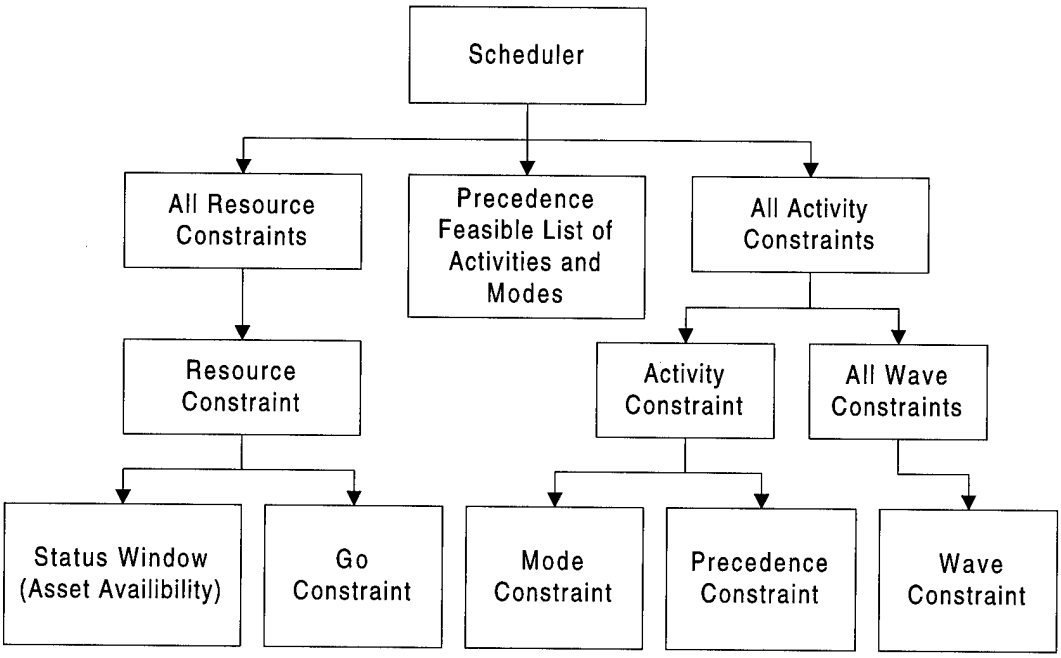


Figure 24 Project Scheduling Hierarchy

Each of the objects in Figure 24 is explained in the following sections. First, the type of object and data contained within each object is explained, followed by the main functions of each object.

3.3.2.5 *The Scheduler Object*

There are two main sections of the combat scheduling data structure: the constraints that are specific to the missions, and the constraints that are specific to the

combat resources. These two objects are both members of a scheduler object, shown in Figure 25. The scheduler object also contains a precedence feasible list of activities and modes. As mentioned before, these are the instructions used by the scheduler to build the schedule.

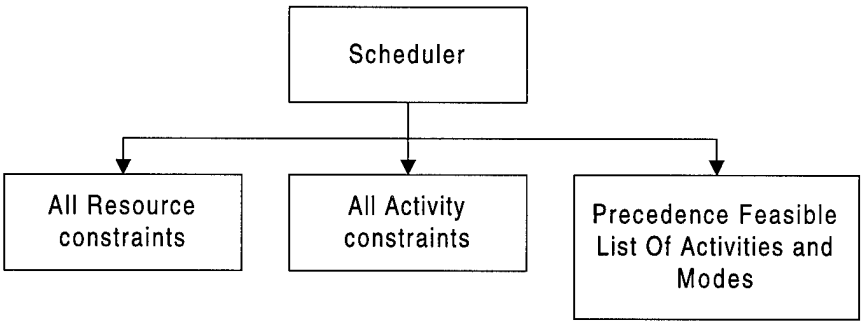


Figure 25 Scheduler Data Structure

3.3.2.6 The Activity List Object

The activity list object contains all of the constraints that are specific to the activities. The two main sources of data in this object are the list of individual activity constraints and all of the wave constraints, as shown in Figure 26.

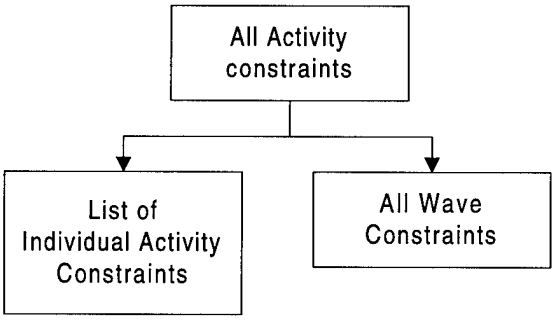


Figure 26 All Activity Constraints

3.3.2.6.1 The Activity Constraint Object

The activity object is shown in Figure 27, and its primary purpose is to contain the information specific to an activity. The functions in the activity list access and update the information as needed.

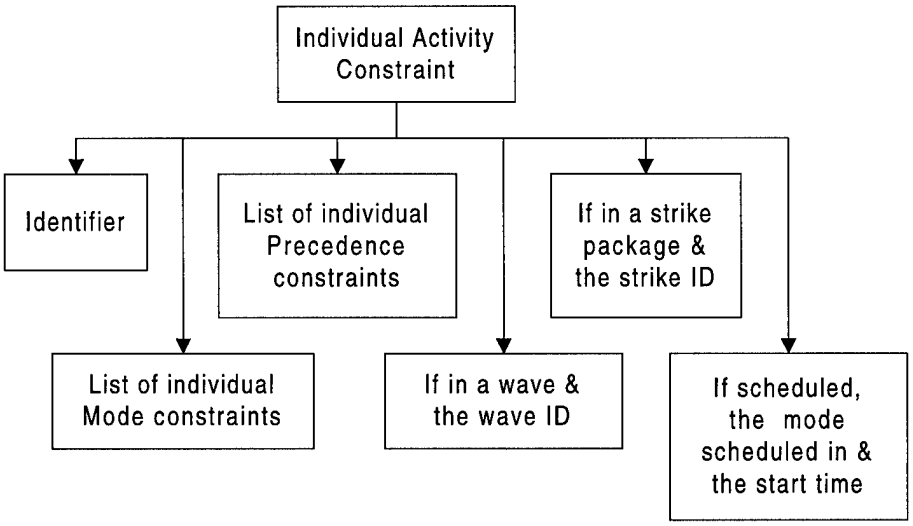


Figure 27 Individual Activity Constraint

The identifier is the string identifier that represents the mission or target the activity represents. The list of modes contains the information specific to the different modes the activity can be executed in. The list of precedence constraints are the precedence constraints specific to the activity. The wave ID represents which wave the activity belongs to, if any. The strike ID represents which strike package the activity is a part of, if any. The activity can belong to at most one wave and one strike package. If the activity is scheduled, the object stores that the activity is scheduled, the mode it is executed in, and the time it currently starts.

3.3.2.6.1.1 The Mode Constraint Object

The individual mode constraint object contains the information that is specific to a particular mode of execution for the activity, shown in Figure 28. The Id of the resource needed represents which resource the activity will use if executed in this mode. The number of assets needed is the number of assets required from the resource for the activity. The individual leg times of the activity provide the times that different milestones occur during the activity, in relation to the start time. The total duration of the activity determines how long the assets from the resource will be in use. The set earliest and latest start times are the earliest and latest times the activity can start, if executed in this mode. The purpose of storing these times is for those occasions when a particular milestone needs to occur within a window of time. For example, in a coordinated land, sea and air attack, a target is required to be attacked at a particular time. Depending on which resource is used to attack the target, the time the activity will need to start may vary.

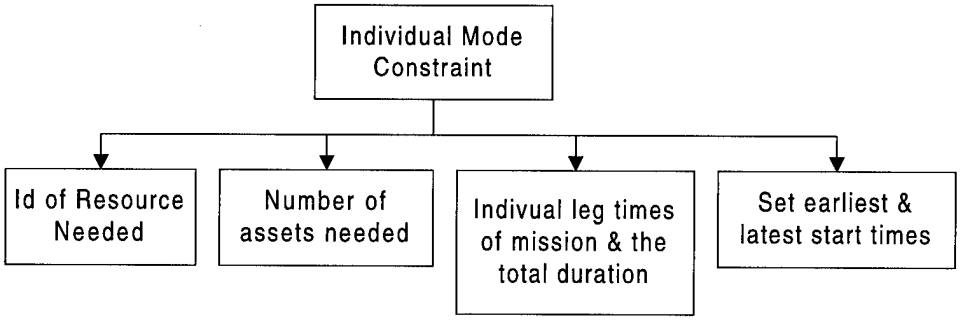


Figure 28 Individual Mode Constraint

3.3.2.6.1.2 The Precedence Constraint Object

The precedence constraint object contains all the information for a precedence constraint that links the current activity to one already scheduled. The precedence

constraint object, shown in Figure 29, is used to contain the precedence constraint information for an activity, which is then accessed by the functions in the Activity List object. The max or min time relationship refers to whether the constraint is a minimum or maximum constraint. The amount of time is the amount of lag for a minimization constraint; for a maximum precedence constraint, the value is the amount of time within which the milestone needs to occur. The own, and other activity's milestone the constraint is attached to, refers to the milestone of the current activity and the other activity that are used in the constraint. The Id of the other activity refers to the other activity that the current activity is constrained to.

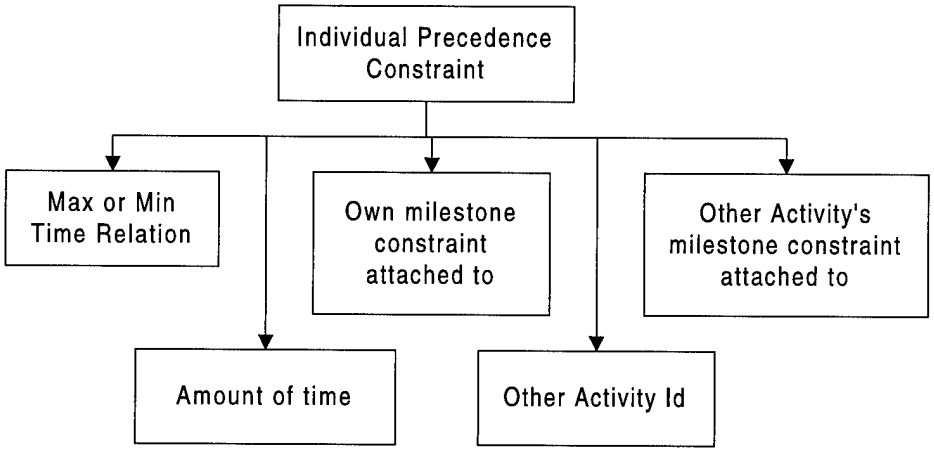


Figure 29 Individual Precedence Constraint

3.3.2.6.2 The Wave List Object

The wave list object contains all of the constraints that are specific to the different waves, as shown in Figure 30. It also stores which wave is currently being scheduled. During the scheduling process, all of the activities within a wave are scheduled before activities are scheduled in the next wave.

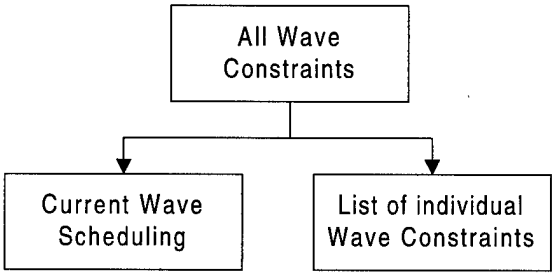


Figure 30 All Wave Constraints

3.3.2.6.2.1 The Wave Object

The wave object contains all of the information specific to an individual wave, as shown in Figure 31. The "start" and "end" milestones refer to the milestones that are used to signify the start and end of the wave. The start of the wave occurs when the earliest time an activity within the wave has its "start" milestone. Likewise, the wave ends at the latest time one of its activity's "end" milestone occurs. The number of activities in the wave is simply the number of activities that are in this wave. Only when all of these activities are scheduled, can activities in the next wave be scheduled. The current time the wave starts and ends refers to the current earliest time the "start" milestone occurs and the latest time the "end" milestone occurs for the activities scheduled. The list of activities scheduled in the wave is a list that references the activity object of each activity in the wave that is currently scheduled. This object is used if the current start and end times of the wave need to be recalculated.

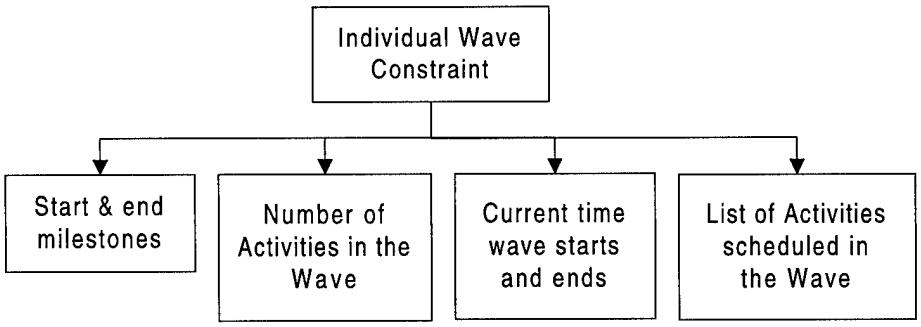


Figure 31 Individual Wave Constraint

3.3.2.7 The Resource List Object

The all resource constraints object contains all of the information for the different resource constraints. As Figure 32 indicates, this object is composed of a list of the individual resource objects. The primary justification for this object is in the functions that need to call all of the resource constraints, and it follows the framework of the activity list object.

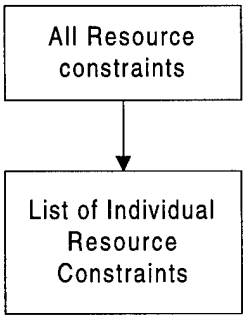


Figure 32 All Resource Constraints

3.3.2.7.1 The Resource Constraint Object

The resource constraint object contains the information specific to a single resource and is shown in Figure 33. The identifier is built from the squadron name and the base the squadron is from. The initial flying day stores the earliest time a mission can

be started, and the time the latest mission needs to be completed. The asset availability with respect to time keeps track of when individual assets are available and is explained in the following section. The list of the Go constraint objects is used to track the information specific to the current Go. The current Go scheduling indicates the Go that the last activity was scheduled in by this resource. The list of possible activities is a list of the activities that have modes that use this resource.

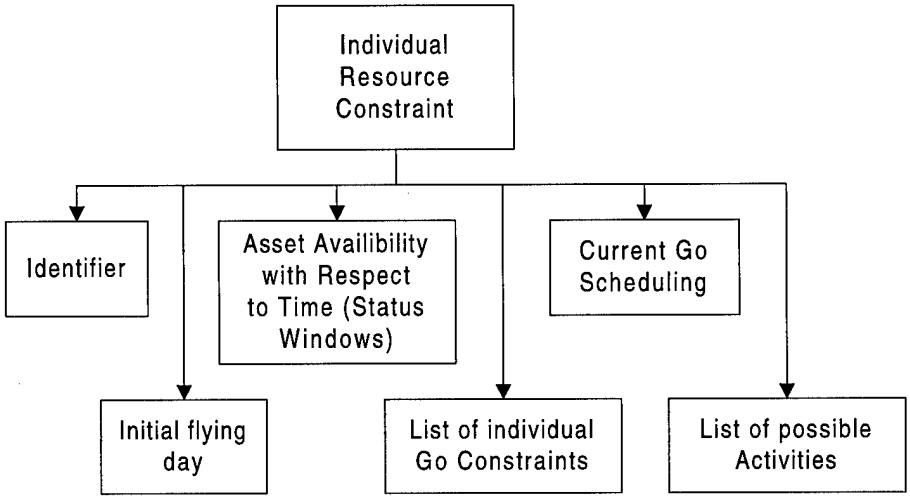


Figure 33 Individual Resource Constraint

3.3.2.7.1.1 Asset Availability with Respect to Time

Asset availability is kept track of through the approach outlined in section 3.3.2.2.1, "Asset Availability Across Continuous Time." The status window object, shown in Figure 34, contains the information specific to the status of an asset across a window of time. The start time is the time that this window starts. The time this window ends is when the next window starts. The status of the asset can be: available, scheduled to an activity, or the end of the list and unavailable. The activity Id is used when the status of the asset is scheduled to an activity. This variable contains the Id of the activity the asset is scheduled to during the window.

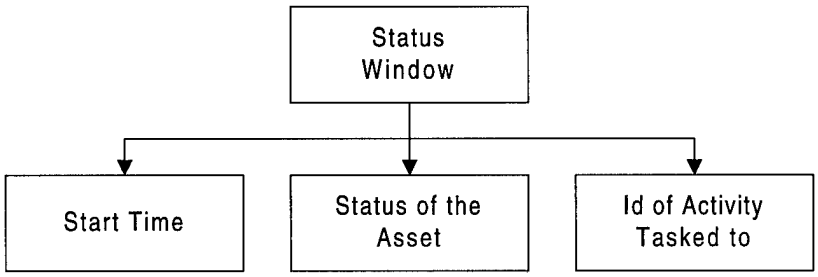


Figure 34 Status Window

Each asset of the resource has an associated dynamic list of status windows to track the asset's availability through the flying day. Initially, each list is composed of two elements, the first of which starts at the beginning of the flying day and has a status of available. The last starts at the end of the flying day and indicates that the asset can not be tasked at any time later than this.

3.3.2.7.1.2 The Go Constraint Object

This object keeps track of all the information specific to a single Go of the resource, as shown in Figure 35. The maximum number of assets is the maximum number of assets that can be tasked during this Go. The takeoff window duration is the maximum amount of time between the first and last start times of activities in the Go. The earliest allowed takeoff is the earliest time an activity within the Go is allowed to start. The current earliest and latest start times refer to the activities within the Go that have already been scheduled. The current latest asset again available is the latest time one of the activities within the go finishes. This value is used to calculate when the next Go can start. Since all assets used in a Go need to be available for use in the next Go, the start time of the next go is the time the latest asset is again available from the previous Go, minus the takeoff window duration. The number of assets currently scheduled tracks

the number of assets that have currently been scheduled in the Go. The assets that can be scheduled in the Go is a list of the assets that can be tasked to missions scheduled in the Go. The list of activities scheduled in the Go is a list of references to the activity objects corresponding to the activities that are currently scheduled in the Go.

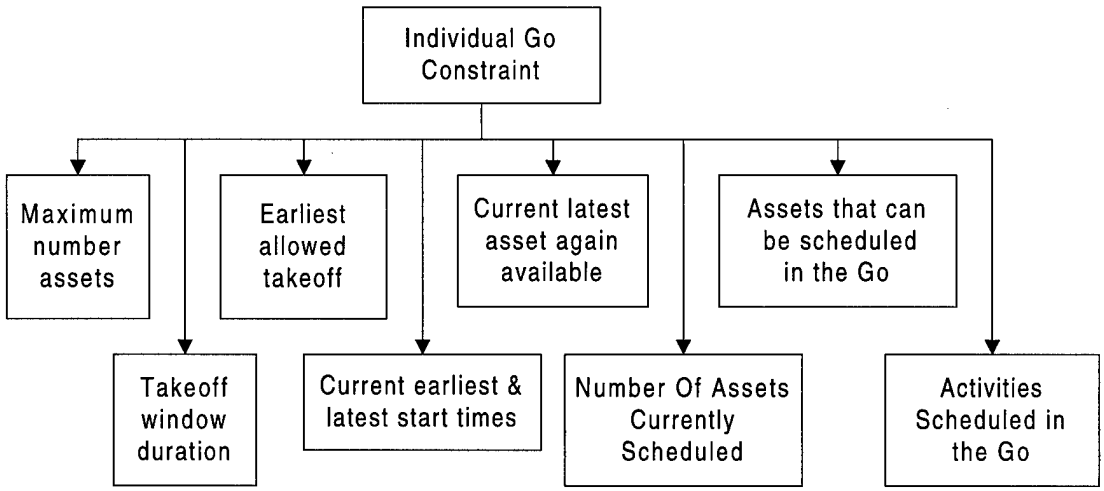


Figure 35 Individual Go Constraints

3.3.2.8 Roadmap Check

It is important to remember that this is not one of the classical mathematical programming techniques that use constants, variables, and equations to define and solve scheduling problems. Instead, the objects mentioned previously are used to represent the scheduling problem. The following sections contain the functions that are used to find solutions to the scheduling problem. These functions are also contained within the objects.

3.3.2.9 The Scheduler Object Functions

The primary function of the scheduler is to schedule the activities at the earliest possible times, subject to the constraints of the problem, and in the order provided by the

precedence feasible list. Figure 36 contains the main steps of the function that accomplishes this. First, the function picks the next activity/mode from the precedence feasible list. Next, the scheduler calls a function in the activity constraints to determine the window of time within which the activity needs to start, to satisfy all activity constraints. This function also returns the number of assets needed, and the duration of the activity. The next function called is from the resource constraints. This function accepts the window of time the activity can start, the total duration of the activity, and the number of assets needed from the resource. This function attempts to schedule the activity at the earliest time with the appropriate resource, subject to the constraints provided from the activity constraints and meeting the constraints of the resources. If the activity is scheduled, then the scheduler calls a function from the activity constraints that updates the activity as scheduled, in the particular mode, and at the start time. If there are any more activities on the list, then the function goes back to the first step of picking off an activity and mode. The final step is scoring the schedule.

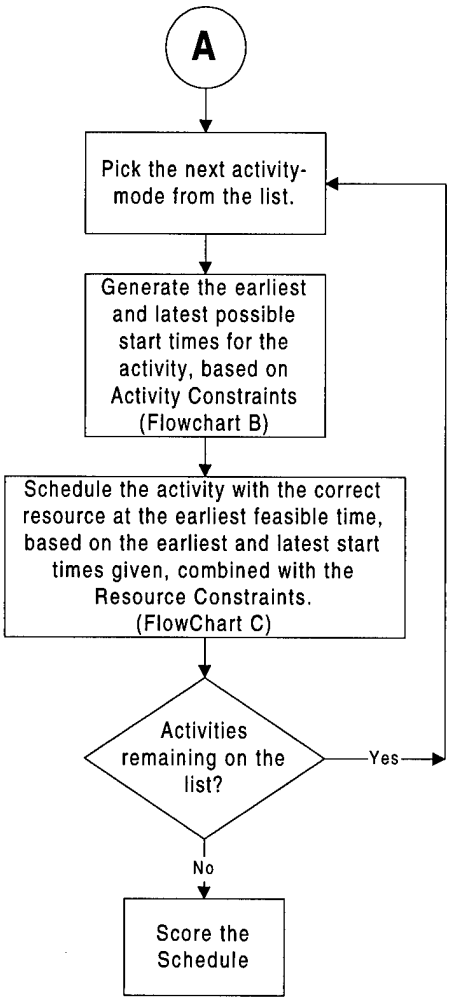


Figure 36 Build Schedule from a Precedence Feasible Mission/Mode Sequence

The scoring of the schedule calls two functions in the all activity constraints object. They return the percentage of activities scheduled, and the length of the schedule generated. In this case, the goal is to maximize the percentage of activities scheduled. If more than one schedule has the same percentage of activities scheduled, then the scheduled that is completed in the shortest amount of time is preferred. (Of course, expert opinion in the form of the combat planners may select between schedules based on intangible factors.) Some of the other functions in the scheduler include initialization and re-initialization routines. The initialization routine calls the activity and resource

constraint objects to create the necessary data structures and to initialize the values used in the process of building a schedule. The re-initialization routine resets all of the scheduling process data structures in the activity and resource constraints. This allows the repeat of the scheduling process, with a new precedence feasible list, and the same activity and resource constraints.

3.3.2.10 The Activity List Object Functions

There are four primary functions in this object. Some of these functions use information that is contained in the activity and wave constraints.

The first of the primary functions determines the window of time within which an activity, in a particular mode, can start, based on the activity constraints. The function takes into account the following constraints: wave constraints, set earliest and latest starts defined by the mode, and precedence constraints with other activities already scheduled. Figure 37 outlines the main steps of this function.

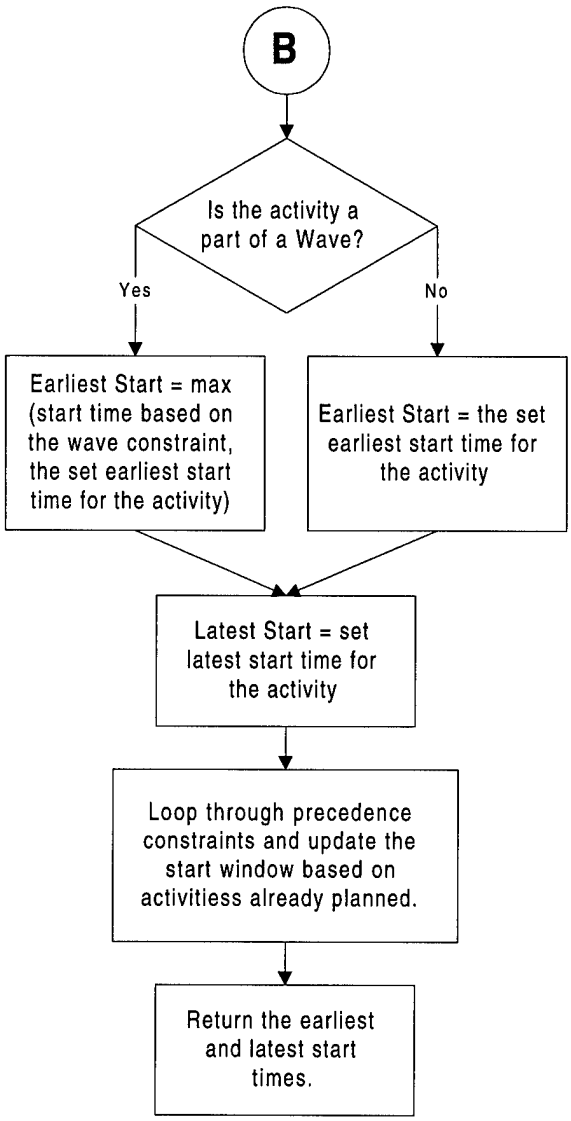


Figure 37 Determine the earliest and latest starts from the target's constraints

The earliest start time based on the wave constraint is the earliest time the activity can start so that the milestone relating to the starting milestone of the window it is in, occurs on or after the ending of the previous wave. Recall a window has a starting and an ending milestone. The time the wave starts is the earliest time the start milestone occurs in any of the activities in the wave. The wave ends after the latest time the end milestone occurs in any of the activities in the wave. In addition, consecutive waves are non-

overlapping, which means that all of the end milestones need to occur in a wave before any of the start milestones can occur in the next wave.

The following example uses the precedence example shown in Figure 38 to demonstrate the idea. This is an example where once a predecessor activity attacks an air-ground target, the successor mission can ingress enemy territory. For the wave example, the predecessor activity is the activity in the previous wave that completes the end milestone at the latest time of any activity in the wave. In this case, the end milestone is milestone four, which occurs after leg three of the activity. The successor activity represents the new activity to be scheduled, which is in the next wave. The start milestone of this wave is milestone two, which occurs after the first leg of the mission. The end of the first wave is therefore time *t0* + the durations of legs one through three of the predecessor activity. This also represents the time the next wave can start, which means the successor activity can start on or after time *t1*, which is the end of the previous wave minus the duration of leg one of the successor activity.

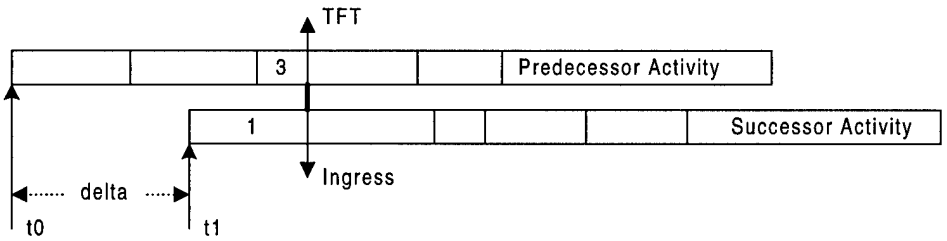


Figure 38 Precedence Example

The step of looping through the precedence constraints in Figure 37 works along the same lines as handling the wave constraint. At this point a possible start window has already been found for the activity, i.e. the activity can start on or after the earliest start time, and can start no later than the latest start time. The predecessor activity, in Figure

38, represents an activity already scheduled, and to which the current activity, the successor, is attached by a precedence constraint. For this example, milestone two of the successor activity needs to occur on or after the time milestone four of the predecessor activity occurs. The function adds the time the predecessor activity starts to the durations of the legs that occur before milestone four in the predecessor activity. To this value, the time value of the constraint is added. For minimum constraints, this value is usually zero, unless there is a buffer (this is explained further in section 3.3.2.6.1.2 on page 58). From this value, the durations of the legs that occur before milestone one of the successor activity are subtracted. The resulting value is the time after which the current activity needs to start, based on this precedence constraint. This value is compared to the current start window for the activity, and if it is greater than the earliest possible start time, it now becomes the earliest start time. If this was a maximum precedence example, then the time value represents the latest possible start time and the minimum latest start time would be kept for the activity's start window. In this manner, the precedence constraints shrink the start window of the activity.

The second primary function is to update the appropriate information, if the activity is scheduled; this is done by updating the particular activity to being scheduled, at the start time, and in the mode it is executed in. If applicable, it also involves including the activity in the list of activities that are scheduled in the current wave.

The third function involves determining how much later an activity, that is already scheduled, can be shifted to start later, without violating any activity constraints. This function does not consider any resource constraints that can affect how much later the activity can start. The value calculated also assumes no other activities are to be

shifted. Thus, if the activity in the middle of a group of activities chained together by precedence constraints, the activities that occur before and after the current activity bracket how much it can be shifted. In the terms of project scheduling, this is referred to as the independent slack of the activity. The independent slack is the amount of time an activity can be shifted to start later, without affecting the earliest start times of any activities following the activity.

The amount the activity can be shifted is found in the following manner. First the slack based on the set start time is calculated by subtracting the time the activity currently starts from the latest possible start time. If applicable, the slack based on the wave is found by subtracting the time the end milestone occurs in the activity from the latest time the end milestone occurs in the wave. Slacks based on the activities' maximum precedence constraints are then calculated, followed by any other scheduled activities' minimum precedence constraints that attach to the current activity. The minimum of all of the previously calculated slack values is the one that is returned. This function is used by another function in the resource section that determines if a Go can be shifted to start later.

The fourth function is a scoring function. This function is actually two different functions that return values that evaluate the goodness of the constructed schedule. One function returns the percent of activities scheduled. The other function returns the time that the last activity is completed.

Other functions included in the Activity List object include initialization and re-initialization routines. Both of these function loop through the list of activities and call either the initialization or re-initialization routine for each activity.

3.3.2.10.1 The Activity Constraint Object Functions

The functions for the Activity Constraint Object are only the initialization and re-initialization functions. These functions simply set the activity to not being scheduled.

3.3.2.10.2 Wave List Object Functions

The primary purpose of this object is to determine the earliest time the current wave can start. The function returns the time the previous wave ends. If the current wave is the first wave, then a value of zero is returned. The re-initialization function sets the current wave being scheduled to zero, and calls all wave object's re-initialization function.

3.3.2.10.2.1Individual Wave Constraint

This object contains a function that determines the current time the wave ends, by looping through all of the activities currently scheduled in the wave and finding the one whose end milestone occurs the latest. The re-initialization routine does the following: 1) Sets the current time the wave starts to a "large" value. 2) The current time the wave ends and the number of activities scheduled are set to zero. 3) The list of all the activities currently scheduled in the wave is emptied.

3.3.2.11 The Resource List Object's Functions

The primary purpose of this object is to provide an interface between the scheduler and the constraints of the different resources. It also has initialization and report functions which loop through all of the resource objects and call the appropriate functions.

3.3.2.11.1 The Resource Constraint Object's Functions

The functionality of the resource object is that it needs to be able to accept an activity to be scheduled, along with the activity constraints, and determine if the resource can accommodate the activity and at what earliest time. If the activity is to be scheduled at this time, there needs to be a function that can task the resource at this time. An overview of the process through which this is done is shown in Figure 39. The input to this process is the activity constraints, which are found through the functions in the activity list object. They include: the number of assets needed, the earliest and latest start times, and the duration of the activity.

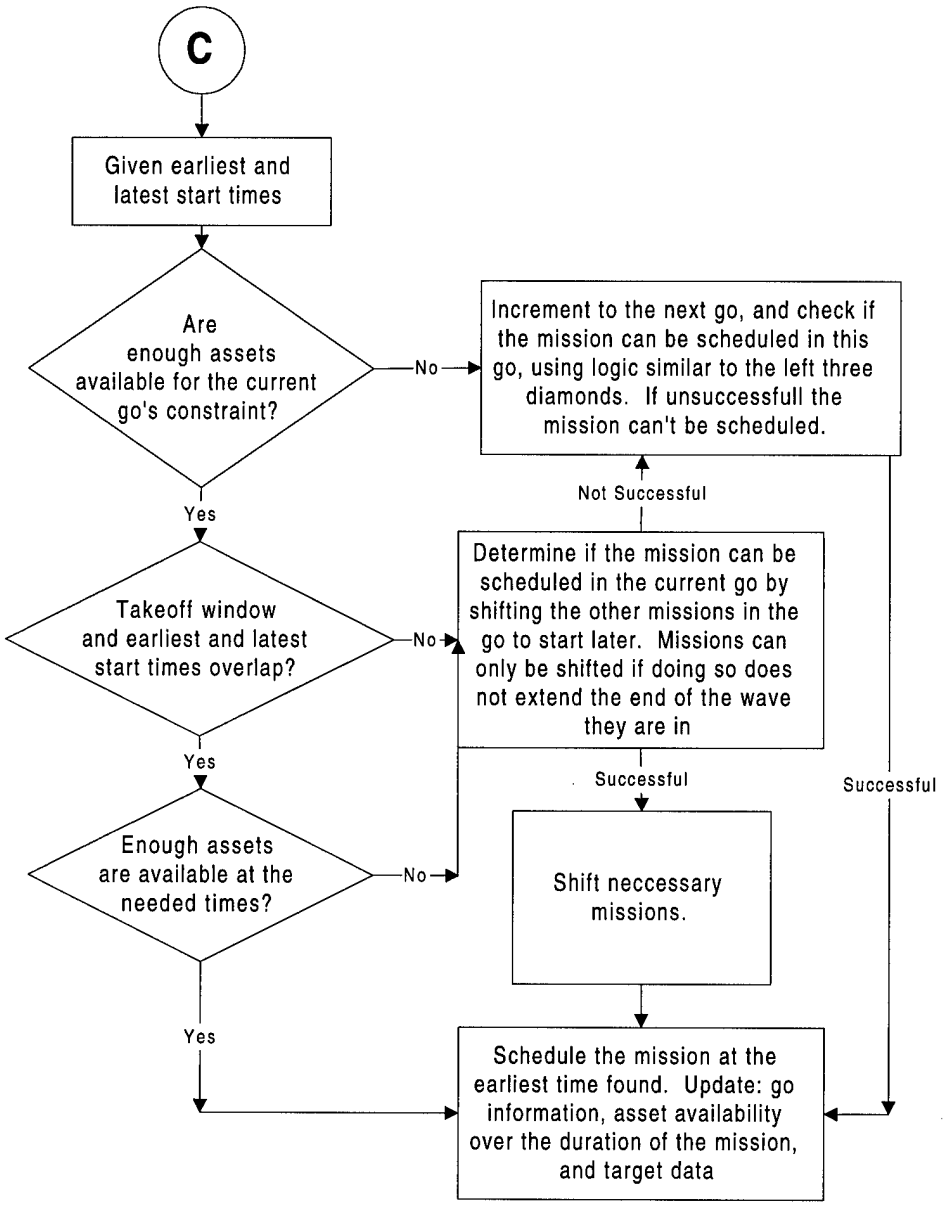


Figure 39 Determine if resource can schedule the activity

The following are the main steps for determining how early a resource can be tasked to an activity. The steps of the process are as follows: First, the function checks to see if there are enough assets available in the current Go. If so, then the function checks to see if the activity window provided by the activity constraints overlaps the takeoff window for the Go. If they do, then the overlap amount of time is then used for the start

window for the next step. This step involves checking if enough individual assets are available, at the same time, within the start window, and for the duration of the activity. If enough assets can schedule the activity, then this time is returned as the earliest start time the resource can accomplish the activity. These steps covered the left three diamonds on Figure 39.

If the bottom two diamonds, the last two checks, were in the negative, then the function checks to see if the current Go can be shifted to start later. This involves checking how much the other activities that are scheduled in the Go, can be shifted to start later without extending the end of the wave they are in or violating any precedence constraints. If the mission can be scheduled within the time that the Go can be shifted to, and the mission will start at a time earlier than if it had been scheduled in the next Go, then return the time that the resource can be tasked to the activity.

If not enough assets were available in the current Go, the first diamond, or the current Go could not be shifted to a later enough time to accommodate tasking the activity, then increment to the next Go. This ends up at the box that is on the upper right corner of Figure 39. Then the function attempts to task the resource using logic similar to that in the above paragraphs, except this time, if the any of the three diamonds are not successful, then the resource can not be tasked to the activity. If the activity can be tasked to the resource, then the start time is returned.

The bottom two boxes of Figure 39 involve the scheduling of the mission at the earliest time found. This is separated from the first function so that the above function can be called multiple times for different resources to determine the earliest time that the activity could be started or finished. The updating of the variables of the resource object,

for tasking the activity, involves several steps. The first of which involves checking to see if the scheduling of the activity involved shifting the current Go to start later. If it does, then the appropriate activities are shifted to the earliest time they can be scheduled, and still schedule the current activity. If the activity is scheduled in the next Go, then the resource is incremented to the next Go. The tasking of the individual assets involves looping through the individual assets until enough assets can be tasked to the activity at the appropriate time, for the given duration.

3.3.2.11.1.1Asset Availability with Respect to Time

The availability of individual assets is tracked through a dynamic list of the status window objects. The term dynamic portion of a dynamic list simply means that the number of objects within the list can increase or decrease as needed. The concept for tracking the availability of the assets is covered in section 3.3.2.2.1. Briefly, the concept is that the availability of an individual resource can be thought of as a set of concurrent and non-overlapping windows of time. Within each window, the status of the asset is constant. A start time, a status, and any other relevant information define windows. The end of a window occurs when the next window in the list starts. For this formulation, the last element in the list, signifies the end of the list, and the time after which the asset is not available.

The process by which a function determines if an asset can be tasked to an activity, within the given start window, and for the given duration, is as follows. The function first goes through the status windows that end before the earliest start time. Next, it starts to check all windows that end before the latest start time minus the duration. Each of these windows is first checked to see if the status is available. If it is

available, the function checks to see if there is a period of time within the given window that can accommodate the activity within the start window requirement. If so, then this earliest start time is returned. Otherwise, if none of the windows checked can task the activity, then a negative is returned.

The other primary function involves attempting to task the asset at a particular time. This involves checking to see if there is a big enough window of available time at the given start time, and for the needed duration. If there is, then the dynamic list is modified, possibly by inserting new status window objects, to reflect the fact that the asset is no longer available when it is tasked to the activity. If the asset can not accommodate the activity, then a negative response is returned.

The initialization and re-initialization of the lists of status windows involves setting them to be equivalent to the flying day. The list is composed of two elements, the first indicating the start of the flying day, the second indicating the end of the flying day. Any extra elements are stored to be used again later. The reason that the extra elements are stored, rather then de-allocated is that while Java provides automatic garbage collection (the process of freeing up memory no longer in use), the garbage collection only occurs when the system is idle. This means that if calculations are being done for an extended period of time, and memory is constantly being allocated and then de-allocated, the amount of free memory can diminish. This is because while the memory has been de-allocated, the memory manager has not gone through and determined that the memory can used for something different.

3.4. Solution Engine

The purpose of the solution engine is to search through the solution space, defined by the combat scheduling data structure, to find good taskings of aircraft to targets, based on one or more objectives. For the purpose of this research, the objective is to schedule as many of the missions as possible. While meeting the given constraints. The secondary objective is to minimize the length of the schedule. Three methods are explored and combined for finding solutions to the combat planning problem formulated. The first method is using genetic algorithms to explore the solution space. The second method is the generation of starting solutions that the genetic algorithm attempts to improve. The starting solutions are generated through simple heuristics. The third method is in fine tuning scheduling solutions in an attempt to schedule a couple more missions.

3.4.1. Genetic Algorithm

The genetic algorithm used is based on the one developed by Hartman, and expanded by Van Hove. In both of their formulations, the genotype is composed of two lists. The first list is of the order in which activities are scheduled. The second is a list of modes the associated activity is executed in.

To this end, the genotype contains three lists. The first list is the order in which resources will be tasked to accomplish different activities. The second and third lists are an associated preferred activity and the mode the activity needs to be executed in, to use this resource. The term preferred activity indicates that this is the activity that this resource will accomplish.

Figure 40 outlines the main steps of how the genetic algorithm is used to search the solution space. First, the starting solutions are created. For now, these are assumed to be randomly built genotypes (section 3.4.2 discusses using heuristics to build good starting solutions). A schedule is then built for each of the genotypes and the resulting score is associated with the particular genotype using the process outlined in flowchart A, shown in Figure 36 on page 66, with the following addition. Once the activities listed in the genotype are added to the schedule, the method loops through the activities and attempts to schedule the activities that are unscheduled at the earliest time possible. This is added since the crossover and mutation operators used do not guarantee that all of the activities are listed in the genotype.

Once the schedule is built, the score of the schedule is determined through a combination of the percentage of activities scheduled, and the time at which all activities are completed. The next step involves using crossover and mutation operators to double the population size. A new schedule is then built for each of these new genotypes, again using the process outlined in flowchart A, with the schedule scores being associated with the genotype. Next, the population of genotypes is reduced by half, saving the genotypes with the highest associated score. The stopping criteria is either no improvement in the top score over a set number of generations, or the maximum number of generations has been reached. The final step involves displaying the top solution(s).

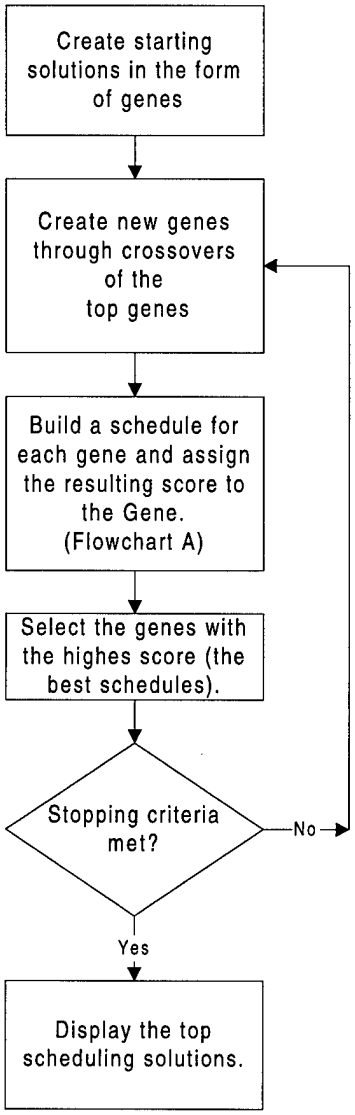


Figure 40 Use genetic algorithms to generate better schedules

3.4.1.1 Crossovers

The crossover operator is the process by which two parent genotypes are used to generate two new children genotypes. The process is as follows. First, the function makes a copy of the two parent genotypes. Next, a location is picked along the same point of the two copies. The information contained within the genotypes after this point is then swapped between the two copied parents. Two locations can also be picked from

along the copies. In this case, the information between the two locations is swapped between the two copied parents. Unfortunately, since only simple crossovers are implemented in the methodology, the resulting "children" genotypes may contain activities that are not in a precedence feasible order, and may contain some activities listed multiple times and others not listed. To work around this problem, once the schedule has been built from a genotype, the activities are looped through. For any activity that is not scheduled, the scheduler attempts to schedule the activity in the earliest possible mode that completes the activity at the earliest time.

3.4.1.2 Mutations

Once the children genotypes are generated through the crossover operator mentioned in the previous section, mutations are performed on the new genotypes. The purpose of the mutations is to add some more variation in the solution space being searched. The mutations also serve to help fine tune solutions. The purpose the mutations are to make a small random change to the genotype, which help fine tune the solution. Currently the following mutation operators are performed on the genotypes.

The first mutation picks an element of the genotype and randomly changes its value to a new resource, activity, and the mode the activity needs to be executed in to use the resource. The next mutation swaps the location of two elements in the genotype. The third mutation picks a random element of the genotype and inserts it at a random location along the genotype. One of the mutations is performed on each of the new genotypes generated.

3.4.2. Starting Solutions

The following functions are used to generate starting solutions, in the form of a genotype, which will then be evolved using the genetic algorithm described in the previous section. Multiple methods are used to generate different genotypes so that multiple areas of the solution space are searched. Plus, with crossovers, the "good" portions of different genotypes can be combined to generate a better solution.

For the following procedures, when referring to adding an element to the gene, the first element added is the first element of the genotype, the next is the second, and so on. For many of the procedures, the elements of the genotype can either be just the resource, or the resource and the preferred activity just scheduled. Both of which will generate the same schedule; however, mutations and crossovers will produce different results, with there being less of an affect on the genes built with both the resource and the preferred activity.

3.4.2.1 Starting Solution #1

This procedure uses the maximum number of times a resource could be chosen, to build the genotype. Figure 41 contains the pseudo-code for this procedure. The following is the general procedure used to create the genotype. The first thing this procedure does is to loop through all of the different activities, checking which resource each mode requires, and the number of assets needed. Through this the minimum number of assets a resource can be required to allocate to accomplish an activity is determined, and the number of times the resource is listed for different modes is counted as well. To get a rough estimate of the maximum number of times a resource should be listed in a gene, the minimum of the following two values is taken: 1) The number of

times a resource is listed for different modes. 2) The maximum number of sorties as set by a resource, divided by the minimum number of assets that can be allocated for an activity, rounding up to the nearest integer.

The first element of the genotype equals the first resource on the list that has a value that indicates it should be on the list at least one more time. The number of times the resource should be in the gene is then decremented by one. The next element of the genotype equals the next resource on the list, based on the same criteria. When the end of the resource list is reached, the procedure goes back to the beginning of the list. This is repeated until no more resources should be added to the list. Note: No preferred activities are on this gene.

```
-Loop j from 0 to the number of resources –1
  --Set maxUse[j] equal to zero
  --Set minSingUse[j] equal to the # of assets for resource j
-Loop j from 0 to the number of activities –1
  --Loop k from 0 to the number of modes in activity j
    ---Increment maxUse[ Id of resource needed for activity j in mode k ] by 1
    ---If the number of assets needed in the mode is less than minSingUse[Id of resource need for activity j
       in mode k ]
        ----Set minSingUse[Id] equal to the number of assets needed by the mode
-Set totalNumElements equal to 0
-Loop j form 0 to the number of resources –1
  --Set maxUse[j] equal the minimum of maxUse[j] and the maximum number of sorties as set by resource
    j
  --Set maxUse[j]  equal to maxUse[j] divided by minSingUse[j], and rounded up to the nearest integer
  --Increment totalNumElements by maxUse[j]
-Set counter equal to zero
-Initialize genotype to be of length totalNumElements
-Set elementCounter equal to 0
-Loop elementCounter from 0 to totalNumElements –1
  --While maxUse[counter] is less than zero
    --Increment counter by 1
    --If counter is greater than or equal to the number of resources
        ---Set counter equal to 0
  --Decrement maxUse[counter] by 1
  --Set genotype element[elementCounter] equal to resource counter
-Return completed genotype
```

Figure 41 Pseudo Code Starting Solution #1

3.4.2.2 Starting Solution #2

This starting solution is generated through a greedy procedure that attempts to finish activities as soon as possible, starting at the first activity and working through all of the activities. Once an activity is scheduled, it is assumed locked and is not changed.

Figure 42 contains the pseudo code used in this function. Following is the generalized method used. For the first activity in the list, determine which mode can complete the activity at the earliest time. The activity is then scheduled in that mode. The resource corresponding to the mode is added to the gene. The procedure then determines the "best" mode for the next activity. If the activity can not be scheduled in any mode, a counter is incremented, and the activity is skipped. This continues until the end of the activity list is reached. If there were any activities not scheduled, for each activity the procedure adds the resource used in the first mode of the activity

```
-Loop j from 0 to the number of activities –1
  --Set earliestFinished equal to a "large" value
  --Set earliestMode equal to –1
  --Loop k from 0 to number of modes for activity j –1
    ---Get the window of time within which activity j can be scheduled (s. t. the set window precedence
       constraints)
    ---Determine the earliest activity j could be schedule to the resource
    ---If the activity can be scheduled within the window
        ----Set tempFinished equal to the time started plus duration of the mode (or plus legs one through?)
        ----If tempFinished is less than earliest finished
            -----Set earliestFinished equal to tempFinished
            -----Set earliestMode equal to k
  --If earliestMode is greater than or equal to zero
    ---Schedule activity in earliestMode at time earliestFinished minus duration of mode (or legs one
       through?)
    ---Set the activity to being scheduled, in the earliestMode, and at the start time
    ---Add element to gene consisting of resource used in earliestMode
-Loop j from 0 to the number of activities –1
  --If activity j is not scheduled
    ---Add element to gene consisting of resource used in first mode of activity j
```

Figure 42 Pseudo Code Starting Solution #2

3.4.2.3 Starting Solution #3

To speed up this and other functions, each resource contains a list of activities that the resource could be tasked to accomplish. The idea behind this function is to attempt to spread out the utilization of resources.

The pseudo code for this function is contained in Figure 43. Following is a description of the process. First this function loops through all of the activities, and generates the total number of sorties listed for each resource. Next a percent possible utilization is calculated for each resource. This is done by dividing the total number of sorties by the maximum number of sorties as set by the resource (if the maximum number of sorties is zero, then a value of –1 is used). The resource with the lowest percentage, and that is greater than zero, is found. The activity with the lowest Id number is then found that can be scheduled by the previous resource. The resource Id is added as an element to the genotype. For the resource just scheduled, both the total number of sorties listed and the maximum number of sorties for the resource are decremented (a separate array needs to be set up so that the actual values are not changed). A new percent possible utilization is then calculated for the resource. For the other modes of the activity scheduled, the other modes are looped through. The resource corresponding to each mode has their total number of sorties decremented by the number of sorties needed by the mode, and a new percent possible utilization is calculated. If no activities can be scheduled by the mode, then the maximum number of sorties and the percent possible utilization are set to –1. The list is again searched as the process repeats. The loop stops when there are no more percentages greater then zero. If there were any activities not scheduled, for each activity the procedure adds the resource used in the first mode of the activity.

```
-Loop j from 0 to the number of resources –1
  --Set sortiesRes [j] equal to 0
-Loop j from 0 to the number of activities –1
  --Loop k from 0 to the number of modes for activity j
    ---Increment sortiesRes [resource used by mode] by number of sorties used by mode
-Loop j from 0 to the number of resources –1
  --If the maximum number of sorties for the resource is greater than 0
    ---Set percentPosUtil [j] equal to sortiesRes [j] divided by the maximum number of sorties defined by
       resource j
  --Else
    ---Set percentPosUtil [j] equal to –1
-Set done equal to false
-Loop while done equals false
  --Set lowestPercent equal to "large" value
  --Set lowestId to –1
  --Loop j from 0 to the number of resources –1
    ---If percentPosUtil [j] is less than lowestPercent and greater than 0
       ----Set lowestPercent equal to percentPosUtil [j]
       ----Set lowestId equal to j
  --If lowestId is less than 0
    ---Set done equal to true
  --Else
    ---Set found equal to the value false
    ---Loop j from 0 to number of possible activities for resource and while found equals false
       ----If possible activity [j] is not scheduled and the number of sorties available is less than or equal
          to the number needed
          -----Get the window of time within which the activity can be scheduled in the mode
               corresponding to the resource (s. t. the set window and precedence constraints)
          -----Determine the earliest the preferred activity can be scheduled
          -----If the activity can be scheduled within the window
                ------Schedule the window to the resource at the time returned
                ------Set activity to being scheduled and at the start time
                ------Add element to gene consisting of resource used in earliestMode
                ------Set found equal to true
                ------Loop k from 0 to the number of modes for the activity
                      -------Decrement sortiesRes [resource of mode k] by the sorties used by mode k
                      -------If current number of sorties available for resource is greater than 0
                            --------Set percentPosUtil [j] equal to sortiesRes divided my the current
                                  number of sorties available for the resource
                            --------Set percentPosUtil [j] equal to –1
    ---If found equals false
       ----Set percentPosUtil [lowestId] equal to –1
-Loop j from 0 to the number of activities –1
  --If activity j is not scheduled
    ---Add element to gene consisting of resource used in first mode of activity j
```

Figure 43 Pseudo Code Starting Solution #3

3.4.2.4 Starting Solution #4

This function is similar to the previous method of spreading out utilization of the resource. The difference is that in the previous method the resource with the lowest possible utilization was selected. For this method, the activities are worked through and they are scheduled in the mode with the lowest percent possible utilization.

Figure 44 contains the pseudo code used for this function. Following is a description of the process. First, the percent possible utilization is calculated, in the same manner as the previous function. Next, the first activity on the activity list is chosen, and it is scheduled in the mode corresponding to the resource that has the lowest percent possible utilization and is able to be scheduled to complete the activity. The corresponding resource is added as an element to the gene. If the activity can not be scheduled in any mode, then it is skipped. The percent possible utilization is updated as before. This procedure is completed when all activities on the list have been cycled through. Once all the activities have been looped through, then random elements are added to the end of the genotype for any activity that is not scheduled.

```
-Loop j from 0 to the number of resources –1
  --Set sortiesRes [j] equal to 0
-Loop j from 0 to the number of activities –1
  --Loop k from 0 to the number of modes for activity j
    ---Increment sortiesRes [resource used by mode] by number of sorties used by mode
-Loop j from 0 to the number of resources –1
  --If the maximum number of sorties for the resource is greater than 0
    ---Set percentPosUtil [j] equal to sortiesRes [j] divided by the maximum number of sorties defined by
       resource j
  --Else
    ---Set percentPosUtil [j] equal to –1
-Loop j from 0 to number of activities -1
  --Set allPrior = true
  --Loop j from 0 to the number of precedence constraints minus one or until  allPrior = false
    ---If constraint [j]'s id of other activity is not scheduled
       ----Set allPrior = false
  --If allPrior = true
    ---Set lowestPercent equal to "large" value
    ---Set lowestId to –1
    ---Loop k from 0 to the number of modes for activity j –1
       ----Set tempArrayMode [k] equal to k
       ----Set tempArrayUtil [k] equal to percentPosUtil [resource for mode k]
    ---Loop k from 0 to (the number of modes for activity j) -2
       ----Loop m from k+1 to the number of modes – 1
            -----If tempArrayUtil [m] is less than tempArrayUtil [k]
                 ------Exchange tempArrayUtil [m] and tempArrayUtil [k]
                 ------Exchange tempArrayMode [m] and tempArrayMode [k]
    ---Set done equal to false
    ---Set counter equal to 0
    ---Loop while counter is less than the number of modes and done equals false
       ----If the resource for the mode tempArrayMode [counter] has enough available sorties
            -----Get the window of time within which the activity can be scheduled (s. t. constraints)
            -----Determine the earliest the activity could be scheduled to the resource within the window
            -----If the activity can be scheduled within the window
                 ------Schedule the activity to the resource at the time returned
                 ------Set done equal to true
                 ------Add element to the genotype consisting of resource used in mode
                 ------Loop k from 0 to the number of modes for the activity
                      -------Decrement sortiesRes [resource of mode k] by the sorties used by mode k
                      -------If current number of sorties available for resource is greater than 0
                           --------Set percentPosUtil [j] equal to sortiesRes divided my the current
                                    number of sorties available for the resource
                           --------Set percentPosUtil [j] equal to –1

-Loop j from 0 to the number of activities –1
  --If activity j is not scheduled
    ---Add element to gene consisting of resource used in first mode of activity j
```

Figure 44 Pseudo Code Starting Solution #4

3.4.3. Process of Fine Tuning Solutions

One of the disadvantages of the genetic algorithms approach is that while genetic algorithms work well for generating good solutions quickly, they are not that good for quickly fine tuning solutions. Therefore, the following automated process is used to attempt to schedule a few more activities.

1. First, the genotype is compressed such that only the elements that schedule activities are listed. Next elements are added to the genotype that corresponds to looping through the unscheduled activities and attempting to schedule them. The rest of the genotype consists of looping through the activities a final time and adding the activities that are still unscheduled to the genotype in the order they appear on the activity list.
2. Activities are scheduled in the order prescribed by the genotype until an activity is reached that can not be scheduled. If all the activities are scheduled, then exit the process.
3. The activity that could not be scheduled becomes the driving activity and the resources that could be used by the activity are stored.
4. Unschedule the activity that was last scheduled and is still scheduled. Store the mode it was scheduled in as an attempted mode for the activity.
5. If the activity was using a resource that could have been used by the driving activity, then attempt to schedule the activity in a mode that has not already been attempted. If successful go to step 2.
6. Remove all the attempted modes for the activity. If there are any activities that remain scheduled, go to step 4.

The stopping criteria for this process includes, a time limit, one hundred percent of the missions being scheduled, or the process was exited due to all of the activities being unscheduled.

3.5. Displaying Generated Schedules

The generated schedules are currently displayed through text report files. Currently there are two main reports generated, a mission report and an asset report. The mission report lists each mission and whether it was scheduled or not. If it was scheduled, the report lists which squadron and base where used, the start time and the end time of the mission. The asset report loops through the different squadrons, and for each aircraft, it lists the missions and start and stop times.

3.6. Summary

This chapter presented the methodology developed for solving combat planning problems, and the framework for a computer-based tool that aids combat planners in tasking air resources to targets. Chapter 4 presents a case study to demonstrate the capabilities of the methodology and the scheduling tool. Chapter 5 provides a summary of the research, the significant contributions, and recommendations for future work.

4. Case Study

The purpose of this chapter is to apply the prototype tool this research developed to a set of one hundred notional combat problems and to demonstrate the applicability of the approach and the tool. This chapter includes how the test problems were generated, the different techniques used, and the performance of the techniques. The purpose of this case study is to be a proof of concept and a demonstration of the prototype tool generated.

4.1. Problem Generation

The basis for the case study is source data from an unclassified Thunder simulation of the Gulf War. However, since the purpose of this case study is provide proof of concept and to avoid any possible security problems, the notional data has been used. Generic titles have also replaced aircraft and target names.

Three different notional aircraft types are used in the case study. Their capabilities are listed in Table 2. The speed value given is used to calculate the amount of time it will take the aircraft to go from the base to the FLOT, to the target, and then back to base. The average recovery time, in hours, is the time between when the aircraft lands and when it is again available for a new mission.

Table 2 Notional Case Study Aircraft Types

	Speed (knots)	**Average Recovery Time (hours)**
Aircraft 1	450	0.8
Aircraft 2	500	1.0
Aircraft 3	550	1.0

This case study uses five random air bases in Saudi Arabia. Their names and locations are presented below in Table 3. The numbers on the left side correspond to the numbers located on the map in Figure 45.

	Air Base	Latitude	Longitude
1	Dhahran	26	50
2	Jidda	21	39
3	Medina	24	39
4	At Ta'if	21	40
5	Bahrain	26	50

Table 3 Notional Case Study Airbases

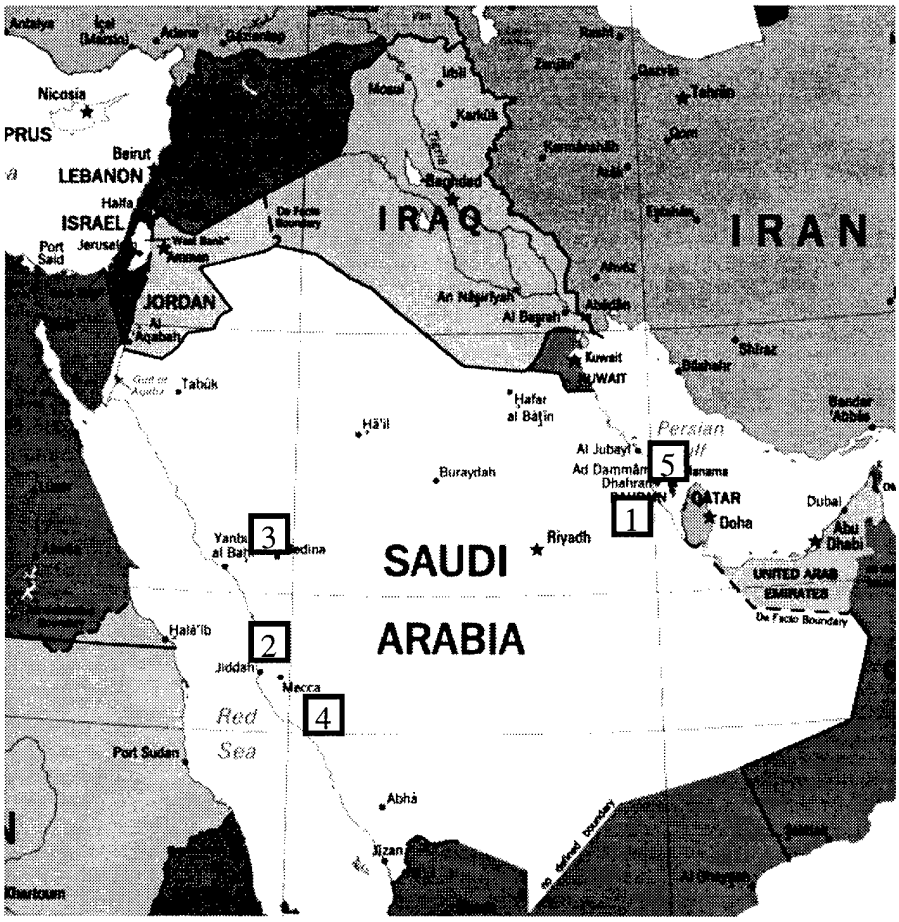


Figure 45 Notional Case Study Map Bases

At the bases, there are ten notional squadrons, each of which is composed of twenty aircraft. However, the number of aircraft available varies between the squadrons. Each squadron can fly a maximum of two Goes, with the maximum number of aircraft listed in Table 4. The reason the availability for each Go is not set to the maximum number of aircraft is to reflect maintenance problems, possible attrition, and other difficulties that are squadron specific and affect the availability of their aircraft. Each squadron has a takeoff window of one hour for each Go, and the recovery time is the average time for the aircraft type. The reference for time starts at midnight, with a value of zero. The flying days are ten hours long and were arbitrarily assigned to reflect how the different squadrons can be ready to fly missions at different times of the day. Each squadron can fly a maximum of two Goes; the number of aircraft in each Go was again arbitrarily assigned.

Table 4 Notional Case Study Squadrons

	Base Stationed At	**Aircraft Type**	**Flying Day (hours)**	**Max Number Aircraft first Go & second Go**
Squad 1	Dhahran	Aircraft 1	5-15	18 & 14
Squad 2	Dhahran	Aircraft 2	7-17	16 & 14
Squad 3	Jidda	Aircraft 1	11-21	18 & 16
Squad 4	Jidda	Aircraft 3	7-17	14 & 14
Squad 5	Medina	Aircraft 2	5-15	18 & 14
Squad 6	Medina	Aircraft 3	12-22	18 & 16
Squad 7	At Ta'if	Aircraft 1	7-17	12 & 10
Squad 8	At Ta'if	Aircraft 2	10-20	16 & 14
Squad 9	Bahrain	Aircraft 1	8-18	18 & 16
Squad 10	Bahrain	Aircraft 3	6-16	14 & 12

For the notional case study, one hundred targets are used. Table 5 below contains the information pertaining to locations of the different targets. Different sites were picked within Iraq to determine the location of the targets. The targets are grouped into

two waves; the first wave contains twenty-five targets, starts at the takeoff of the missions, and ends once all of the aircraft are off their targets. The second wave contains seventy-five targets, starts when the aircraft enter enemy territory, and ends when all the aircraft have landed and are ready for a new mission. The first and second wave columns of the table below contain the number of targets at each location, for the particular wave.

Table 5 Notional Case Study Target Location and Waves

Target Ids	Location	Latitude	Longitude	1st Wave	2nd Wave
1-2	Site 1	33	39	2	
3-4	Site 2	33	40	2	
5-7	Site 3	33	42	3	
8-11	Site 4	32	43	4	
12-14	Site 5	32	44	3	
15-16	Site 6	31	45	2	
17-18	Site 7	30	46	2	
19-22	Site 8	30	47	4	
23-25	Site 9	31	48	3	
26-32	Bayji (a)	35	43		7
33-38	Tikrit (b)	34	42		6
39-46	Samarra (c)	34	43		8
47-50	Karkuk (d)	35	44		4
51-61	Baghdad (e)	33	44		11
62-66	Karbala (f)	33	44		5
67-74	Al Hillah (g)	33	44		8
75-79	Ad Diwaniyah (h)	32	45		5
80-86	Al Kut (i)	33	46		7
87-95	Al Busayyah (j)	30	46		9
96-100	Al Basrah (k)	31	47		5

Figure 46 contains a map that shows the relative locations of all of the targets. Sites one through nine are indicated by the numbers 1 through 9, and the targets around the cities are referenced by the letter shown next to their name in Table 5.

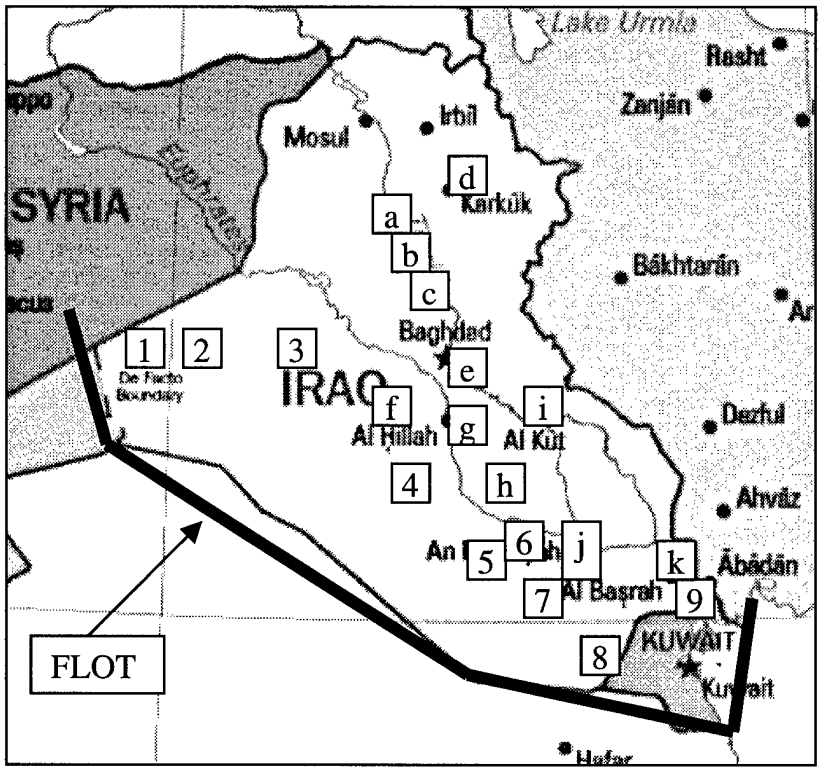


Figure 46 Notional Case Study Map Iraq

Figure 46 also displays the location of the FLOT. The values used for the FLOT are shown in Table 6. These points are used to determine the ingress and egress points of a mission.

Table 6 Notional Case Study FLOT points

	Latitude	Longitude
1	29	49
2	29	48
3	29	45
4	32	39

The weaponeering options for all one hundred targets have been randomly assigned. The number of weaponeering options for a target is randomly assigned to be either two or three, with a fifty percent probability of each occurring. Each of the weaponeering options uses one of the aircraft types, and each allows using any squadron composed of that aircraft type. The number of assets required for each weaponeering

option is randomly assigned through the discrete probability function shown in Table 7. Each of the weaponeering options for a target requires a unique aircraft type. Therefore, if a target has three weaponeering options, then each of the aircraft types is represented by an option.

Table 7 Discrete PDF for Number Of Assets Required

Probability	Number of Assets Required
40%	2
50%	4
10%	6

Random weaponeering options are generated for all one hundred targets to create a sample problem. This process was repeated one hundred times to create a set of one hundred test problems. All test problems have the same bases, squadrons, FLOT, target locations, and wave constraints. Only the weaponeering options vary for each problem. Table 8 shows the weaponeering options for the first five targets of the first problem.

Table 8 Notional Case Study First Five Targets Weaponeering Options

	Option 1		Option 2		Option 3	
Target	Aircraft Type	# of Aircraft	Aircraft Type	# of Aircraft	Aircraft Type	# of Aircraft
1	Aircraft 1	4	Aircraft 2	4		
2	Aircraft 1	4	Aircraft 2	2	Aircraft 3	4
3	Aircraft 1	6	Aircraft 3	2		
4	Aircraft 1	4	Aircraft 2	4	Aircraft 3	6
5	Aircraft 1	2	Aircraft 2	2	Aircraft 3	4

4.2. Results

When generating schedules for the different problems, the only objective used was to schedule as many of the missions as possible. The length of the schedule is constrained by the flying days defined for the individual squadrons. Solutions were

generated using a 400 MHz, DELL Dimension XPS R400, with 256 Megs of RAM. The following seven techniques were used to generate solutions for the scheduling problems in the problem set.

1. Solution generated just through the greedy heuristic in section 3.4.2.2, titled Starting Solution #2. This heuristic sequentially picks activities off the activity list and schedules them to be completed at the earliest time possible. If any activity can not be scheduled, it is skipped. This heuristic is used to generate one of the starting solutions used by the genetic algorithm. This process also took less then one second to generate a solution for each problem and is considered to be the baseline technique.
2. Take the solution generated by the greedy heuristic in technique #1 and run the fine tuning algorithm, covered in section 3.4.3, for 60 seconds.
3. Take the solution generated by the greedy heuristic in technique #1 and run the fine tuning algorithm, covered in section 3.4.3, for 200 seconds.
4. Generate starting solutions for the Genetic Algorithm (GA) and then create multiple generations for 120 seconds. One of the starting solutions used was the solution generated in the greedy heuristic used for technique #1. The genetic algorithm used a population size of 50 (50 parents yields 50 more children. The top 50 of the 100 genotypes become the new parents). In over two minutes, approximately 15 generations were completed.
5. Take the solution generated from the GA in technique #4 and fine tune it for 60 seconds.

6. Run the GA described in technique #4 for 180 seconds, instead of 120 seconds.
7. Take the solution generated from the GA in technique #6 and fine tune it for 60 seconds.

Table 9 below contains the average percent of missions scheduled using each of the seven techniques on the set of one hundred problems.

Table 9 Percentage of Missions Scheduled

	#1 greedy heuristic (1 Second)	**#2 FT (60 Seconds)**	**#3 FT (200 Seconds)**	**#4 GA (120 Seconds)**	**#5 GA (120 Seconds), FT (60 Seconds)**	**#6 GA (180 Seconds)**	**#7 GA (180 Seconds), FT (60 Seconds)**
Mean	91.5%	95.3%	96.0%	97.2%	97.9%	97.5%	98.0%
St. Dev	3.9	3.3	3.1	2.4	2.1	2.4	1.9

While the values seem to indicate that there is little difference between some of the techniques, these results are misleading because they do not take into account that the techniques were each performed on the same set of problems. The values in Table 10 represent the average percent difference between two different techniques. These values were found first by subtracting the percent of missions each technique scheduled for each problem. The 95% confidence interval was then found for this difference. If the confidence interval includes zero, then there is no statistical difference between the two techniques for this problem set. On the table, the average difference is the average difference between the two techniques, the confidence interval is the average difference value plus or minus the "95% Conf. Int."

Table 10 Difference Between Techniques

	#2-#1	#3-#1	#3-#2	#4-#1	#4-#3	#5-#1	#5-#4
Average Difference	3.8%	4.5%	0.7%	5.7%	1.2%	6.5%	0.7%
95% Conf. Int.	0.4%	0.4%	0.2%	0.5%	0.4%	0.5%	0.2%

	#6-#1	#6-#5	#7-#1	#7-#5	#7-#6
Average Difference	6.0%	-0.5%	6.6%	0.1%	0.6%
95% Conf. Int.	0.5%	0.3%	0.5%	0.3%	0.2%

Explanation of Table 10 results:

1. #2-#1: 3.8 ± 0.4 more missions are scheduled by the fine tuning algorithm when it is run on the baseline solution for 60 seconds.
2. #3-#1: 4.5 ± 0.4 more missions are scheduled by the fine tuning algorithm when it is run on the baseline solution for 200 seconds.
3. #3-#2: 0.7± 0.4 more missions are scheduled by the fine tuning algorithm between 60 and 200 seconds.
4. #4-#1: 5.7± 0.5 more missions are scheduled by running the genetic algorithm for 120 seconds with the baseline solution as a starting solution, as compared with just running the baseline solution.
5. #4-#3: 1.2± 0.4 more missions are scheduled by running the genetic algorithm for 120 seconds with the baseline solution as a starting solution, as compared to just fine tuning the starting solution for 200 seconds.
6. #5-#1: 6.5± 0.5 more missions are scheduled running the genetic algorithm for 120 seconds with the baseline solution as a starting solution, and then fine tuning the solution for 60 seconds, as compared to just the baseline solutions.

7. #5-#4: 0.7± 0.2 more missions are scheduled by running the fine tuning algorithm for 60 seconds on the schedule generated by the genetic algorithm for 120 seconds, as compared to just running the genetic algorithm.
8. #6-#1: 6.0± 0.5 more missions are scheduled running the genetic algorithm for 180 seconds with the baseline solution as a starting solution, as compared to just baseline solution.
9. #6-#5: 0.5± 0.3 more missions are scheduled running the genetic algorithm for 120 seconds with the baseline solution as a starting solution, and then fine tuning the solution for 60 seconds, as compared to just running the genetic algorithm for 180 seconds on the baseline solution.
10. #7-#1: 6.6± 0.5 more missions are scheduled running the genetic algorithm for 180 seconds with the baseline solution as a starting solution, and then fine tuning the solution for 60 seconds, as compared to just the baseline solutions.
11. #7-#5: 0.1± 0.3 since this confidence interval includes the value zero, there is no statistical advantage of running the genetic algorithm for an additional 60 seconds after running it the initial 120 seconds, if it will be fine tuned for 60 seconds afterwards.
12. #7-#6: 0.6± 0.2 more missions are scheduled by running the fine tuning algorithm for 60 seconds on the schedule generated by the genetic algorithm for 180 seconds, as compared to just running the genetic algorithm.

From this analysis, the only techniques examined that were statistically different were between running the GA for two minutes and fine tuning for one minute, and

running the GA for three minutes and fine tuning for one minute. It is also interesting to note that running the GA for two minutes and fine tuning for one minute (technique #5) produced slightly statistically significant better results then fine tuning for three minutes (technique #6).

Table 11 contains the number of solutions generated in the problem set that scheduled 100% of the missions; the value is out of 100 solutions generated. It is interesting to note that while technique #5 generated a slightly higher percentage of missions than #6, technique #6 produced more solution with 100% of the activities scheduled.

Table 11 Number of schedules produced with 100% completion

#1 greedy heuristic	#2 FT (60 Seconds)	#3 FT (200 Seconds)	#4 GA (120 Seconds)	#5 GA (120 Seconds), FT (60 Seconds)	#6 GA (180 Seconds)	#7 GA (180 Seconds), FT (60 Seconds)
1	2	2	22	22	27	27

For a further comparison of the differences between the solutions generated by techniques #5 and #6, the results were compared based on the individual problems of the problem set; the results are shown in Table 12. The percentage value in the table represents the percentage of time the techniques scheduled the same percent of activities or that one performed better than the other did. These results indicate that the heuristic for fine tuning will be able to improve the schedule more often than just letting the GA run. However, the few times the GA is able to make an improvement, the improvement is larger than the improvement made by the heuristic for fine tuning the schedule.

Same Result	Technique Five Better	Technique Six Better
38%	40%	22%

Table 12 Comparison of Techniques #5 & #6

To estimate the difficulty of scheduling 100% of the mission for the problems in the problem set, the following value is generated. This value is the fewest number of sorties that are needed to schedule 100% of the missions if there are no constraints on the number of resources, no wave constraints, and no go constraints. A single sortie is defined to be a single aircraft flying a single mission.

Since other constraints exist in the problem, the value generated is the lower bound for the estimate of the minimum number of assets required to schedule 100% of the activities. The value is calculated as follows. For each target, generate the minimum number of aircraft required for any of its weaponeering options. Summing up these minimum values gives the lower bound for the absolute minimum number of sorties required to schedule 100% of the missions for this individual problem. The value calculated is compared to the percent of missions scheduled using technique five. Figure 47 is the resulting graph, and some of the points on the graph represent more then one data point. The maximum number of sorties, as defined by the Go constraints is 302.

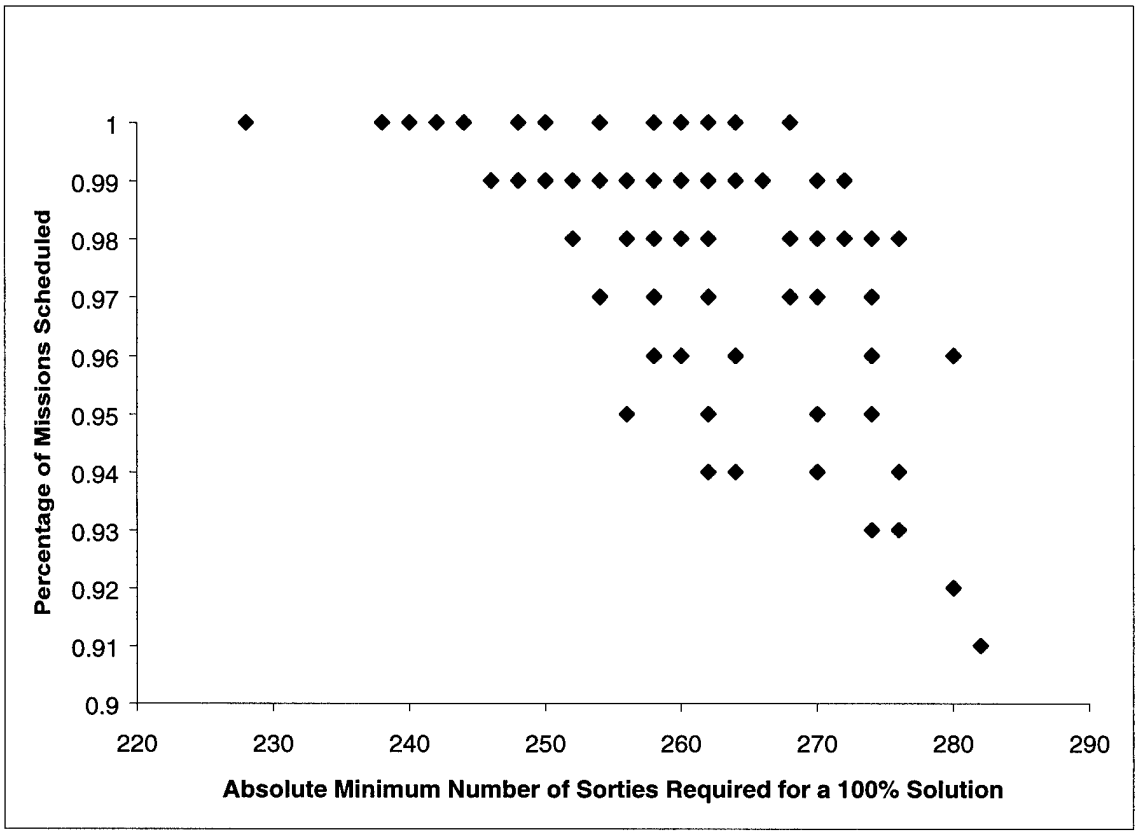


Figure 47 Estimate of Problem Difficulty vs. Percentage Scheduled

As can be seen on the graph, the higher the lower estimated bound is, the more likely it is that a lower percentage of missions will be scheduled. This indicates that the higher the lower estimated bound for the minimum number of sorties is, the more likely it is that fewer if any solutions exist that can result in 100% of the missions being scheduled. The graph also seems to indicate the source of at least some of the variation in the results of the techniques.

4.3. Summary

The results from the case study indicate that the methodology developed is able to make significant improvements to the solution generated by the greedy heuristic. The results also indicate that the methodology is able to handle problems of 300 sorties, and

show promise for handling large problems. While the results are promising, further testing is needed against problems with known optimal solutions, and real world combat scheduling problems need to be tested before any further conclusions can be drawn.

5. Conclusions and Recommendations

5.1. Research

This research did not focus on a specific aspect of the problem of tasking resources to targets. Instead, it incorporates several aspects of the problem: 1) How the combat planner views the problem. 2) Creating a user-friendly interface that allows a combat planner to enter in problem information both easily and quickly. 3) Determine what the goals of the solver should be. In this case, it is to schedule as many of the missions as possible and to minimize the length of the schedule. 4) Be able to generate the scheduling solutions in an acceptable amount of time. 5) Present the generated schedules.

5.2. Contributions

The contributions of this research are in two main areas. The first is how it contributes to the military problem of combat planning. The second area is in contributions to the field of scheduling in general.

5.2.1. Contributions to Combat Planning

The research expands the combat planning formulation to include the concept that squadrons flying missions can be grouped into a Go, where all missions in a Go have to start within a takeoff window. A methodology is presented for generating solutions for the problem of tasking allocated combat aircraft to strike targets that incorporates ideas of project scheduling, object oriented programming, and genetic algorithms. Finally, a prototype GUI for combat scheduling is developed that demonstrates a possible balance

between ease of use, ability to enter in problem data quickly, and the level of detail the problem incorporates.

5.2.2. Academic

This research makes several contributions to the field of project scheduling. The first is an example of an object-oriented approach to modeling and solving project scheduling problems. Along with the objet-oriented approach, this research develops a flexible method for tracking resource availability. This method allows resource usage to be tracked across time, and no time interval needs to be set for when the resource is available.

5.3. Recommendations

The following recommendations are made for extending the research:

1. Incorporate target hardness, damage required, and weapons effectiveness database and allow weaponeering options to be automatically assigned, and then allow the combat planner to adjust the options accordingly.
2. Strike package missions and GUI for combining targets into one of a possible set of "standard strike packages" and the interface then automatically adds the appropriate precedence constraints
3. Include priority classes for different targets. Then attempt to schedule all of the targets of the highest priority first, and then work down to the targets of lower priority. One method of doing this would be to score the schedules based on the percentage of missions in each priority class that are scheduled.

4. Include capability to model cruise missiles. Would require understanding constraints for how many can be launched at a time, the time between launches, and the maximum number of launches.
5. Allow aircraft to takeoff at one base, complete their mission, and then land at another base.
6. Improve the map GUI for better latitude and longitude control as currently latitude and longitude is considered to change constantly both vertically and horizontally. Also improve the loading and saving process of the GUI, to include all the different source information into a project file. Also, include help files.
7. Add the additional constraint of making sure the necessary munitions are available for the missions.
8. Add refueling and other support missions.
9. Develop an interface for interactively building and modifying schedules. This could be used to examine and modify schedules built by the solution engine, and could allow a combat planner to build a schedule from scratch. The best implementation would be where the solution engine builds a schedule and then the combat planner indicates what they do not like about the schedule. If the changes can not easily be made to the schedule, then the solution engine is rerun with the updated problem.
10. Expand the current methods for searching the solution space, and explore other methods: such as an expert system and tabu searches.
11. Package the program for easy installation on PCs and work stations.
12. Allow the drop in the number of aircraft available in a Go to be a function of the types of missions undertook in the previous Go.

13. Allowing importing of information from existing operational databases to limit the amount of data entry required.
14. Determine how this tool could be integrated into the computer aides already in the field.
15. Expand the testing of the tool to include larger sized problems to determine the upper limit for problem size. Also test real world problems and compare the solutions generated by the tool with ones generated manually by combat planners.

5.4. Summary

This research develops a new approach for combat planning. This approach is a combination of project scheduling, object-oriented programming, and genetic algorithms. A prototype interface is also developed, balancing between ease of use, accurately defining the problem, and generating solutions in an acceptable amount of time.

The methodology developed has applications for project scheduling in both the military and the civilian sector. The project scheduling problem this research solves is a multi-modal generalized resource constrained project-scheduling problem (MMGRCPSP) with the addition of the Go constraints. The Go constraints restrict the resources to start activities within a set period of time of each other, and can allow a different quantities of resources to be available during different Goes. While the Go constraint was developed for restricting when a squadron can fly missions, the same principle can be applied to other areas where resources work in groups or batches. One example is shift work, where workers start and stop working at the same time. Another example is where workers need to be transported to and from a remote location where they all arrive at the same

time and all leave at the time. The concept can also be applied to how the availability of resources can be grouped into lots or kits.

Appendix A. Notional Case Study Weaponeering Options Problem #1

Target	Aircraft type	# of assets	Aircraft type	# of assets	Aircraft type	# of assets
Target 1	Aircraft 2	4	Aircraft 1	4		
Target 2	Aircraft 2	2	Aircraft 1	4	Aircraft 3	4
Target 3	Aircraft 3	2	Aircraft 1	6		
Target 4	Aircraft 1	4	Aircraft 3	6	Aircraft 2	4
Target 5	Aircraft 2	2	Aircraft 3	4	Aircraft 1	2
Target 6	Aircraft 1	2	Aircraft 3	4		
Target 7	Aircraft 3	2	Aircraft 2	4		
Target 8	Aircraft 1	2	Aircraft 3	4		
Target 9	Aircraft 1	2	Aircraft 3	4	Aircraft 2	2
Target 10	Aircraft 2	4	Aircraft 1	2		
Target 11	Aircraft 1	4	Aircraft 2	2	Aircraft 3	2
Target 12	Aircraft 3	4	Aircraft 1	6		
Target 13	Aircraft 2	2	Aircraft 3	2		
Target 14	Aircraft 1	4	Aircraft 3	4		
Target 15	Aircraft 1	4	Aircraft 3	2		
Target 16	Aircraft 2	6	Aircraft 3	4	Aircraft 1	4
Target 17	Aircraft 2	4	Aircraft 1	2		
Target 18	Aircraft 1	4	Aircraft 3	4	Aircraft 2	4
Target 19	Aircraft 2	4	Aircraft 1	2		
Target 20	Aircraft 1	2	Aircraft 3	2		
Target 21	Aircraft 1	4	Aircraft 2	4	Aircraft 3	2
Target 22	Aircraft 1	4	Aircraft 2	4		
Target 23	Aircraft 3	2	Aircraft 2	4		
Target 24	Aircraft 3	2	Aircraft 2	2	Aircraft 1	4
Target 25	Aircraft 2	4	Aircraft 1	4		
Target 26	Aircraft 2	4	Aircraft 1	2		
Target 27	Aircraft 2	4	Aircraft 1	4	Aircraft 3	6
Target 28	Aircraft 3	2	Aircraft 1	2	Aircraft 2	2
Target 29	Aircraft 1	4	Aircraft 3	4		
Target 30	Aircraft 2	2	Aircraft 1	2		

Target 31	Aircraft 1	4	Aircraft 2	2		
Target 32	Aircraft 3	4	Aircraft 2	4	Aircraft 1	4
Target 33	Aircraft 3	2	Aircraft 1	4		
Target 34	Aircraft 2	6	Aircraft 1	4	Aircraft 3	4
Target 35	Aircraft 3	4	Aircraft 2	2	Aircraft 1	4
Target 36	Aircraft 1	4	Aircraft 2	2		
Target 37	Aircraft 3	2	Aircraft 2	4		
Target 38	Aircraft 3	4	Aircraft 2	2		
Target 39	Aircraft 3	2	Aircraft 2	4		
Target 40	Aircraft 1	2	Aircraft 2	4		
Target 41	Aircraft 2	6	Aircraft 1	6		
Target 42	Aircraft 2	2	Aircraft 3	2	Aircraft 1	2
Target 43	Aircraft 1	4	Aircraft 3	2	Aircraft 2	4
Target 44	Aircraft 2	4	Aircraft 3	2	Aircraft 1	2
Target 45	Aircraft 1	6	Aircraft 2	4		
Target 46	Aircraft 3	2	Aircraft 1	2	Aircraft 2	2
Target 47	Aircraft 3	4	Aircraft 2	4		
Target 48	Aircraft 1	4	Aircraft 2	4	Aircraft 3	2
Target 49	Aircraft 1	2	Aircraft 2	6	Aircraft 3	6
Target 50	Aircraft 3	2	Aircraft 2	2	Aircraft 1	4
Target 51	Aircraft 2	4	Aircraft 1	4		
Target 52	Aircraft 1	2	Aircraft 2	6	Aircraft 3	4
Target 53	Aircraft 2	6	Aircraft 3	4		
Target 54	Aircraft 2	2	Aircraft 3	4		
Target 55	Aircraft 2	4	Aircraft 3	4		
Target 56	Aircraft 3	2	Aircraft 2	2	Aircraft 1	2
Target 57	Aircraft 3	2	Aircraft 1	4		
Target 58	Aircraft 1	4	Aircraft 3	4		
Target 59	Aircraft 2	2	Aircraft 1	4	Aircraft 3	2
Target 60	Aircraft 3	4	Aircraft 1	2	Aircraft 2	4
Target 61	Aircraft 1	2	Aircraft 2	4	Aircraft 3	4
Target 62	Aircraft 3	4	Aircraft 2	6	Aircraft 1	2
Target 63	Aircraft 1	4	Aircraft 2	4	Aircraft 3	2
Target 64	Aircraft 1	2	Aircraft 2	4		
Target 65	Aircraft 2	4	Aircraft 3	4	Aircraft 1	4
Target 66	Aircraft 3	2	Aircraft 1	4		
Target 67	Aircraft 3	4	Aircraft 1	4		
Target 68	Aircraft 3	4	Aircraft 1	4	Aircraft 2	6
Target 69	Aircraft 2	2	Aircraft 1	4	Aircraft 3	4
Target 70	Aircraft 1	2	Aircraft 3	4		
Target 71	Aircraft 1	2	Aircraft 2	4	Aircraft 3	2
Target 72	Aircraft 1	4	Aircraft 3	4	Aircraft 2	4
Target 73	Aircraft 2	2	Aircraft 1	2	Aircraft 3	2
Target 74	Aircraft 1	2	Aircraft 2	4	Aircraft 3	4
Target 75	Aircraft 1	4	Aircraft 2	2	Aircraft 3	6

Target 76	Aircraft 2	6	Aircraft 1	2	Aircraft 3	2
Target 77	Aircraft 3	4	Aircraft 1	2		
Target 78	Aircraft 1	4	Aircraft 2	4	Aircraft 3	4
Target 79	Aircraft 1	4	Aircraft 2	2		
Target 80	Aircraft 2	4	Aircraft 1	4		
Target 81	Aircraft 2	4	Aircraft 3	4		
Target 82	Aircraft 2	2	Aircraft 1	4		
Target 83	Aircraft 2	4	Aircraft 3	4		
Target 84	Aircraft 2	2	Aircraft 1	2		
Target 85	Aircraft 1	2	Aircraft 3	4		
Target 86	Aircraft 2	4	Aircraft 3	4		
Target 87	Aircraft 1	4	Aircraft 2	4		
Target 88	Aircraft 3	2	Aircraft 1	2		
Target 89	Aircraft 3	4	Aircraft 2	2		
Target 90	Aircraft 3	2	Aircraft 2	4		
Target 91	Aircraft 3	2	Aircraft 2	2	Aircraft 1	6
Target 92	Aircraft 2	2	Aircraft 3	4	Aircraft 1	2
Target 93	Aircraft 3	2	Aircraft 1	2		
Target 94	Aircraft 1	4	Aircraft 2	4		
Target 95	Aircraft 1	2	Aircraft 2	4		
Target 96	Aircraft 1	4	Aircraft 3	2	Aircraft 2	4
Target 97	Aircraft 3	4	Aircraft 1	2	Aircraft 2	2
Target 98	Aircraft 1	2	Aircraft 3	4	Aircraft 2	2
Target 99	Aircraft 3	2	Aircraft 1	6	Aircraft 2	2
Target 100	Aircraft 3	2	Aircraft 2	4		

Appendix B. Source Code

The source code for the prototype tool generated is not included as part of this document. The main points of the code have been included in the methodology section. Those interested in obtaining a copy of the source code should direct their request to:

Dr. Dick Deckro
AFIT/ENS
2950 P St.
WPAFB, OH 45433-7765
Richard.Deckro@afit.af.mil

Bibliography

AFI 13 AOC: Instruction Governing the AF Air Operations Center

Bailey, Glenn. Course notes Oper 595. Spring 1999.

Carruthers, J. A. and A. Battersby. "Advances in Critical Path Methods," *Operations Research Quarterly, 17*:359-380 (1966).

Boctor, Fayez F. "A New and Efficient Heuristic for Scheduling Projects With Resource Restrictions and Mutiple Execution Modes," *European Journal of Operational Research, 90*(2):349-361 (April 1996).

Bowman, Edward H. "The Scheduling-Sequencing Problem," *Operations Research*, 7(5):621-624 (1959).

Chambers, Lance. *Practival Handbook of Genetic Algorithms Applications Volume 1.* New York: CRC Press, 1995.

Chambers, Lance. *Practival Handbook of Genetic Algorithms New Frontiers Volume 2.* New York: CRC Press, 1995.

Chretienne, Philippe, Edward G. Coffman, Jr, Jan Karel Lenstra, and Zhen Lin, editors. *Scheduling Theory and its Applications.* New York: John Wiley and Sons, 1995.

Christofides, N., R. Alcares-Valdes, and J. M. Tamarit. "Project Scheduling with Resource Constraints: A Branch and Bound Approach," *European Journal of Operational Research, 29*(3):262-273 (June 1987).

Coad, Peter, and Jill Nicola. *Object-Oriented Programming.* Englewook Cliffs, NJ: Yourdon Press, 1993.

Dantzig, George B. and Philip Wolfe. "Decomposition Principle for Linear Programs," Operations Research, 8:101-111(1960).

Davis, E. W. and G. E. Heidorn. "Optimal Project Scheduling under Multiple Resource Constraints," *Management Science, 17*(12):B803-B816 (August 1971).

Demeulemeester, E. "Minimizing Resource Availability Costs in Time-Limited Project Networks," *Management Science, 41*(10):1590-1598 (October 1995).

Demeulemeester, E., B. Dodin, and W. Herroelen. "A Random Activity Nework Generator," *Operations Research, 41*(5): 972-980 (September-October 1993).

Demeulemeester, E., W. Herroelen, W. P. Simpson, S. Baroum, J. H. Patterson, and Kum-Khiong Yang. "On a Paper by Christofides et al. For Solving the Multiple-

Resource Constrained, Single Project Scheduling Problem," *European Journal of Operational Research, 76*(1):218-228 (July 1994).

Demeulemeester, E. L. and W. S. Herroelen. "An Efficient Optimal; Solution Procedure for the Preemptive Resource-Constrained Project Scheduling Problem," *European Journal of Operational Research, 90*(2):334-348 (April 1996).

Hartmann, Sonke. Project Scheduling with Multiple Modes: A Genetic Algorithm, Manuskripte aus den Instituten fur Betriebswirtschaftslehre, No. 435, University of Kiel, Germany, March 1997.

Herrolelen, Will S. "Resource-Constrained Project Scheduling—State of the Art," *Operational Research Quarterly, 23*(3):261-275 (1972).

Hinton, Donald W. *A Decision Support System For Joint Force Air Component Commander (JFACC) Combat Planning*. MS thesis, AFIT/ENS/GOA/97M-09. School of Engineering, Air Force Insitute of Technology (AU), Wright-Patterson AFB OH, March 1997.

Hinton, Donald W. *Personal Conversation*, Air Force Wargaming Institute, CADRE, USAF, Maxwell AFB, 17 Dec 1998.

Icmeli, Olya, and S. Selcuk Erenguc. "A Branch and Bound Procedure for the Resource Constrained Project Scheduling Problem with Discounted Cash Flows," *Management Science, 42*(10):1395-1408 (October 1996).

Icmeli, Oya, and Walter O. Rom. "Solving The Resource Constrained Project Scheduling Problem With Optimization Subroutine Library," *Computers Operations Research, 23*(8):801-817 (1996).

Kelley, James E. "Critical-Path Planning and Scheduling: Mathematical Basis,: *Operations Research, 9*(3):296-320 (May-June 1961).

Kolisch, Rainer and R. Padman. An Integrate Survey of Project Scheduling. Manuskripte aus den Instituten fur Betriebswirtschaftslehre, No. 463, University of Kiel, Germany, July 1998.

Kolisch, Rainer and Hartmann, Sonke. Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis,

Kolisch, Rainer and Arno Sprecher. PSPLIB – A Project Scheduling Problem Library. Manuskripte aus den Instituten fur Betriebswirtschaftslehre, No. 396, University of Kiel, Germany, March 1996.

Kolisch, Rainer, Arno Sprecher, and Andreas Drexl. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. Manuskripte

aus den Instituten fur Betriebswirtschaftslehre, No. 301, University of Kiel, Germany, December 1992.

Kolisch, Rainer, Christoph Schwindt and Arno Sprecher. "Benchmark Instances for Project Scheduling Problems." Handbook on Recent Advances in Project Scheduling.

Lined, Peter van der. *just JAVA 1.1 and Beyond.* 3rd ed. California: Sun Microsystems Press, 1998.

Ozdamar, Linet. A Genetic Algorithm Approach to a General Category Project Scheduling Problem. Research Report, Marmara University, Istanbul, 1996.

Ozdamar, Linet and Gunduz Ulusoy. "A Local Constraint Based Analysis Approach to Project Scheduling under General Resource Constraints," *European Journal of Operational Research, 79*:287-298 (1994).

Patterson, James H. "A Comparison of Exacy Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem," *Management Schince, 30*(7):854-867(July 1984).

Petrovic, R. "Optimization of Resource Allocation in Project Planning," Operations Research, 16:559-586(1968).

Pritsker, Alan B., Lawrence J. Watters, and Phillip M. Wolfe. "Multi-Project Scheduling with Limited Resources: a Zero-One Programming Approach,: *Management Science, 16*(1):93-108(September 1969).

Shtub, Avraham, Jonathan F. Bard, and Shlomo Globerson. *Project Management: Engineering, Technology, and Implementation.* Englewood Cliffs, NJ: Prentice Hall, 1994.

Slowinski, Roman, Boleslaw Soniewicki, and Jan Weglarz. "DSS for Multiobjective Project Scheduling," *European Journal of Operation Research, 79:*220-229(1994).

Sweeney, Dennis J. and Richard A. Murphy. "A Method of Decomposition for Integer Programs," Operations Research, 27(6):1128-1141(November 1979).

Talbot, F. Brian. "Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive case," *Management Science, 28*:1197-1210(1982).

Talbot, F. Brian and Hames H. Patterson. "An Efficient Integer Programmin Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems," *Management Science, 24*(11):1163-1175 (July 1978).

Tillotson, David III. *Restructuring the Air Operations Center*, AU-ARI-92-1. Air University Press; Maxwell Air Force Base, Alabama: Air University, April 1993.

Ulusoy, Gunduz and Linet Ozdamar. "A Framework For an Interactive Project Scheduling System Under Limited Resources," *European Journal of Operational Research, 90:*362-375 (1996).

Van Hove, John C. *An Integer Program Decomposition Approach to Combat Planning.* Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1998

Vita

2Lt David Koewler was born in Hastings, Minnesota on 21 April 1975. He received his undergraduate degree in Operations Research from the United States Airforce Academy in 1997. His first assignment was to graduate school at the Air Force Institute of Technology, where he received a Masters in Operation Research in 1999. Upon graduation, he was assigned to the Air Force Wargaming Institute (CADRE) at Maxwell AFB, AL.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1999	Master's Thesis

4. TITLE AND SUBTITLE
An Approach for Tasking Combat Resources to Targets

5. FUNDING NUMBERS

6. AUTHOR(S)
David A. Koewler, 2nd Lieutenant, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
2950 P Street
WPAFB, OH 45433-7767

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/99M-08

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
Dr. Neal Glassman
801 North Randolph Street Room 732
Arlington VA 22203-1977

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Tasking allocated combat aircraft to strike targets is a complicated and time-consuming process for combat planners. Currently, the process of scheduling missions is a two to four day process. To be able to respond quickly to the changing conditions of the battlefield, the military needs to compress the time that this process requires. Despite efforts to develop computer-based tools to automatically plan missions, combat planners still manually perform most of the tasking and scheduling of aircraft and targets. Unfortunately some of the tools currently available are perceived to be complicated and time consuming to use by the planners. They also generate schedules that often require major manual modifications. Unfortunately, this means the combat planners often feel it is easier and faster to schedule the missions manually.

With these issues in mind, this research created a prototype-scheduling tool that strikes a balance between ease of use, accurately defining and solving the problem, and generating solutions in an operationally useful amount of time. A combination of concepts from project scheduling, object-oriented programming, heuristics methodology, and genetic algorithms form the basis for the methodology developed in the research. This methodology is then demonstrated in a computer-based tool. While the focus of this research is combat planning, the object-oriented approach developed for project scheduling has applications in both the military and civilian sector. This includes a flexible method for modeling the availability of resources.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Air Campaign Planning, Scheduling, Job Shop Scheduling, Heuristic Methods	129
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL