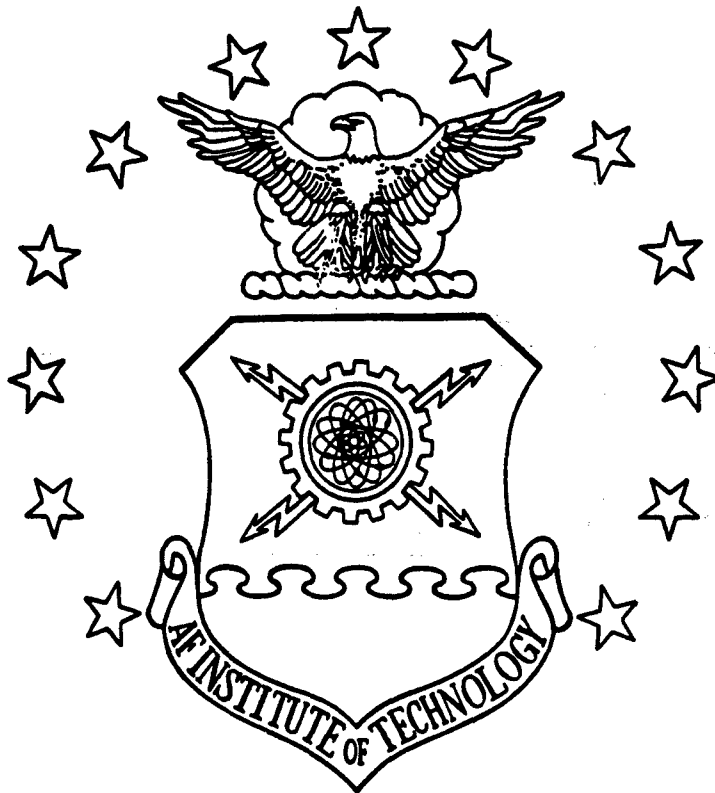




*SENSITIVITY ANALYSIS OF THE THUNDER COMBAT
SIMULATION MODEL TO COMMAND AND CONTROL
INPUTS ACCOMPLISHED IN A PARALLEL ENVIRONMENT*

THESIS

David A. Davies, Major, USAF

AFIT/GOA/ENS/98M-02

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DTIC QUALITY INSPECTED 4

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

19980427 144

AFIT/GOA/ENS/98M-02

*SENSITIVITY ANALYSIS OF THE THUNDER COMBAT
SIMULATION MODEL TO COMMAND AND CONTROL
INPUTS ACCOMPLISHED IN A PARALLEL ENVIRONMENT*

THESIS

David A. Davies, Major, USAF

AFIT/GOA/ENS/98M-02

Approved for public release; distribution unlimited

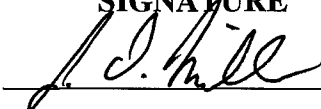
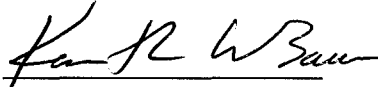
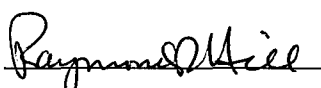
AFIT/GOA/ENS/98M-02

THESIS APPROVAL

NAME: David A. Davies, Major, USAF **CLASS:** GOA-98M

THESIS TITLE: *Sensitivity Analysis of the THUNDER Combat Simulation Model to Command and Control Inputs Accomplished in a Parallel Environment*

DEFENSE DATE: 4 March 1998

COMMITTEE:	NAME/TITLE DEPARTMENT	SIGNATURE
Advisor	John O. Miller, Lt Col, USAF Assistant Professor of Operations Research Department of Operational Sciences	
Reader	Kenneth Bauer Professor Department Of Operational Sciences	
Reader	Raymond R. Hill, Major, USAF Assistant Professor of Operations Research Department of Operational Sciences	

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/GOA/ENS/98M-02

***SENSITIVITY ANALYSIS OF THE THUNDER COMBAT
SIMULATION MODEL TO COMMAND AND CONTROL
INPUTS ACCOMPLISHED IN A PARALLEL ENVIRONMENT***

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 Analysis**

**David A. Davies, M.S.
Major, USAF**

March, 1998

Approved for public release; distribution unlimited

ACKNOWLEDGEMENTS

This thesis was not an individual effort. There were many people who helped me with my struggle and without their help this project would, in all likelihood, not have been completed. First and foremost were my wife Marie, and our two children Sarah and Andrew. Never once did they begrudge me the countless hour I spent in the **S**imulation & **M**odeling room at AFIT.

Of course, I would like to thank my advisor, Lt Col J.O. Miller for his unending patience and support throughout this project and his many attempts to extricate me from my own personal version of Dantes' Perl Inferno. I would also like to thank my readers: Dr. Ken Bauer, for his encouragement, guidance and many saves during my trials and Maj Ray Hill for his meticulous attention to detail in helping me to prepare the best possible product.

I am also deeply indebted to Walt Hobbs and Clark Dorman. As an apprentice programmer it is always a joy to work under the masters. Their guidance and help during the development of MMMP was invaluable.

I can't begin to say enough about my classmates. Without their help throughout the entire AFIT experience, I certainly would not be where I am today. The days of going it alone are forever buried in history.

Finally, Dan Schornak and all of the people at the MSRC were more than patient with me as I slowly learned the nuances of working in a UNIX environment. This work was supported in part by a grant of time from the DoD HPC at Wright Patterson AFB.

Table of Contents

ACKNOWLEDGEMENTS.....	ii
List of Figures	vi
List of Tables.....	vii
Abstract.....	ix
1. Introduction	1
1.1 Background	1
1.2 Simulation and Analysis Facility (SIMAF)	4
1.3 Major Shared Resource Center (MSRC)	5
1.4 Problem Statement.....	6
1.5 Scope.....	7
1.6 Limitations and Assumptions.....	8
1.6.1 Unclassified Data.....	8
1.6.2 Scenario	9
1.6.3 Scenario Data Files	9
1.7 Thesis Overview	9
2. Literature Review.....	10
2.1 Previous THUNDER Studies	10
2.2 Modeling and Simulation (M&S).....	11
2.3 C2 Modeling	12
2.4 Parallelization.....	13
2.5 THUNDER Overview	16
2.5.1 General.....	16
2.5.2 Campaign Planning	16
2.5.3 Model Fidelity	17
2.6 Operational Effectiveness Analysis	17
2.7 Sensitivity Analysis of THUNDER to C2 Inputs.....	19
2.8 Response Surface Methodology	20
2.9 Experimental Design Selection	22
2.9.1 Random Numbers	26
2.9.2 Factorial Designs	27
2.9.3 Coding.....	28

2.9.4	Fractional Factorial Design	28
2.9.5	Plackett-Burman Designs.....	30
2.9.6	Resolution	31
2.10	Regression Analysis	33
2.10.1	Coefficient of Multiple Determination	35
2.10.2	Residual Analysis.....	36
2.10.3	F test for Regression Relation	38
2.10.4	t- test.....	40
2.10.5	Lack of Fit test	41
2.10.6	Metamodel Validation.....	43
2.11	Output Measures.....	45
3.	Methodology	46
3.1	Objectives	46
3.2	Identify Input Variables.....	47
3.3	THUNDER Data Files	48
3.3.1	Air Assets.....	48
3.3.2	Integrated Air Defense System (IADS)	50
3.3.3	Unit Strength.....	53
3.3.4	Standoff Reconnaissance (SREC)	57
3.3.5	Air to Air Engagements	60
3.3.6	Intelligence, Surveillance, and Reconnaissance (ISR) Effects.....	62
3.3.7	Input Variable Summary	67
3.4	Select Output Metrics	68
3.5	Design Experiment.....	70
3.6	Parallel Processing of Experimental Design	72
3.6.1	Initialization Module.....	75
3.6.2	Decomposition Module.....	77
3.6.3	Resource Module.....	80
3.6.4	Execution Module	82
3.6.5	Monitor Module	83
3.6.6	Job Process Module.....	83
3.7	Response Surface Methodology	85
3.7.1	Effects Screening	89
3.7.2	Resolution V Design.....	89
3.7.3	Metamodel Validation.....	90
4.	Results.....	92
4.1	RSM Objectives.....	92
4.1.1	Overview.....	92
4.2	OBJECTIVE 1 – Plackett-Burman Screening Results.....	93
4.2.1	MOE Normal Probability Plots	94

4.2.2	MOE #1: Days Needed to Achieve Air Superiority (AIRSUP)	96
4.2.3	MOE #2: Days Needed to Halt FLOT (FLOT)	96
4.2.4	MOE #3: Red TAHAIT Destroyed	97
4.2.5	MOE #4: Exchange Ratio	98
4.3	Fractional Factorial Results	98
4.3.1	MOE #1: Days Needed to Achieve Air Superiority	100
4.3.2	MOE #2: Days Needed to Halt FLOT (FLOT)	102
4.3.3	MOE #3: Red TAHAIT Destroyed	103
4.3.4	MOE #4: Exchange Ratio	105
4.4	Analysis of Results	106
4.4.1	MOE #1: Days Needed to Achieve Air Superiority	107
4.4.2	MOE #2: Days Needed to Halt FLOT (FLOT)	108
4.4.3	MOE #3: Red TAHAIT Destroyed	108
4.4.4	MOE #4: Exchange Ratio	109
4.5	Metamodel Validation	109
5.	Conclusions and Recommendations	113
5.1	Conclusions	113
5.1.1	Parallelization	113
5.1.2	C2 Sensitivity Analysis	114
5.2	Recommendations for Further Study	115
Appendix A.	THUNDER Data Files	117
Appendix B.	Campaign Objectives	121
Appendix C.	Perl Scripts	132
Appendix D.	Validation Results	180
Appendix E.	Data Normality	188
Appendix F.	Regression Results	196
Bibliography		204
Vita		208

List of Figures

Figure 1. Stylized view of the ASC MSRC.....	5
Figure 2. Strategy-to-Task Hierarchy	19
Figure 3. Response Surface Methodology Flow Diagram	25
Figure 4. Heteroscedastic “Megaphone” Residual Plot.....	37
Figure 5. Thesis Methodology.....	46
Figure 6. Input Variable Definition.....	47
Figure 7. Integration Calculation -- <i>adsector.dat</i>	52
Figure 8. Unit Posture Calculation	54
Figure 9. Unit Strength C3 Degrade Curve.....	57
Figure 10. SREC Maximum Effect Multiplier – <i>srec.dat</i>	59
Figure 11. Engagement Probabilities – <i>detect.dat</i>	61
Figure 12. Perception data -- <i>airrules.dat</i>	64
Figure 13. Select Output Metrics	68
Figure 14. Experimental Design	70
Figure 15. Experimental Design Generation	71
Figure 16. Parallel Processing	73
Figure 17. MMMP Parallelization Methodology.....	74
Figure 18. MMMP Directory Structure	78
Figure 19. Data File Preprocessing.....	80
Figure 20. Resource Module Methodology	81
Figure 21. Monitor Subroutine Options.....	83
Figure 22. Job Process Module Overview	84
Figure 23. Response Surface Methodolgy	86
Figure 24. Sensitivity Analysis Flow Diagram.....	88
Figure 25. MOE Normal Probability Plots – Five Significant Variables.....	95
Figure 26. Validation of Metamodel Throughout Design Space.....	110
Figure 27.. C3 Message Capacity -- <i>typec3.dat</i>	118
Figure 28. <i>squadron.dat</i> excerpt.....	119
Figure 29. Comm/Firing Ability State Calculation -- <i>adsector.dat</i>	120

List of Tables

Table 1. 2^{11-7}_{III} Fractional Factorial Design.....	30
Table 2. Plackett-Burman 11 Factor Design	31
Table 3. ANOVA Table	39
Table 4. ANOVA Table – Lack of Fit	42
Table 5. C2 Aircraft Design Levels.....	50
Table 6. Integration Degrade Levels.....	53
Table 7. Message Capacity Design Levels – <i>typeC3.dat</i>	55
Table 8. C3 Unit Strength Degrade Curve -- <i>grdrules.dat</i>	56
Table 9. Unit Strength C3 Degrade Design Levels.....	57
Table 10. Capability Increase with Constant ACS Multipliers	61
Table 11. ACS Multiplier Design Levels.....	62
Table 12. Perception Degrade Design Levels.....	66
Table 13. Design Levels for Mean Error Quantities	67
Table 14. Input Variable Values	68
Table 15. Validation Experiment Variable Levels	90
Table 16. Variable Identification	93
Table 17. Coded Design Matrix and Average MOE Results for Plackett-Burman Screening Experiment.....	94
Table 18. Air Superiority Plackett-Burman Screening Results	96
Table 19. FLOT Plackett-Burman Screening Results	97
Table 20. TAHAIT* Plackett-Burman Screening Results	97
Table 21. Exchange Ratio Plackett-Burman Screening Results	98
Table 22. Coded Design Matrix and Average MOE Results for Resolution V Fractional Factorial Experiment.....	99
Table 23. Air Superiority Metamodel Parameter Estimates.....	100
Table 24. Air Superiority Metamodel Summary of Fit	101
Table 25. Air Superiority Metamodel Lack of Fit	101
Table 26. FLOT Metamodel Parameter Estimates	102
Table 27. FLOT Metamodel Summary of Fit.....	103
Table 28. FLOT Metamodel Lack of Fit.....	103
Table 29. Red TAHAIT* Metamodel Parameter Estimates.....	104
Table 30. Red TAHAIT Metamodel Summary of Fit.....	104
Table 31. Red TAHAIT Metamodel Lack of Fit.....	105
Table 32. Exchange Ratio Metamodel Parameter Estimates.....	105
Table 33. Exchange Ratio Metamodel Summary of Fit	106
Table 34. Exchange Ratio Metamodel Lack of Fit	106
Table 35. MOE Response Surface Analysis.....	107
Table 36. Plackett-Burman Screening Data vs. Metamodel Comparison	111
Table 37. Plackett-Burman Validation Data vs. Metamodel Comparison	111

Table 38. Comparison of RMSE, MSPR and MAPE for Plackett-Burman Screening	
Experiment Data Set	112
Table 39. Comparison of RMSE, MSPR and MAPE for Plackett-Burman Validation	
Experiment Data Set	112
Table 40. SREC Multiplier – Low Level	117
Table 41. SREC Multiplier – Baseline	117
Table 42. SREC Multiplier – High Level.....	118
Table 43. Plackett-Burman Validation Design Flag Matrix	180
Table 44. Plackett-Burman Validation Variable Design Levels	185
Table 45. C2 Aircraft Design Levels.....	185
Table 46. Integration Degrade Levels.....	185
Table 47. Message Capacity Design Levels – <i>typeC3.dat</i>	186
Table 48. Unit Strength C3 Degrade Design Levels.....	186
Table 49. ACS Multiplier Design Levels.....	186
Table 50. Design Levels for Mean Error Quantities	186
Table 51. Perception Degrade Design Levels.....	187

Abstract

This research had two objectives. The first was to develop a methodology to demonstrate the parallel processing capability provided by Air Force's Aeronautical System's Command (ASC) Major Shared Resource Center (MSRC) and apply that methodology to the SIMAF Proof of Concept project. Secondly, AFSAA/SAAB requested a sensitivity analysis of THUNDER to the modeled command and control (C2) inputs.

The power of parallelization can not be overemphasized. The data collection phase of this thesis was accomplished at the MSRC using a script developed to automate the processing of an experimental design, providing the analyst with a *launch and leave* capability. On average it took 45 minutes to process a single replication of THUNDER. For this thesis we made 1,560 runs in slightly less than 3 days. To accomplish the same number of runs on a single CPU machine would have taken slightly more than 3 months.

For our sensitivity analysis we used a Plackett-Burman Resolution III screening design to identify which of 11 input variables had a statistical impact upon THUNDER. The decision to investigate only the significant variables reduced the number of input variables from 11 to 5. This reduced the number of design points necessary to obtain the same Resolution V information from 128 to 16 and eliminated the need for 3,360 THUNDER runs. A significant savings! Using response surface methodology (RSM) techniques, we were then able to generate a response surface depicting the relationships between the input parameters and the output measures.

1. Introduction

1.1 Background

With the end of the Cold War, the US military has been required to shoulder the tremendous burden of adapting to new circumstances and challenges. Two noteworthy factors have emerged creating an environment of change for our military forces: the decreased military strength of the former Soviet Block and the victory in Desert Storm. Together these factors present a "post war" climate in Congress and the populace that expects a smaller, less costly military force (SAB-TR-96-02ES, 1996: 2).

Open a newspaper and there will invariably be an article about some group clamoring to reduce the Defense budget. Certainly, the DoD should be expected to tighten its belt in this new era. Quite often though, tightening our belt simply means doing more with less. However, a better way to operate in this era of ever shrinking defense budgets is to operate smarter and more efficiently – operate cost effectively. Current systems need to be reevaluated to ascertain if they are operating at peak efficiency and if they are providing a significant contribution to the overall effectiveness of our fighting forces.

The US military typically has never been able to rely upon overwhelming numbers to defeat an enemy. Instead, we have relied on the synergistic effects of our military systems to supply the necessary leverage to ensure our adversary "comes to the table." Coordination of these military systems (our focus will be on air forces) is of paramount importance.

The most effective way of organizing air forces to ensure unity of effort and successful conduct of military operations is under a single air commander, exercising command and control through a dedicated command and control system.

Air Combat Command Instruction 13-150

Command and control (C2) systems are in a state of rapid technological change. Information processing capabilities are growing at an exponential rate. The successful integration of these new capabilities will allow the US the ability to collect, control, exploit and defend information while denying the adversary the same. The ability to understand what is occurring in the battlespace has made the Air Force aware of C2-imposed limitations on combat effectiveness. As a consequence, the true potential of aerospace power has not been completely realized (SAB-TR-96-02ES, 1996: 3).

This, of course is a completely unacceptable condition. To remedy this, in the spring of 1996, the Air Force Scientific Advisory Board (SAB) was chartered by the Air Force Chief of Staff, to develop a Vision for Aerospace C2 for the 21st Century. The Vision for Aerospace C2 as defined by the SAB is:

Global command and control of aerospace forces throughout the spectrum of military operations by exploiting information to know, predict, and dominate the battlespace.

(SAB-TR-96-02ES, 1996: 6)

In developing this Vision, the SAB recognized the fundamental truth that the act of leading and directing the resources assigned to a military commander is essential to military operations (SAB-TR-96-02ES). Historically, the beneficial impact of effective C2

has been proven time and time again. Most recently, in the Gulf War we saw that one of our major military thrusts was the disruption of Iraq's C2 network. The result of this emphasis seemed to be a resounding success. However, caution is urged at this point. What proved effective against Iraq may not prove effective against another adversary.

Desert Storm could be considered as only one data point in a larger study investigating the effectiveness of C2 systems to influence the outcome of a war. Before making any unfounded decisions concerning the utility of C2 systems more data is needed. Short of declaring war, the use of large-scale computer combat simulations is the next best method of collecting data points for this type of study.

Currently, THUNDER, reputed to be the USAF's premier analytic campaign level simulation model, is the best available campaign model to conduct detailed analysis on the outcome of a war between two nation-state sized adversaries in a single theater. Unfortunately, identifying the causal relationships between the C2 modeled input parameters and output measures is definitely not intuitively obvious from THUNDER's output measures.

This analysis requires an analyst to hypothesize a solution, perform model runs, update the hypothesis, perform more model runs, and so on until some termination criteria is satisfied. Based upon the stochastic nature of THUNDER, a sufficiently large number of replications for each level of input factors must be performed in order to make any meaningful statistical comparisons regarding the nature of the output. With each replication requiring a nontrivial amount of time to accomplish, the amount of time involved with taking just one iterative step towards the solution is considerable. A better

approach is to develop a designed experiment, which would allow the exploration of these relationships in the most efficient manner possible. Also, the execution of these multiple replications simultaneously in a parallel environment will significantly reduce the amount of time required to achieve a statistically supportable conclusion. With the grunt work accomplished by a machine, the analyst can spend more time doing what analysts do best, analyzing. The result is a better product provided in a more responsive manner to the decision-maker.

1.2 Simulation and Analysis Facility (SIMAF)

The Air Force's Aeronautical System Command (ASC) is currently developing a capability to link models, simulations, hardware-in-the-loop, operator-in-the-loop, and system-in-the-loop resources to create a robust virtual environment to support the assessment of alternative systems in the acquisition process. ASC's Simulation & Analysis Facility (SIMAF) will use the Distributed Interactive Simulation (DIS) protocol to link extensive resources indigenous to Wright Patterson Air Force Base's Wright Labs. Additionally, DIS will facilitate linkage to assets remote from the base, providing a scaleable virtual environment capability. The SIMAF will leverage off the extensive analytical talent organic to ASC for planning and post processing. The facility will function both as a virtual integrator of models, simulations, and hardware via communications and networking nodes and as a physical gateway for ASC modeling, simulation, and analysis to the synthetic battlespace.

AFIT/ENS is supporting a demonstration of the SIMAF capability in March 1998 through scenario development, experimental design, and battleroom visualizations. The

fighter aircraft. Virtual engagement level model output from the SIMAF demonstration will be used to calibrate the THUNDER campaign. Subsequently, campaign level model analysis will provide effectiveness data for an analysis of alternatives (AoA) on the notional aircraft. Specifically, this thesis will demonstrate the connectivity of the SIMAF facility to the parallel operating environment provided by ASC MSRC.

1.3 Major Shared Resource Center (MSRC)

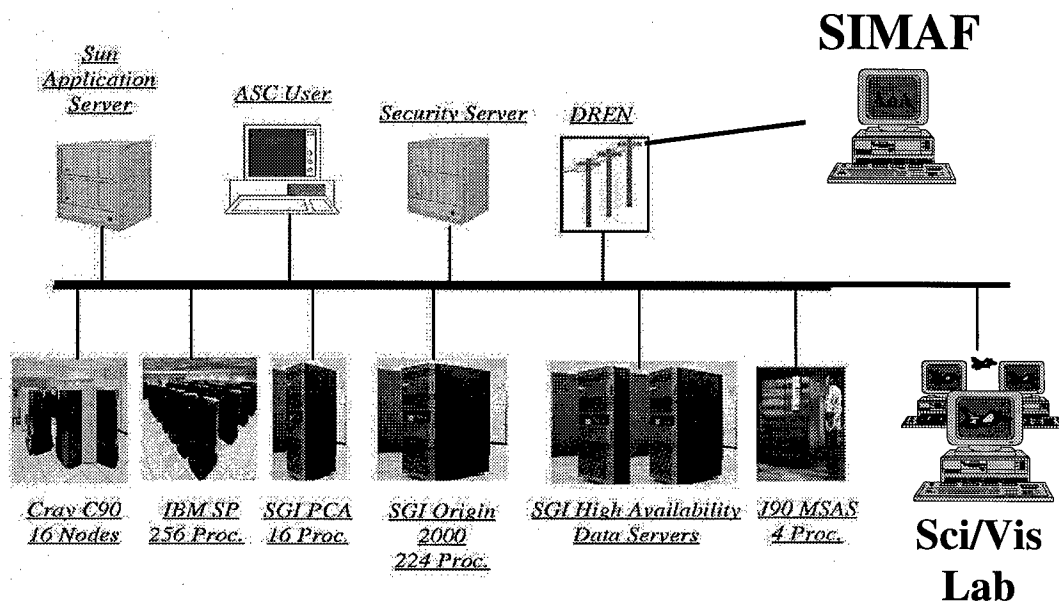


Figure 1. Stylized view of the ASC MSRC

The ASC's Major Shared Resource Center (MSRC) at Wright Patterson AFB is primarily a computational science research facility for advanced scientific high performance computing research and scientific visualization. The ASC MSRC is a part of the DoD High Performance Computing (HPC) Community and its mission is to

state-of-the art HPC capabilities to all Defense Research, Development, Test and Evaluation users (www.asc.hpc.mil).

1.4 Problem Statement

Many current campaign models use nearly perfect intelligence in making force-allocation decisions. This makes it very difficult to evaluate new systems that attempt to improve the quality or timeliness of information.

RAND TLC 1996

As Sun Tzu observed, "Know the enemy as you know yourself and in one hundred battles you will not be in peril." That C2 systems can act as a force multiplier is an accepted fact among military professionals. However, as pointed out by RAND many of the current campaign models do not model C2 adequately. Perfect intelligence is only "perfect" if you are not interested in truly capturing the impact of C2 on the outcome of a campaign. THUNDER 6.4 has evolved to the point where it models C2 capabilities reasonably well¹. However, to date no formal sensitivity analysis has ever been accomplished to determine which of the different C2 components currently modeled by THUNDER actually have the most significant impact upon the outcome of the campaign.

In response to this omission, AFSAA/SAAB (Battle Management/C2) requested an effort be undertaken to investigate the sensitivity of THUNDER to the available C2 inputs. Specifically, SAAB requested a thesis effort to generate a set of C2 response surfaces, where the input variables would be the various C2 related parameters in Thunder and the outputs would be the principal theater-level measures such as forward line of the

¹ THUNDER 6.5 has a more robust ISR modeling capability, but was not available for use until this thesis had passed the point of no return.

troops (FLOT) movement, attrition, days to halt, etc. The response surfaces would be used to form meta-models for quick reaction questions, and provide better insight into Thunder C2 processes, and C2 studies.

Contrary to the practice of using perfect intelligence, very little analysis of complex stochastic models is done *perfectly*. Typically the analysis method employed is an iterative one involving making an educated guess, performing model runs and then analyzing the model's output. Unfortunately, this method misses counter-intuitive results that are sometimes hiding just below the surface. Instead, a designed experiment should be used to reveal the maximum amount of information about a model with the minimum investment of time and resources. The problem though, is that it takes time to set up and perform the experiment. However, if this experimental design is processed in a parallel environment this time delay caused by data gathering can be significantly reduced. The ASC MSRC and their network of computers provides the unique opportunity to show the benefits of parallel processing an experimental design in a way that has not been attempted before.

1.5 Scope

This thesis has two specific objectives. First, we will develop a methodology to demonstrate the parallel processing capability provided by the MSRC and application of said methodology to the SIMAF Proof of Concept project. Second we will address the tasking of AFSAA/SAAB and provide a sensitivity analysis of THUNDER to the modeled C2 inputs. Our aim was to direct the parallelization efforts towards developing a methodology to exploit the tremendous capabilities of the MSRC in support of the SIMAF

project. The sensitivity analysis study requested by AFSAA/SAAB provides a rich environment within which to develop the parallelization methodology.

1.6 Limitations and Assumptions

The limitations and assumptions made for the timely completion of this thesis effort are factors that limit or influence the alternatives considered. Since they form the basis for the sensitivity analyses performed the limitations and assumptions should be carefully defined and stated explicitly.

1.6.1 Unclassified Data

Perhaps the biggest limitation imposed upon this thesis effort is the use of unclassified data. THUNDER is a data driven model in that the data necessary to run the model (e.g. scenario inputs, weapons systems parameters, etc.) is contained in data files which are separate from the actual computer model. As such any scenario can be simulated for which there is data. Currently, there are 80 plus data files required to develop an operational scenario. The database used for this thesis is based upon a Southwest Asia (SWA) scenario developed by HQ ASC/XR. In order to keep this thesis at the UNCLASSIFIED level ASC/XR was tasked to develop this SWA database using notional values obtained from unclassified sources².

² The Naval AOB was obtained from "Conduct of Persian Gulf War" Final Report to Congress (Unclassified) p. 110. Data for the land based fixed wing US AOB was culled from the same report (p. 106). Red forces AOB was extracted from "Storm Over Iraq" by Hallion.

While the numbers are as realistic as can be obtained from these sources, the focus of this thesis must remain on the methodology and not on the quantitative results obtained.

1.6.2 Scenario

The scenario selection was dictated by the use of THUNDER. THUNDER models major regional contingencies (MRC) and the scenario selected was a Southwest Asia scenario similar to the Gulf War. The analysis considered the impact of C2 on a worst case scenario and did not evaluate the impact of C2 on lesser contingencies.

1.6.3 Scenario Data Files

THUNDER currently requires in excess of 80 data files to generate the simulation scenario. With each data file containing numerous input variables the sensitivity analysis requested by SAAB can not feasibly take into account each of these variables. Instead all variables not related to C2 are held at fixed levels. Chapter 3 highlights the input files and variables examined by this thesis.

1.7 Thesis Overview

Chapter 2 is a review of all pertinent literature required for the development of this thesis. Chapter 3 provides an outline of the specific methodology that will be used throughout this thesis. Chapter 4 presents the results of the sensitivity analysis. Finally, Chapter 5 provides conclusions and recommendations. The various appendices contain support material necessary for a more complete understanding of this thesis process.

2. Literature Review

2.1 Previous THUNDER Studies

A simulation model as complex as THUNDER provides an excellent testbed for academic study in the field of OR. In the last four years at AFIT there have been four thesis efforts undertaken to probe, analyze, or otherwise explore THUNDER and its ability to model theater-level warfare.

Capt. Timothy Webb performed some pioneering work with THUNDER analyzing the overall model variability, output measure interrelationships, and sensitivity to input parameters. Webb concluded that all of THUNDER's output measures required a minimum of five replications, several output measures required up to 20 replications and two required in excess of 30 replications to be within 10% of the mean value (Webb, 1994: 5-1).

Capt. Steven Forsythe demonstrated that response surface methodologies (RSM) techniques could be used to optimize the air apportionment in a THUNDER scenario (Forsythe, 1994: 5-1). Forsythe used a screening design to model the relationship between the air apportionment and three output measures. His efforts provided a methodology for the development of metamodels, which allowed real-time answers to what-if questions posed by decision makers.

Capt. Ryan Farmer extended the work of Webb and Forsythe by using RSM techniques and multivariate analysis to build metamodels of THUNDER. He concluded that a Resolution V experimental design was satisfactory in producing a reasonably good

metamodel (Farmer, 1996: 6-3). Farmer's effort incorporated a rather large design space with each factor's levels perturbed from baseline values plus or minus 40%. This large design space did not affect the validity of his results. This research effort will create a design space defined by the baseline values plus or minus 25%.

Finally, Major James Grier provided a methodology for linking procurement dollars to an alternative force structures' combat capability using RSM techniques (Grier, 1997). Grier used factor analysis to reduce the dimensionality of his 34 output measures down to six factors, which corresponded to notional theater CINC campaign objectives. RSM techniques were then applied to the factors to produce a metamodel to serve as a "quick turn" tool designed to capture the cost and capabilities of alternative force structures.

2.2 Modeling and Simulation (M&S)

As we downsize our forces and face new evolving threats to our nation's security, the well-worn phrase "do more with less" will become a way of life for us. Resourcefulness and imagination, key ingredients of successful military operations, will play greater roles in how we go about our business. Competition for reduced resources will force all of us to make hard decisions on how to spend each and every dollar.

John M. Shalikashvili Chairman of the Joint Chiefs of Staff

Investing precious budgetary dollars in the development of new C2 systems comes at the expense of other weapons systems. Without proper analysis and justification, promising new C2 systems may well "stay on the bench." War provides the ultimate "test-bench" for any new system. Although still in development during the Gulf War, two JSTARS (USAF-Grumman Joint Surveillance Target Attack Radar System) test aircraft flew 49 combat sorties (USAF Fact Sheet).

Evaluating the effectiveness of new weapons systems to influence the outcome of a campaign requires actual operating conditions. Lacking warfare though, simulation is our next best avenue for investigating the interactions of said weapons systems, force allocations, and strategy – “Anything short of war is simulation.”

Simulation is being increasingly used to influence force structure and defense strategy decisions.³ Therefore, any campaign analysis must carefully consider all relevant systems and the interactions of those systems.

2.3 C2 Modeling

It is absolutely necessary to study new systems in the context of the environment in which they will participate. We must measure the inter-dependencies that are becoming a bigger and bigger part of a system's contribution to the overall campaign (Smith, 1992: 13)

With respect to C2 systems, the environment within which we are most concerned about their operability and interoperability is warfare. Short of warfare, the best we can do is to test our C2 systems during operational exercises. While certainly providing a rigorous test of these systems, exercises fall short of the strenuous demands induced by warfare. Since warfare is unforgiving, it is not the best environment for exploring the operability and interoperability of C2 systems. It would be better to perform this analysis so that making mistakes and learning lessons only results in time lost and not lives lost. This is the bread and butter of simulation. With a properly designed simulation

³ The U.S. Joint Staff uses the TACWAR campaign model for evaluating for structures for major regional contingencies and the Air Force uses THUNDER to evaluate aircraft, weapons, and operational concepts in joint theater campaigns. (RAND TLC 1996)

experiment, the inter-dependencies of C2 systems can be thoroughly analyzed for significance.

2.4 Parallelization

Whenever there is a bottleneck at some point in a system, consideration should be given to duplicating the bottleneck process thus altering the configuration of the system so several of the processes operate in parallel. Such was the case of the data collection phase of this thesis. This phase required hundreds of replications of THUNDER. While manual execution of these runs was certainly possible, it would have been prohibitively time consuming. The potential gains from automating and parallelizing this process would be tremendous.

Numerous papers, research articles, and books have been written on the parallelization of simulation models each showing the gains available from parallelization. The emphasis though, in the majority of this work has been the decomposition of a process and then the parallelization of the decomposed subprocesses. Skordos developed a methodology for an effective approach of simulating fluid dynamics on a cluster of non-dedicated workstations (Skordos, 1994). While this approach achieved 80% parallel efficiency (speedup/processor) using 20 HP-Apollo workstations in a cluster where there were 25 non-dedicated workstations (Skordos, 1994: 4), what was applicable to this thesis was the parallelization methodology. Their distributed system exploited the common file system of the workstations and was organized into the following four modules (Skordos, 1994: 5):

1. The initialization program produces the initial state of the problem to be solved as if there was only one workstation.
2. The decomposition program decomposes the initial state into subregions, generates local states for each subregion and saves them in separate files, called "dump files." These files contain all the information that is needed by a workstation to participate in a distributed system.
3. The job-submit program finds free workstations in the cluster and begins a parallel subprocess on each workstation.
4. The monitoring program checks every few minutes whether the parallel processes are progressing correctly

Even though the THUNDER simulation model can not be decomposed and each component parsed out, individual runs of THUNDER can be launched on different machines effectively achieving parallelization.

There have been at least two previous efforts made at accomplishing the task of automating the execution of THUNDER runs on multiple machines. The RAND Corporation performed the earliest work in this area with the development of TRCON (THUNDER Controller). Walt Hobbs, Senior Scientist at the RAND Corporation, developed TRCON to perform exploratory modeling and other research.

Specifically, the RAND script provides the analyst the capability to automatically modify four THUNDER data files and then perform runs of THUNDER with the modified data files on a network of computers (Hobbs, 1995). The script provides a good methodology for implementing the initialization module mentioned by Skordos. The greatest limitation of this THUNDER controller however, is that it does not allow for modification of any other of THUNDER's data files. This limitation can be critical if the experimental conditions require modifying different data files. The data file modification subroutines written by Hobbs elegantly modify their designated data files, but at

considerable programming expense. Extension of Hobb's modification subroutines to encompass a more robust suite of data files, while possible, would have required excessive effort, and was not necessary to achieve the objective of this thesis.

In the development and demonstration of a genetic algorithm and neural network (GA/NN) toolkit for campaign modeling, System Simulation Solutions, Inc. (S³I) has extended the efforts of Hobbs. S³I was awarded a small business incentive research (SBIR) to streamline the process of parameter development and determination of input/output relationships within THUNDER (Dorman et al., 1997). Clark Dorman, Senior Computer Programmer at S³I, has followed the basic methodology outlined by Hobbs to launch runs of THUNDER on multiple machines, but has integrated a GA/NN process to optimize the allocation of aircraft. As was the case with Hobbs, the approach taken by Dorman is extremely efficient if it is desired to optimize the allocation of aircraft.

Both of these scripts are written in Perl, which is short for **P**ractical **E**xtraction and **R**eport **L**anguage. Perl is a scripting language. It packages a series of shell commands into a convenient text file. The beauty of Perl scripts though, is they are often more readable than scripts in other languages. Anyone forced to decipher another's script written with the awk, or csh scriptings languages can appreciate the joy of working with a Perl script. What's more, Perl is free. A quick surf to <http://www.perl.com> and Perl can be downloaded for UNIX, NT, 95 and other platforms.

As the objective of this thesis was a sensitivity analysis of THUNDER to C2 inputs, neither of the Hobbs or Dorman scripts completely met all of the thesis requirements. Chapter 3 contains the specific parallelization methodologies used in this

thesis. Essentially though, the four modules used by Skordos were adapted and expanded using the framework provided by the aforementioned scripts to provide the methodology used in the parallelization of THUNDER runs at the MSRC.

In terms of computational parallelization potential, the MSRC is a significant addition to the SIMAF team. The MSRC brings to the table virtually unlimited data storage capacity and a network of computers that would make most offices envious. By using their resources and the script developed for this thesis, the time delay normally experienced during the data gathering phase is greatly reduced, allowing more time to be spent on the more important task of actually analyzing the data.

2.5 THUNDER Overview

2.5.1 General

THUNDER is the Air Force's two-sided, campaign model, which simulates conventional air-land combat. The Air Force uses THUNDER to conduct analyses, ranging from individual weapon systems AoAs through total force structure planning. One of the model's primary measures of outcome for theater level campaigns is Forward Line of our Own Troops (FLOT) movement. Other measures include attrition, exchange ratios, numbers of targets destroyed, and weapons inventory drawdown among others.

2.5.2 Campaign Planning

The user fills the role of campaign planner for both blue and red forces. Apportionment of aircraft is by mission category (Defensive Counter Air (DCA), Offensive Counter Air (OCA), etc.) and provides priorities for different target types in

each mission area. THUNDER uses the apportionment decision along with reconnaissance data and mission effectiveness data and allocates sorties to targets. The user can change apportionment and target prioritization at defined time intervals, usually every 12 hours of simulated time.

2.5.3 Model Fidelity

Depending upon the particular study requirements, THUNDER allows the user to operate in one of three resolution modes for many of its functions: low, high, and very high. The low-resolution mode eliminates many of the details of the battle allowing the model to run relatively fast. For example, with low resolution ISR selected, both sides have perfect knowledge of the other. High resolution mode captures many of the model details not available with low resolution. Very high resolution provides the user the ability to model very intricate details of the campaign, but the cost is significantly slower run times.

2.6 Operational Effectiveness Analysis

The effectiveness analysis performed during the course of this study will quantitatively determine the extent to which a given alternative satisfies the requirements set forth by the warfighting CINC as stated in the campaign objectives (COs). The effectiveness analysis process used was adapted from the Common Missile Warning System (CMWS) Analysis of Alternatives (AoA) Study Plan and employs an application of the Strategy-to-Task Framework shown in Figure 2 (CMWS AoA 1997 p. 8).

The Strategy-to-Task Framework depicts the hierarchical structure existing between the following warfighting components (Kent 1989):

1. National Security Objectives (NSO) – define what is necessary to protect the fundamental principles, goals, and interests of the United States. Guidance for the development of NSOs is contained in *The National Security Strategy of the United States*, an annual report to Congress by the President.
2. National Military Strategy (NMS) – represents the employment policies of our military forces to achieve the NSOs.
3. Campaign Objectives (CO) – formulated by the combatant commander and define how military forces will be employed in a given region (e.g., halt invading armies).
4. Operational Objectives (OO) – the objectives to be achieved by regional military operations. (e.g., destroy/damage/disrupt lead elements of an armored advance)
5. Operational Tasks (OT) – more specific than the OOs, but not readily quantifiable. (e.g., destroy/damage advancing armored vehicles or mine/cut key attack routes)
6. Operational Capabilities (OC) – sometimes called metrics, are qualitative or quantitative measures of a system's performance. (e.g., Percent of tanks killed or Time to stop Red advancement)

2.7 Sensitivity Analysis of THUNDER to C2 Inputs



Figure 2. Strategy-to-Task Hierarchy

(CMWS AOA Study Plan 1997)

THUNDER can be naively thought of as a black box, transforming inputs into outputs. When thought of in this light, THUNDER is simply a function and if given input, will generate output. Unfortunately, it is nowhere as simple as that. Complicating the situation is the stochastic nature of THUNDER, which uses random variables to determine the outcomes of different situations. Although THUNDER employs a deterministic ground war, stochastic models are used in THUNDER to model everything from the probability a detection between opposing aircraft occurs to the deviation of a bomb from its desired impact point. Using the deterministic function for a line, $y = m * x + b$, and knowing an input value for x , an exact value for y can easily be determined. With a stochastic model like THUNDER though, setting values for input variables can result in many distinct realizations for the output measures.

THUNDER is a complex simulation model of theater warfare where there are hundreds of input variables and depending upon the analyst and analysis methodology chosen even more output measures. Consequently, a tasking as apparently benign as performing a sensitivity analysis of THUNDER to the available C2 input variables is most definitely not. The disciplined and methodological approach provided by response surface methodology was needed.

2.8 Response Surface Methodology

There is a wealth of literature available on the subject of RSM. For a more in-depth analysis of RSM techniques, the reader is referred to two excellent texts: *Response Surface Methodology* by Myers and Montgomery, and *Empirical Model-Building and Response Surfaces* by Box and Draper. What is presented below is a brief review of those aspects of RSM applicable to this thesis.

The first line of the first paragraph on page one of Myers and Montgomery's book succinctly states that RSM is a collection of statistical and mathematical techniques useful for developing, improving and optimizing processes (Myers and Montgomery, 1995: 1). According to Box and Draper, RSM seeks to relate a response, or output variable, to the levels of a number of predictors, or input variables, that affect it (Box and Draper, 1987: 1). Both of these statements are true, and provide an insight into the usefulness of RSM. But Law and Kelton provide another description, one that perhaps provides a clearer understanding of what RSM attempts to accomplish. In their text *Simulation Modeling and Analysis*, which should be reference material for all analysts, Law and Kelton describe a simulation model as a simple mechanism that turns input parameters into output

performance measures (Law and Kelton, 1987: 679). If you can accept this, they point out that a simulation is, in a sense, simply a function whose explicit form is probably unknown (Law and Kelton, 1987: 679). From this viewpoint RSM is a set of techniques helpful in the development of the formulas to approximate this unknown function. These techniques encompass (Khuri and Cornell, 1987):

1. Designing experiments that will result in adequate and reliable measurements of the response(s) of interest in a region of interest.
2. Analyzing the results of those experiments to determine a function that describes the data collected.
3. Searching for the optimal settings of the input variables that result in a desired response.

Since the objective of this thesis is to perform a sensitivity analysis of THUNDER to the modeled C2 input parameters, we are more concerned with the first two techniques listed above than with optimizing the response. Accordingly, our objective is to develop a parsimonious function that adequately maps the response surface generated by our experimental design over a particular region of interest.

The function serving proxy for the response surface for each of the different output measures (y) under consideration in this thesis are all influenced by the levels of the C2 variables (x_i) and takes the form:

$$y = f(x_1, x_2, \dots, x_k) + \varepsilon \quad (1)$$

where the form of the real response function f is unknown and ε represents the random error in the system or other sources of variability not accounted for in Equation 1 (Myers and Montgomery, 1995: 3).

The function used to represent a response surface is commonly referred to as a simulation metamodel. A metamodel can take on many different forms, but are typically polynomial regression models. If the response surface is being approximated over a relatively small design space or curvature is not a problem, a first-order model is quite often adequate. In general, a first-order model is

$$y_u = \beta_0 + \sum_{i=1}^k \beta_i x_{iu} + \varepsilon_u \quad (2)$$

while a second-order model takes the form

$$y_u = \beta_0 + \sum_{i=1}^k \beta_i x_{iu} + \sum_{i=1}^k \beta_{ii} x_{iu}^2 + \sum_{i < j} \beta_{ij} x_{iu} x_{ju} + \varepsilon_u \quad (3)$$

where $u = 1, \dots, n$ is the simulation replication number, y_u is the output measure for the u^{th} replication, x_{iu} is the level of the i^{th} variable on the u^{th} replication, the β 's are the model coefficients to be estimated using regression analysis, and ε_u is the unexplainable error from the simulation model (Donohue, 1995: 194). The second-order model would be used when curvature in the response surface was detected through lack-of-fit in the first order model. The actual methodology involved in RSM is discussed in subsequent sections and begins with a selection of an appropriate experimental design.

2.9 Experimental Design Selection

“Pluralites non est ponenda sine necessitate.” (Multiplicity ought not to be posited without necessity.)
The Principle of Occam's Razor

In mathematical terms, this quotation should be considered a theorem, violation of which could lead to the failure of the entire experiment. In simulation terms, *experimental design* refers to the purposeful formulation of a plan of attack for deciding before the runs

are made, which particular configurations to simulate so that the desired information is obtained with the least amount of simulating (Law and Kelton, 1991: 657). The selection of an appropriate experimental design is an area that has received considerable attention in the simulation community. Gordon, Ausink, and Berdine used experimental design strategies while investigating spacecraft control simulation (Gordon et al, 1994: 303-309). Bailey and Clark applied experimental design strategies using simulation in the management of taxi routes (Bailey and Clark, 1992: 1217-1222). Donohue provides a review of recent research into the efficient accomplishment of simulation studies using experimental design (Donohue, 1994: 200-206). Donohue's paper outlines a twelve-step methodology for an efficient and statistical approach to the design and analysis of simulation experiments which is needed to draw meaningful conclusions from the experimental data (Donohue, 1994: 200). Donohue's methodology is listed below:

1. State the problem requiring experimentation and state the objective of the study (e.g.; prediction, optimization, sensitivity analysis).
2. Choose the factors (controllable input variables).
3. Select the response variable (output variable).
4. Determine the operability region (range of values for each factor within which the system can operate).
5. Specify the region of interest (a subregion of the operability region within which you want to perform the current experiment).
6. Choose a statistical model (e.g.; ANOVA, regression, spatial correlation).
7. Select criteria for choosing an experimental design (e.g.; minimize generalized variance, minimize mean squared error).
8. Choose an appropriate experimental design class (e.g.; factorial, Latin square, central composite).

9. Select the levels of the factors for each design point (experimental run).
10. Perform the experiments and collect data.
11. Analyze and summarize the data; check for adequacy of the statistical model.
12. Draw inferences and conclusions.

The first nine steps of this methodology are arguably the most difficult.

Considerable mental gymnastics are required to balance maximal information and experimental efficiency desires. If the objective of the study is too ambitious, then step ten may prove intractable. Step ten of this methodology truly involves the *grunt work* of any study.

When a simulation model requires an hour or more to make a single replication the prospect of implementing even an efficient experimental design can quickly turn the stomach of even the heartiest analyst. It is for this reason we have relied on the parallel computing environment provided by the MSRC to process the experimental runs in parallel, letting the computer take care of the *grunt work* and leaving the analysts free to accomplish more important tasks.

The eleventh step listed is also fraught with considerable difficulty. How exactly do you check for adequacy of the statistical model? Figure 3 shows some excellent methodological guidance for this step that was provided during OPER 683 -- Response Surface Methodology. Note however, this methodology assumes an overall objective of process optimization and the stated goal of this thesis is a sensitivity analysis. This apparent conflict in purposes is resolved by modifying the analytical flow of this methodology so that it is more amenable to sensitivity analysis.

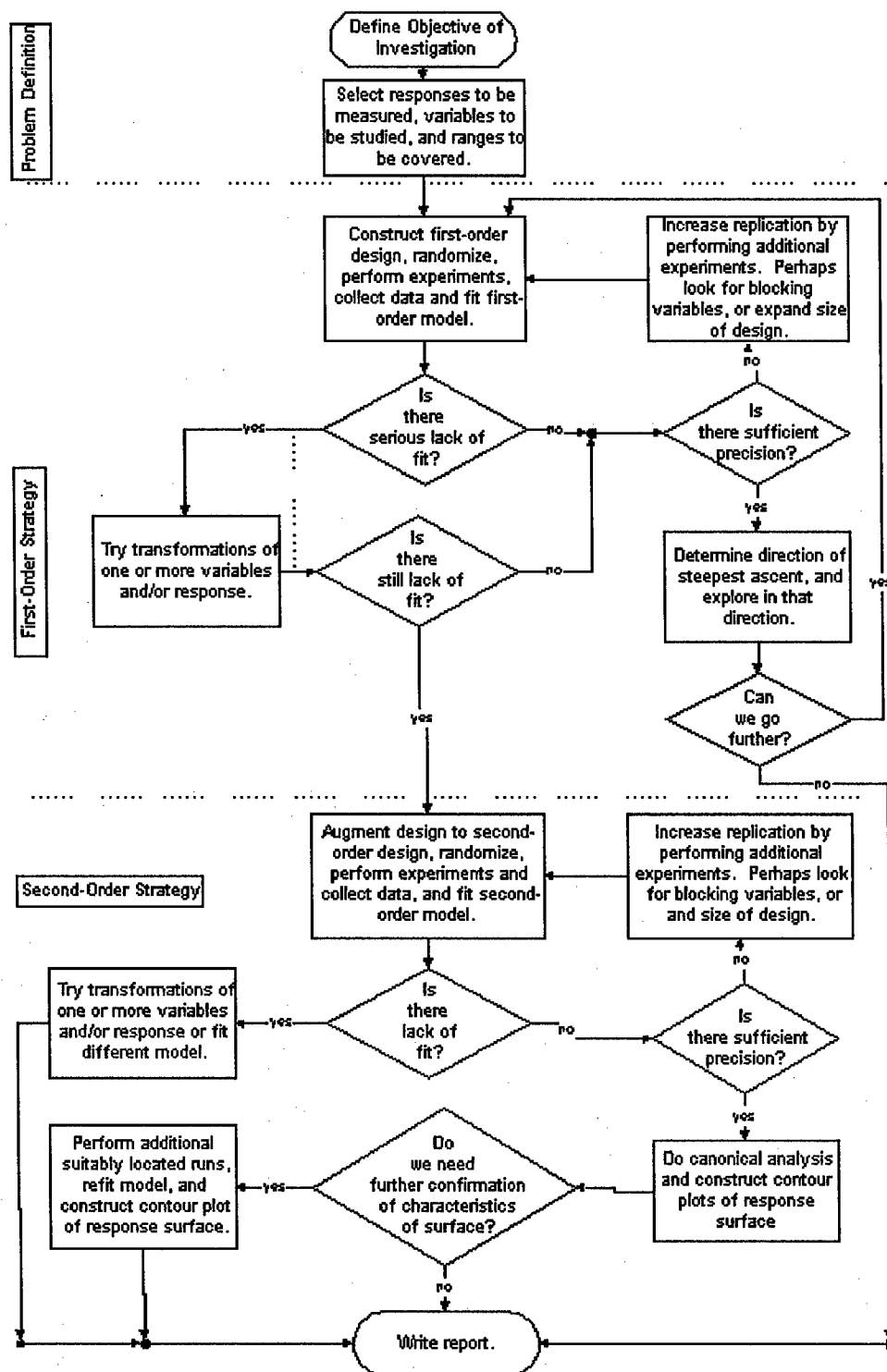


Figure 3. Response Surface Methodology Flow Diagram

The attractive aspect of Figure 3 is it allows the analyst to stop with a first-order design if it proves adequate. As Albert Einstein put it, *everything should be made as simple as possible—but no simpler*. Often times an analyst tries to violate the Principle of Occam's Razor in an attempt to garner face validity based upon the difficulty level of the analysis. Following this flowchart helps to avoid unnecessarily complicating the process.

2.9.1 Random Numbers

Donohue also identifies two strategic experimental design issues that must be considered when operating in a simulation environment (Donohue, 1994: 201). The first is to choose a method for the assignment of random number streams to design points, and the second is to decide whether or not to use an appropriate variance reduction technique (VRT). VRT issues are directly related to the efficiency of the experiment. If methods can be found to reduce the variance of a simulation's output measures, then fewer runs of the model will be required to achieve the required accuracy levels.

If the purpose of a study is sensitivity analysis, then consideration should be given to using the method of common random numbers to control the randomness in the simulation (Law and Kelton, 1991: 311). A sensitivity analysis of a simulation model is performed to determine if the simulation output changes significantly as the levels of the input parameters are changed. Law and Kelton argue that unless an attempt is made to control the randomness in a simulation, the effects of changing one aspect of the model may be confounded with other changes (e.g., different random values from some input distribution) that inadvertently occur (Law and Kelton, 1991: 311).

However, there is a danger involved here. Specifically, if CRNs are used, the distribution characteristics of the errors are changed (i.e., they are no longer independent). Instead of being able to use classical methods for generating a CI, a paired t-test is performed to compute the desired $100(1 - \alpha)\%$ CI (For a synopsis of generating a CI with correlated data see Banks et al.). The implication of this limitation is that the degrees of freedom are cut in half.

Since the degrees of freedom and t-values as well as standard error are inversely related, reducing the degrees available normally increases the half-width of a given CI. However, if CRN are successfully implemented, this limitation is overcome by a reduction in variance attained within the model. The problem though, is there is no guarantee that correlated sampling will always induce a positive correlation between comparable runs of a model (Banks et al., 1996: 483). Further, it is recommended that if synchronization of two models using CRN is not possible then independent streams of random numbers should be used (Banks et al., 1996: 483).

2.9.2 Factorial Designs

Our research indicated factorial designs are widely used in experiments involving several factors where it is necessary to investigate the joint effects of the factors on a response variable (Myers and Montgomery, 1995: 508). When each factor, or variable, has only two possible levels the experimental design is called a 2^k factorial design. Two important characteristics of 2^k factorial designs prompted us to consider their use. First, a 2^k design can be used in a screening experiment to identify the important system variables.

Second, a 2^k design can be used to fit a first-order response surface model to the data.

Both of these characteristics are in line with the objectives of our sensitivity analysis.

2.9.3 Coding

If the sensitivity analysis involves input factors having different units of measure the resultant analysis may be difficult to interpret. However, if the variables are transformed into coded variables (dimensionless with mean of zero and the same standard deviation) with values between -1 and $+1$, interpretation of the regression equation becomes straightforward. A general formula for coding variables is (Myers and Montgomery, 1995: 22):

$$x_i = \frac{\xi_i - [\max \xi_i + \min \xi_i] / 2}{[\max \xi_i - \min \xi_i] / 2} \quad (4)$$

where ξ_i is the actual value of the variable. Tests for curvature will require adding center points to the experimental design with a coded value of 0.

2.9.4 Fractional Factorial Design

Each complete replicate of a 2^k design requires 2^k runs. If the simulation is a stochastic model, then replications at each design point are required in order to reach any sort of supportable conclusions. As the number of input variables increases, the expense of performing this many experiments may become prohibitive. For this reason, fractional factorial designs are often used in place of the full-factorial design to reduce the number of experiments required.

For example, in our analysis a complete 2^{11} factorial experimental design would require 2,048 distinct design points. With 30 replications at each design point a full factorial analysis requires 61,440 runs of THUNDER. With one 30-day run of THUNDER taking approximately 30 minutes, this requires 30,720 dedicated computer processing hours. Clearly this is not an option.

Two options are readily available to reduce the number of runs required in a simulation experiment: fractional factorials and Plackett-Burman designs. Fractional factorial designs are often used in screening experiments when there are many variables, where it is assumed that most do not significantly impact the performance measure under consideration. This effect, where a performance measure is driven by a small subset of the input parameters, has been called the Sparsity-of-Effect Principle and is one of the main reasons for the success of fractional factorial designs (Myers and Montgomery, 1995: 134). The general form of a fractional factorial design is 2^{k-p} , where k refers to the number of input parameters and p addresses the degree of reduction performed in the fractional factorial experiment. Of the possible 2^k runs, this type of design requires only 2^p as many runs. For this thesis, the 2^{11-7} design shown in Table 1 would have provided sufficient information for the initial stages of investigation (clarification of this issue follows the discussion of Plackett-Burman designs).

Table 1. 2^{11-7}_{III} Fractional Factorial Design

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
Design Point 1	-1	-1	-1	-1	1	-1	-1	1	-1	1	1
Design Point 2	-1	-1	-1	1	-1	1	1	-1	1	-1	-1
Design Point 3	-1	-1	1	-1	-1	1	1	-1	-1	1	1
Design Point 4	-1	-1	1	1	1	-1	-1	1	1	-1	-1
Design Point 5	-1	1	-1	-1	-1	1	-1	1	1	-1	1
Design Point 6	-1	1	-1	1	1	-1	1	-1	-1	1	-1
Design Point 7	-1	1	1	-1	1	-1	1	-1	1	-1	1
Design Point 8	-1	1	1	1	-1	1	-1	1	-1	1	-1
Design Point 9	1	-1	-1	-1	-1	-1	1	1	1	1	-1
Design Point 10	1	-1	-1	1	1	1	-1	-1	-1	-1	1
Design Point 11	1	-1	1	-1	1	1	-1	-1	1	1	-1
Design Point 12	1	-1	1	1	-1	-1	1	1	-1	-1	1
Design Point 13	1	1	-1	-1	1	1	1	1	-1	-1	-1
Design Point 14	1	1	-1	1	-1	-1	-1	-1	1	1	1
Design Point 15	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
Design Point 16	1	1	1	1	1	1	1	1	1	1	1

2.9.5 Plackett-Burman Designs

A Plackett-Burman design is a special case of a fractional factorial design for studying up to $k = N - 1$ variables in N runs, where N is a multiple of four and not a power of two (Myers and Montgomery, 1995: 169).

Accordingly, a Plackett-Burman design requiring only 12 design points was selected to reduce this number to a more manageable level. In addition to the 12 design points, we added two center points to test for curvature. With the same 30 replications at each design point we have reduced the number of runs down to 360 – a 99% decrease.

As expected, this savings is not free. The price paid for dramatically reducing the number of runs required is a loss of information from the simulation experiment. This loss essentially reflects a reduced experimental design's potential inability to distinguish which factor or interaction among the factors has a significant impact upon the outcome of the experiment.

Table 2. Plackett-Burman 11 Factor Design

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}
Design Point 1	1	1	1	1	1	1	1	1	1	1	1
Design Point 2	-1	1	-1	1	1	1	-1	-1	-1	1	-1
Design Point 3	-1	-1	1	-1	1	1	1	-1	-1	-1	1
Design Point 4	1	-1	-1	1	-1	1	1	1	-1	-1	-1
Design Point 5	-1	1	-1	-1	1	-1	1	1	1	-1	-1
Design Point 6	-1	-1	1	-1	-1	1	-1	1	1	1	-1
Design Point 7	-1	-1	-1	1	-1	-1	1	-1	1	1	1
Design Point 8	1	-1	-1	-1	1	-1	-1	1	-1	1	1
Design Point 9	1	1	-1	-1	-1	1	-1	-1	1	-1	1
Design Point 10	1	1	1	-1	-1	-1	1	-1	-1	1	-1
Design Point 11	-1	1	1	1	-1	-1	-1	1	-1	-1	1
Design Point 12	1	-1	1	1	1	-1	-1	-1	1	-1	-1

2.9.6 Resolution

One aspect of the experimental design deserving special consideration is the selection of an appropriate experimental design. Specifically, what level of detail is required of the analysis. A 2^k full factorial design will include k main effects, $\binom{k}{2}$ two factor interactions, $\binom{k}{3}$ three-factor interactions, ..., and one k -factor interaction. If the purpose of the study requires the estimation of each of these $2^k - 1$ effects then a full factorial design is required. However, and fortunately so, this is usually not the case. For this thesis we selected 11 input parameters. A full factorial design in these factors (2048 design points) would enable us to estimate all interaction effects for all 11 of these variables. A considerable effort would be required to run these design points.

As previously mentioned though, the Sparsity-of-Effect Principle tells us that the full design would not be worth our trouble since the majority of the higher order interactions are inconsequential. The methodology advocated by Dr. Genichi Taguchi,

robust parameter design, even suggests that two factor interactions are insignificant and his designs often do not allow estimation of the interactions of the control variables (Myers et. al., 1992: 132). This rather extreme view concerning the insignificance of interaction effects has sparked considerable debate, with the answer probably somewhere in the middle.

If the estimation of higher order interactions is not necessary the full factorial design can be reduced. If this reduction is done smartly, we can still retain the maximum amount of information from the minimum number of runs. The amount of information retained is related to a design's *resolution*. An experimental design is of Resolution R if no p -factor effect is aliased with any other factor containing less than $R - p$ factors (Myers and Montgomery, 1995: 138). Resolution III, IV, and V designs are particularly useful for simulation analysis.

Resolution III: No main effect (one-factor) is aliased with any other main effect, but can be aliased with a two-factor interaction. Also, two-factor interactions can be aliased with each other. Resolution III designs are often used for screening experiments or to fit a first order design.

Resolution IV: No main effect is aliased with any other main effect, or two factor interaction, but two factor interactions are aliased with each other. A Resolution IV design should be the minimum resolution used if it is important to isolate the main effects from interaction effects.

Resolution V: no main effect or two-factor interaction is aliased with any other main effect or two-factor interaction. But two-factor interactions can be aliased with three-factor interactions. Resolution V designs provide substantial amounts of information, but requires two times the number of runs needed for a Resolution IV design (Myers and Montgomery, 1995: 138).

2.10 Regression Analysis

Regression analysis is a statistical methodology that uses the relationship between quantitative variables so that a response can be predicted (Neter et al., 1996: 3).

Regression analysis can be considered a distant cousin to factor analysis in that it essentially seeks to determine the underlying relationships between the levels of the given input variables and some output measure. Regression analysis is usually used for one of three purposes: description, control or prediction (Neter et al., 1996: 9). This thesis uses regression analysis to describe the sensitivity of THUNDER to the modeled C2 parameters.

If it is possible to mathematically describe the relationships between the input parameters and an output measure, then the output measure will vary with the input parameters in a systematic fashion. This systematic relationship is quantified by a linear combination of the input parameters and some unknown regression coefficients. A regression model of k input variables and one response variable takes the following form:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \varepsilon \quad (5)$$

The regression coefficients, β_j , represent the expected change in the response of the model with a unit change in x_j , given that all of the other input variables are held constant

(Myers and Montgomery, 1995: 17). The error term, ϵ , is a direct consequence of the stochastic nature of a model and refers to the random error within the model or our failure to completely model all aspects of the system. According to Wackerly, Mendenhall and Scheaffer, ϵ represents our inability to provide an exact model for nature (Wackerly et al., 1996: 481).

When there are multiple independent variables and dependent variables, it is substantially easier to accomplish the regression analysis using matrix notation. In matrix terms with k input variables and n response variables, Equation 5 are represented as:

$$y = X\beta + \epsilon \quad (6)$$

where:

$$y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}, \text{ Response vector} \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1k} \\ 1 & x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix}, \text{ Design Matrix}$$

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}, \text{ Coefficient vector} \quad \epsilon = \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}, \text{ Error vector}$$

The crux of the regression problem is determining the β vector. Parameter estimation methods based upon sufficient statistics, like the method of moments and the maximum likelihood methods, have been all been successfully used. However, for regression analysis the most popular method of approximating β is with the method of least squares estimation. Succinctly stated, the method of least squares attempts to fit a

line through the data such that the sum of squares of the errors, ϵ , is minimized. The result of this estimation procedure is an estimate for β given by:

$$b = (X'X)^{-1}X'y \quad (7)$$

The experimental design discussed earlier provides the design matrix, X .

THUNDER output based on the design matrix inputs provides a vector of observed responses, y . The next step applies Equation 7 to provide us our estimates for β . The general analysis strategy involves comparing the vector of fitted responses $\hat{y}$, to the vector of actual responses, y . The next five sections will describe some tests we can apply to our regression analysis to provide some measure the *goodness of fit* for our postulated regression equation.

2.10.1 Coefficient of Multiple Determination

A straightforward test of a regression model is the coefficient of multiple determination (R^2). Straightforward in that most statistical analysis packages directly provide this measure of goodness with no prompting required by the analyst. R^2 measures the proportionate reduction of total variation in the observed responses with the use of the set of X independent variables and can be expressed as (Neter et al., 1996: 230):

$$R^2 = \frac{SSR}{SSTO} = 1 - \frac{SSE}{SSTO} \quad (8)$$

where SSR is the sum of squares of regression ($\sum (\hat{y}_i - \bar{y})^2$), SSE is the sum of squares of error ($\sum (y_i - \hat{y}_i)^2$), and $SSTO$ is the total sum of squares ($\sum (y - \bar{y})^2$).

R^2 is simply the ratio of the amount of variability in the fitted response values, $\hat{y}$, to the variance in the observed values y . Accordingly, a model that explains 100% of the variability has $R^2 = 1.0$. As the ability of the model to fit the data deteriorates, R^2 will decrease.

As a general rule higher values of R^2 indicate a better fit to the data, but not always. Increasing the number of independent variables will never decrease the value of R^2 . Therefore, if we add enough variables to the model we can force a *good* fit while not being able to accurately predict the estimated response. The adjusted coefficient of multiple determination, R^2_{adj} , compensates for this effect by taking into account the number of parameters or degrees of freedom used to estimate the model. R^2_{adj} can be expressed as

$$R^2_{adj} = 1 - \frac{\left[\frac{SSE}{n-p} \right]}{\left[\frac{SSTO}{n-1} \right]} = 1 - \frac{(n-1)}{(n-p)} \frac{SSE}{SSTO} \quad (9)$$

where n is the number of observations and p are the number of parameters used in the regression equation. If R^2_{adj} is used as an adequacy measure and unnecessary terms are included in the regression equation, the value may actually become smaller (Neter et al., 1996: 231).

2.10.2 Residual Analysis

Another useful method for assessing our model is residual analysis. The residual e_i , is the difference between the observed value Y_i and the fitted value $\hat{Y}_i$. The regression

model in Equation 5 makes some assumptions concerning the nature of the unknown true error, ϵ , which must be met for this analysis to remain valid. It is assumed the error terms are independent and identically distributed (*iid*) normal random variables with a mean of 0 and constant variance. If the postulated regression model is appropriate for the data then the residual terms, e_i , will reflect these assumed properties.

A check for the independence of the error terms is made by visually inspecting a plot of the predicted value of a response variable versus the residual. If in fact the error terms are independent, there should be no discernible pattern and the plot should fluctuate in a more or less random pattern around the baseline 0 (Neter et al., 1996: 105). The often-used metaphor is the plot should appear to have been made by a shotgun blast. Analyzing the shape of the residual plot subjectively allows a check for constant variance. Any non-constant variance, or heteroscedasticity, in the data is apparent in the residual plot if the points take the shape of the familiar megaphone Figure 4.

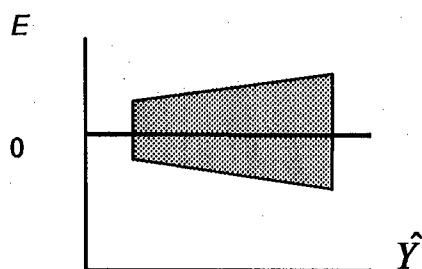


Figure 4. Heteroscedastic “Megaphone” Residual Plot

2.10.3 F test for Regression Relation

As discussed previously, a regression model of k input variables and one response variable has regression coefficients β and takes the form given in Equation 7. We have also shown the methodology for approximating β using the method of least squares estimation. The problem with this method is that a line is fit through the data such that the sum of squares of the errors, ϵ , is minimized -- every time. The question that really needs to be asked is whether or not the regression coefficients describe an actual relationship between the response variable and the independent variables.

This question takes the form of a hypothesis test. In a hypothesis test we consider two possible alternatives. The alternative hypothesis (H_a), typically represents what we would like to conclude from the investigation and the null hypothesis (H_o) supports the opposing viewpoint. This testing, as does any research into the dynamics of a stochastic system, brings along with it a certain level of uncertainty. This uncertainty manifests itself in our propensity to make two types of errors when making assertions about our results. A type *I* error occurs if the null hypothesis is rejected when it is true. The probability of making a type *I* error is denoted by α , where α is called the level of the test. A type *II* error occurs if we fail to reject our null hypothesis when in fact the alternative is true. The probability of a type *II* error is denoted by β , where $(1 - \beta)$ is referred to as the power of the test (Wackerly, 1996: 413).

A decision between the alternatives is made based upon a test statistic, which is formed from the actual data. If the test statistic falls within a specified rejection region, we will reject the null hypothesis in favor of the alternative. The likelihood of rejecting the

null hypothesis is directly related to the size of the rejection region and changes based upon the levels of α and β . Due to the risks involved, if α and β are not already predetermined by governing directives, the analyst is well advised to involve the decision maker in the setting of these values.

To assess the significance of our regression relation between a response y and a set of input variables ($x_1, x_2, \dots, x_i$) using the F-test we form the hypotheses:

$$H_o: \beta_1 = \beta_2 = \dots = \beta_i = 0$$

$$H_a: \text{Not all } \beta_i \text{ equal zero}$$

The test statistic for this test is:

$$F^* = \frac{MSR}{MSE} \quad (10)$$

An analysis of variance (ANOVA) table provides a useful bookkeeping tool for the calculation of F^* (See Table 3, where n is the number of observations and p is the number of estimated parameters in the regression equation) (Neter, 1996: 229).

Table 3. ANOVA Table

Source of Variance	Sum of Squares	Degrees of Freedom	Mean Square	F^*
Regression	SSR	$p - 1$	$MSR = \frac{SSR}{p - 1}$	$\frac{MSR}{MSE}$
Error	SSE	$n - p$	$MSE = \frac{SSE}{n - p}$	
Total	$SSTO$	$n - 1$		

The decision rule to control the risk of type I error at α is (Neter, 1996: 230):

$$\begin{aligned} &\text{If } F^* \leq F(1 - \alpha; p - 1, n - p), \text{ conclude } H_o \\ &\text{If } F^* \geq F(1 - \alpha; p - 1, n - p), \text{ conclude } H_a \end{aligned} \quad (11)$$

Our goal of course, in performing a sensitivity analysis is to derive a useful equation relating the response(s) to the input variables. The F-test provides a straightforward method for determining if there is a regression relation, but if we conclude H_a all we really know is that not all of the regression coefficients are equal to zero. Given that we concluded H_a , we can use individual t-tests to ascertain which of the coefficients are significant.

2.10.4 t- test

If our analysis resulted in a regression equation of the form:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \varepsilon, \quad k = 1, 2, \dots, p$$

we next investigate the relative contribution of each regression coefficient. The purpose of the investigation is to determine if in fact the variable is significant, or just random noise that has no bearing on the response of interest. We know that adding a variable to the regression model will always increase the sum of squares for regression and decrease the error sum of squares (Myers and Montgomery, 1995: 31). However, and more importantly, adding a variable to the equation that is not significant can increase the mean square error, thereby decreasing the usefulness of the model (Myers and Montgomery, 1995: 31).

Assuming we have used the method of least squares to fit our model and the errors are *iid* and distributed $N(0, \sigma^2)$, we can test each β_k for significance with the following hypothesis test (Neter, 1996: 232):

$$H_o: \beta_k = 0 \tag{12}$$

$$H_a: \beta_k \neq 0$$

The test statistic for this test is:

$$t^* = \frac{b_k}{s\{b_k\}} \quad (13)$$

where $s\{b_k\}$ is the standard error of the estimated regression coefficient b_k .

The decision rule to control the risk of type I error at α is:

$$\text{If } |t^*| \leq t(1 - \alpha/2; n - p), \text{ conclude } H_0 \quad (14)$$

Otherwise conclude H_a

2.10.5 Lack of Fit test

After analyzing our model for adequacy using the preceding four tests we are ready to progress to the first decision diamond of Figure 3. We have constructed our experimental design, performed experiments, collected data, fit a first-order model and we are now at the point where we check to see if a linear regression function provides a good fit for the data.

A formal Lack of Fit test requires multiple observations of the response variable at the same setting for all of the independent variables. This allows us to derive a model-independent estimate of σ^2 . Specifically, the test involves partitioning the error sum of squares from Table 3 into two components:

$$SSE = SSLOF + SSPE \quad (15)$$

where $SSLOF$ is the sum of squares due to lack of fit and $SSPE$ is the sum of squares due to pure error and are defined as:

$$SSPE = \sum_{i=1}^m \sum_{j=1}^{n_i} (y_{ij} - \bar{y}_i)^2 \quad (16)$$

$$SSLOF = \sum_{i=1}^m n_i (\bar{y}_i - \hat{y}_i)^2 \quad (17)$$

From Equation 17 it is relatively easy to see how the replicated observations enable us to test for fit. If the fitted values $\hat{y}_i$ are close to the average responses $\bar{y}_i$, then *SSLOF* is small and there is reason to believe the linear model is adequate.

Once again, an ANOVA table provides a convenient bookkeeping tool for tracking the results of our Lack of Fit test.

Table 4. ANOVA Table – Lack of Fit

Source of Variance	Sum of Squares	Degrees of Freedom	Mean Square	<i>F</i> *
Regression	<i>SSR</i>	<i>p</i> - 1	$MSR = \frac{SSR}{p-1}$	
Error	<i>SSE</i>	<i>n</i> - <i>p</i>	$MSE = \frac{SSE}{n-p}$	
	<i>SSLOF</i>	<i>c</i> - <i>p</i>	$MSLOF = \frac{SSLOF}{c-p}$	$\frac{MSLOF}{MSPE}$
	<i>SSPE</i>	<i>n</i> - <i>c</i>	$MSPE = \frac{SSPE}{n-c}$	
Total	<i>SSTO</i>	<i>n</i> - 1		

The test statistic for this test is (Neter, 1996: 240)

$$F^* = \frac{MSLF}{MSPE} \quad (18)$$

For this test our objective is reversed. Now when F^* is compared to the critical value we would like to prove H_o , rather than H_a , and thereby show that the linear regression equation is appropriate. Therefore, the decision rule to control the risk of type I error at α is:

$$\begin{aligned} \text{If } F^* \leq F(1 - \alpha; c - p, n - c), \text{ conclude } H_o \\ \text{If } F^* > F(1 - \alpha; c - p, n - c), \text{ conclude } H_a \end{aligned} \quad (19)$$

This decision rule is not an absolute measure of adequacy of fit. Studies have shown that the observed value for F^* must be at least four or five times the critical value if the regression model is to be useful as a predictor (Myers and Montgomery, 1995: 53).

2.10.6 Metamodel Validation

The validation of a regression equation is the last step in a sensitivity analysis using RSM. Regression equations, by the method of their genesis are not unique. Two different analysts with the same pile of data could construct two entirely different houses of sticks, but the real test of their efforts is when the validation wolf comes knocking on their door. If the proposed model stands up against testing it can be considered validated and have credibility when it is used in place of the model to form conclusions.

Typically, one of three different methods are used to validate a regression model:

1. Collection of new data to check the model and its predictive ability.
2. Comparison of results with theoretical expectations, earlier empirical results, and simulation results.

3. Use of a holdout sample to check the model and its predictive ability (Neter et al., 1996: 434).

The reason for developing the model in the first place often provides the means to validate the model. If the purpose behind the study is an analysis, sensitivity or otherwise of a simulation model, then the usual means of validating the metamodel is to generate additional data within the design space using the same model. An important point to remember during the validation phase is that the data used for validation of a metamodel must be independent of those used to build the metamodel. If the data is dependent in some way, then the only thing that is really getting verified is that the simulation model behaves consistently under similar conditions.

A measure that checks the predictive ability of a proposed regression equation is the mean squared prediction error (MSPR) (Neter et al., 1996: 435):

$$MSPR = \frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{n} \quad (20)$$

where: Y_i is the value of the response variable in the i^{th} validation case, $\hat{Y}_i$ is the predicted value for the i^{th} validation case based on the model building data set, and n is the number of cases in the validation set. According to Neter et al., if the MSPR is fairly close to the model's MSE then the regression equation is not seriously biased and gives an appropriate indication of the predictive ability of the model.

Another predictive measure for validating the metamodel is the Mean Absolute Percentage Error, (Farmer, 1996: 5-2):

$$MAPE = \frac{100}{n} * \sum_{i=1}^n \frac{|Y_i - \hat{Y}_i|}{|Y_i|} \quad (21)$$

where Y_i is the observation from the i th validation run, $\hat{Y}_i$ is the predicted value for the i th validation run using the metamodel based upon the model building data set, and n is the number of runs.

2.11 Output Measures

The Air and Space Power Validation Group (ASPVG), HQ USAF, performed an analysis of THUNDER 6.3 to assess the adequacy of THUNDER to provide measures of merit for nine Theater CINC unclassified campaign objectives (CO) (ASPVG, 1995). This analysis breaks the COs down into Operational Objectives (OO) which are further broken down into Operational Tasks (OT). This top-down approach was similar to the effort to create the Universal Joint Task List (UJTL) by Dynamics Research but yielded measures more amenable to analysis using THUNDER (UJTL 1996). Listed below are the nine COs used by ASPVG to analyze THUNDER:

1. Halt Invading Armies
2. Marshall and Sustain In-Theater Assets
3. Evict Halted Armies from Friendly Territory
4. Gain and Maintain Air Superiority
5. Gain and Maintain Sea Control
6. Gain and Maintain Space Control
7. Gain and Maintain Information Dominance
8. Deny Possession and use of Weapons of Mass Destruction
9. Suppress National Capacity to Wage War

Using this analysis methodology allows for a top-down look at the output of THUNDER as it applies to those objectives considered important by the CINC.

3. Methodology

Chapter 3 covers the methodology used for the accomplishment of this thesis. First, the overarching objectives of this thesis are presented. Then the chronological milestones of this effort are discussed. Figure 5 shows a pictorial representation of the stair-step methodology used throughout this thesis effort.

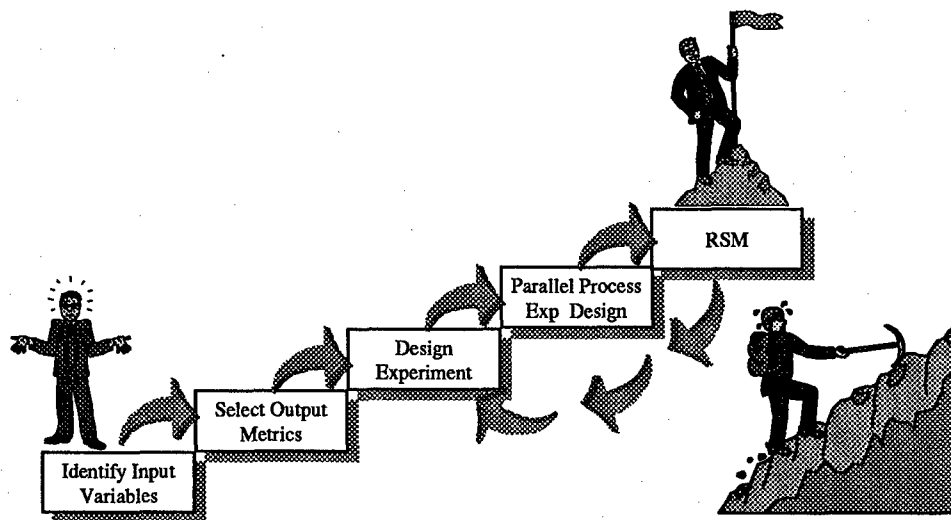


Figure 5. Thesis Methodology

3.1 Objectives

To satisfy the requirements of this research as stipulated by AFSAA/SAAB, the following steps are accomplished:

1. Select C2 related input variables amenable to perturbation to accomplish the sensitivity analysis.
2. Select THUNDER output metrics that can be used to accurately measure the attainment of the specified Campaign Objectives (COs).

3. Design the experiment to provide the maximum amount of information with the minimum number of runs.
4. Run the experiment in a parallel processing environment.
5. Use RSM to develop the requested C2 response surfaces and metamodel.

3.2 Identify Input Variables

Command and control warfare is not just hardware, software, systems, or procedures. It is an integrated military strategy focused on attacking the command and control capabilities of an adversary while protecting friendly C2 capabilities.

Lt Col Norman B Hutcherson

It is this integrated nature of C2 warfare that requires us to investigate a broad spectrum of C2 variables. As shown in Figure 6 the identification of the input variables is the first step.

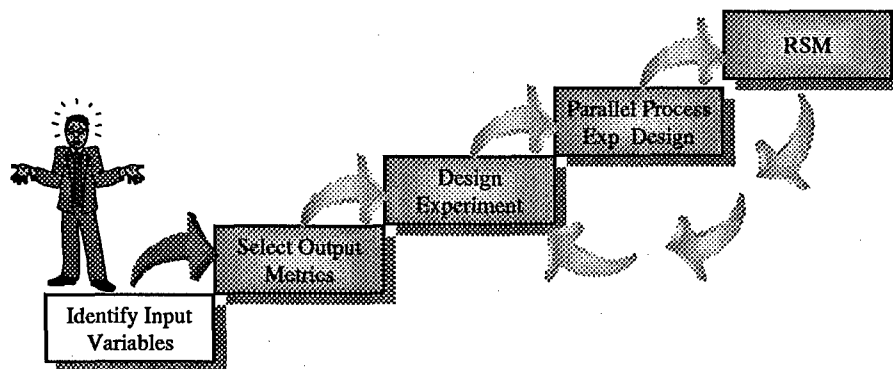


Figure 6. Input Variable Definition

As mentioned in Chapter 1, a THUNDER scenario involves over 80 different data files and hundreds of variables. Our goal is to identify all possible C2 variables that are modeled in THUNDER and then investigate all variables capable of perturbation at THUNDER's high ISR level of resolution.

3.3 THUNDER Data Files

Those data files that have a bearing on this thesis are listed below, and pertinent excerpts from those data files are included in Appendix A.

- | | |
|------------------------|--|
| 1. <i>detect.dat</i> | Probability of engagement given a detection. |
| 2. <i>airrules.dat</i> | Perception data. |
| 3. <i>squadron.dat</i> | Aircraft quantity information. |
| 4. <i>adsector.dat</i> | IADS integration information. |
| 5. <i>typec3.dat</i> | Ground force message capacity. |
| 6. <i>grdrules.dat</i> | Unit strength degrade information. |
| 7. <i>srec.dat</i> | Standoff Reconnaissance effects. |

In addition to these data files, the following files are integral to the proper operation of THUNDER and should be carefully checked before run initiation:

- | | |
|-----------------------|--|
| 1. <i>THUNDER.CTL</i> | Controls the basic operation of THUNDER with path information to data directories and executables. |
| 2. <i>control.dat</i> | System configuration information – length of war, random number information |
| 3. <i>ttgraph.cfg</i> | required in the run directory for generation of <i>ttgraph.rpt</i> |
| 4. <i>macro.afit</i> | This output generation macro is created interactively and its location specified in <i>THUNDER.CTL</i> |

3.3.1 Air Assets

3.3.1.1 Aircraft Variables

The E-3 Sentry is an airborne warning and control system (AWACS) aircraft that provides all-weather surveillance, command, control and communications needed by commanders of U.S. and NATO air defense forces. As proven in Desert Storm, it is the premier air battle command and control aircraft in the world today (USAF Fact Sheet 96-13).

The Joint Surveillance Target Attack Radar System (JSTARS) is an airborne platform equipped with a long-range, air-to-ground surveillance system designed to locate,

classify and track ground targets in all weather conditions. Its capabilities make JSTARS effective for dealing with any contingency, whether actual or impending military aggression, international treaty verification or border violation (USAF Fact Sheet).

The A-50 Mainstay SDRLO (Long Range Detection System) provides Russian Fighter Regiments with an airborne control capability aircraft and has seen widespread service with Soviet Forces. Developed to replace the TU-126 Moss (a variant of the Bear bomber) and equipped with an upgraded "Flat Jack" radar system, the Mainstay first flew in 1980 with about 40 produced by 1992 (<http://www1.tpgi.com.au/users/RAAF/AEWCAIR.HTM>).

3.3.1.2 Design Levels for Air Assets

The two aircraft whose primary mission is C2 are the AWACS and Mainstay. The JSTARS, while not directly tasked for C2 activities, plays a major role in the identification and targeting of Scud missiles and launchers and surface-to-air missile sites for coalition aircraft (Standoff Reconnaissance SREC). The baseline scenario, as it pertains to these variables, developed by ASC/XR is shown in Appendix A in Figure 28. All values are from unclassified sources. Baseline data for US AOB is from Conduct of Persian Gulf War- Final Report to Congress (Department of Defense, 1992: 106). Currently there are only two JSTARS in the USAF inventory, but 13 are slated to be in by 2004. The Red AOB is from Storm Over Iraq (Hallion, 1992: 146).

For the proposed experimental design we need to determine low and high level values for these variables. To set the upper bound for aircraft force levels we assume the greatest single increase of PAA approved by Congress in a given year would be a 50

percent increase (Grier, 1997: 26). Likewise, for the low levels assume that other operational commitments would reduce the availability of these aircraft by 50 percent.

The levels chosen are indicated in Table 5.

Table 5. C2 Aircraft Design Levels

	LOW LEVEL	MID LEVEL	HIGH LEVEL
AWACS	12	15	18
JSTARS	6	8	10
MAINSTAY	2	3	4

3.3.2 Integrated Air Defense System (IADS)

An IADS is a combination of sub-systems whose overarching goal is defense against an air attack. The *front line* of an IADS is made up of the various surface-to-air weapons (surface-to-air missiles (SAMs) and anti-aircraft artillery (AAA)) and the sensors needed to acquire the targets and fire the weapons (early warning/ground control intercept (EW/GCI) radar and acquisition radar). In addition to these fixed assets, interceptor aircraft and repositionable tactical assets round out the arsenal of an IADS. The IADS components that impact this thesis include: Integrated Operations Centers (IOC), Sector Operations Centers (SOC), Jammers, Weasels, EW Radars and Acquisition Radars.

The IOC serves to orchestrate the operations of the EW/GCI, SAMs and AAA. Information is routed from these sub-systems to the IOC, which then forms a coherent picture of the battlefield and allocates defensive assets to counter the threat. The SOC is responsible for the coordination of the use of interceptor aircraft and repositionable tactical assets throughout its sector of responsibility. Jammers and Weasels are aircraft

whose mission is either to reduce acquisition and fire control burnthrough (lethal radius) of an enemy radar, or to actively search for and destroy it.

3.3.2.1 Integration Levels

Integration refers to the probability that an IADS site is actually integrated in the IADS. Non-integrated sites are considered autonomous. ASC/XR has developed a rather intuitive approach to modeling the integration level of the forces based upon six IADS related variables: whether there exists an IOC or SOC, whether there are jammers or Weasels present, and the fraction of surviving EW and acquisition radars. Do these variables all impact C2? Consider the operation and effectiveness of the F-4G Wild Weasel's with their high-speed anti-radar missiles (HARM) during Operation Desert Storm. During this conflict the Iraqis did not use their sector operations centers and radars with a Weasel present because if a system was on for more than a few seconds, operators risked meeting a HARM missile (USAF Fact Sheet 91-03). With these systems turned off, C2 is essentially nonexistent. Perturbing the levels of these six independent variables will result in different levels of ability for any given force to communicate and to provide fire control guidance. The hierarchy for the different communication and fire control ability states to be used in this thesis is shown in Appendix A as Figure 29.

The IADS rules detailed in *adsector.dat* determine the probability of integration. Depending upon the variable values the integration factor varies from a high level of 1.0 down to a low level of 0.0. An example of the calculations is shown in Figure 7 with an explanation of the different states located in Appendix A.

```
IF CommState = 2 AND FireState = 1
    THEN
        LET INTEGRATION = 1.0
    ENDIF

    IF (CommState = 2 AND FireState = 5)
    OR (CommState = 5 AND FireState = 1)
        THEN
            LET INTEGRATION = .75
        ENDIF

        IF (CommState = 3 AND FireState = 1)
        OR (CommState = 2 AND FireState = 2)
        OR (CommState = 5 AND FireState = 3)
        OR (CommState = 2 AND FireState = 5)
            THEN
                LET INTEGRATION = .50
            ENDIF
```

Figure 7. Integration Calculation -- *adsector.dat*

3.3.2.2 Design Levels for IADS Integration

The integration levels generated in *adsector.dat* represent the baseline scenario. In this case though, the baseline scenario can be interpreted as the optimum integration level achievable under the given conditions. We propose the integration level can be perturbed downward to simulate various battlefield conditions that would contribute to the "fog of war." Alternatively, the degrade in integration can be viewed as the result of a potential capabilities increase. For example, the modification of the Weasel to provide increased operational effect would result in increased integration degrade over current capabilities.

The addition of a degrade calculation to the IADS.RULES.SEGMENTS would provide this capability.

Integration Degrade Calculation

$$\text{INTEGRATION} = \text{INTEGRATION} * \text{INTEGRATION.DEGRADE} \quad (22)$$

THUNDER's logic structure will not support a value for integration larger than one. Therefore the levels for this variable were chosen so that the high level represented no degrade over the IADS rules calculations – multiplication factor of one. The baseline value for this effect represents a 25 % degrade in integration over and above the integration level determined by THUNDER. Finally, the low level represents a 50 % degrade in capabilities. Table 6 shows the levels used in this thesis.

Table 6. Integration Degrade Levels

MULTIPLIER	LOW LEVEL	BASELINE	HIGH LEVEL
INTEGRATION.DEGRADE	0.5	0.75	1

3.3.3 Unit Strength

A unit's strength is determined by many factors in THUNDER, two of which are C2 related. The first is the volume of message traffic a unit can process. The second is the amount of unit strength degradation experienced with a delay in message processing. Both are directly related to a unit's posture and the calculation of the force ratio.

Unit posture is a function of the unit's force ratio, user defined rules, orders, and previous FLOT movement. THUNDER has seven postures: Blue Attack Red Delay

Defense (BADD), Blue Attack Red Hasty Defense (BADH), Blue Attack Red Intense Defense (BADI), Static (STATIC), Red Attack Blue Delay Defense (RADD), Red Attack Blue Hasty Defense (RADH), and Red Attack Blue Intense Defense (RADI) (THUNDER Analyst Manual (TAM) Volume Two p. 16). The THUNDER methodology for determining unit strengths and unit force ratios is shown in Figure 8.

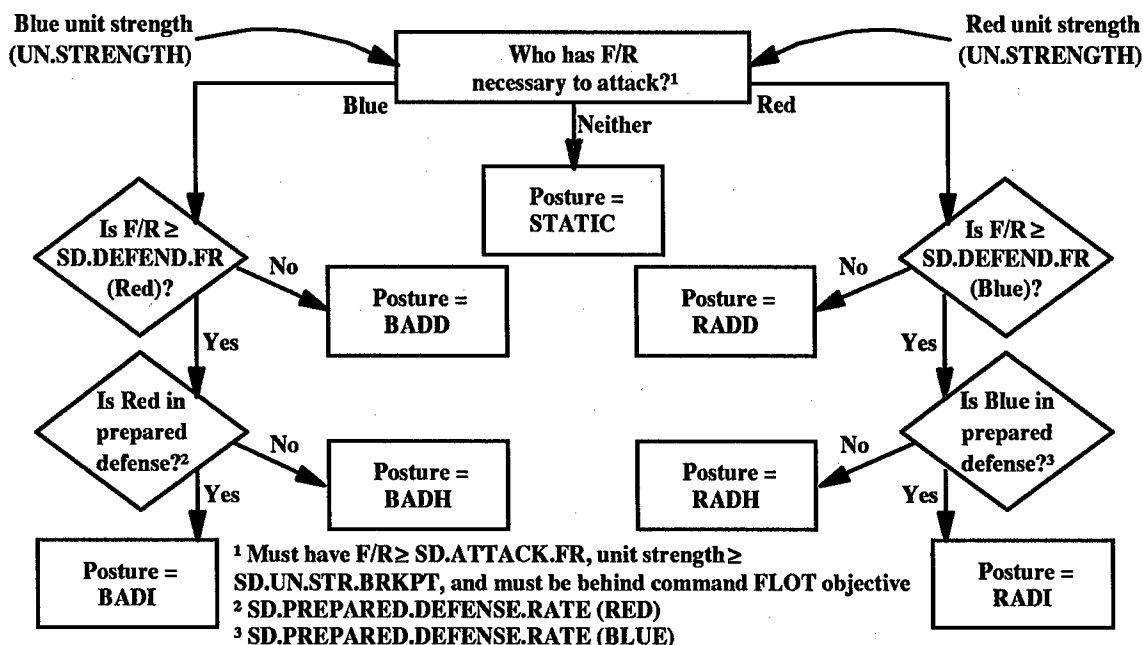


Figure 8. Unit Posture Calculation

(S3I THUNDER 6.4 Advanced Course Briefing Slide)

Integral to the calculation of a unit's posture is the unit's force ratio which compares a unit's strength to that of the enemy's. One aspect of a unit's strength investigated in this thesis is the effect the ground force's message capacity on the various MOOs. With the complete complement of a ground unit's C3 facilities (e.g., vans and antennas) operating, their message processing capacity is maximized and message

requirements are based upon the current warfighting posture (see Appendix A, Figure 27). However, in the conduct of warfare a unit's C3 facilities are rarely fully operational. As air to ground sorties attack and destroy a unit's C3 facilities, the overall unit's message processing capacity decreases. If the message capacity decreases below the message demand by posture listed in Appendix A, Figure 27, a delay in message processing occurs. This in turn, directly affects a unit's measure of strength.

3.3.3.1 Design Levels for Message Capacity

The high and low level values for C3 message capacity are calculated by perturbing the baseline values from the STORM database by 25 percent. The different values used in the experimental design are shown in Table 7.

Table 7. Message Capacity Design Levels – *typeC3.dat*

	Message Capacity Design Levels		
	Low	Baseline	High
Blue Unit C3 Fac	113	150	188
Blue CMD C3 Fac	225	300	375
Blue Log Fac C3 Fac	75	100	125
Red Unit C3 Fac	113	150	188
Red CMD C3 Fac	225	300	375
Red Log Fac C3 Fac	75	100	125

In THUNDER a unit's strength is adjusted according to the status of their C3 facilities. A user defined curve relating message processing delay to a degrade in a unit's strength defines this relationship and is contained in *grdrules.dat*. The baseline degrade curve is shown in Table 8 (Unclassified STORM database).

Table 8. C3 Unit Strength Degrade Curve -- *grdrules.dat*

Unit Strength C3 Degrade Curve	
Delay Hours	Percent Degrade
1.00	0
1.70	10
2.90	15
4.25	20
5.50	25

3.3.3.2 Design Levels for C3 Degrade

To study the effects of the UNIT.STRENGTH.C3.DEGRADE.CURVE on the model output this curve is perturbed from the baseline level to low and high levels offset from the baseline values by 25 percent changes in each direction. These values were selected to represent optimistic (high level) and pessimistic (low level) views on the impact C3 degrade has upon a unit's strength. The design levels for UNIT.STRENGTH.C3.DEGRADE.CURVE used for this thesis are shown in Table 9 and graphically in Figure 9. These degrade curves were selected to "cover the waterfront," which was necessary given the goal of this thesis was to ensure the analysis adequately assesses the sensitivity of THUNDER to this variable. Regardless of our curve's validity, we suspect this is a powerful influence in THUNDER, and those design points with minimal strength degradation should show improved results over those design points using the low level values (maximum strength reduction).

Table 9. Unit Strength C3 Degrade Design Levels

C3 Degrade Design Levels			
Delay	Low	Baseline	High
1.00	0	0	0
1.70	12.50	10.00	7.5
2.90	18.75	15.00	11.25
4.25	25.00	20.00	15.00
5.50	31.25	25.00	18.75

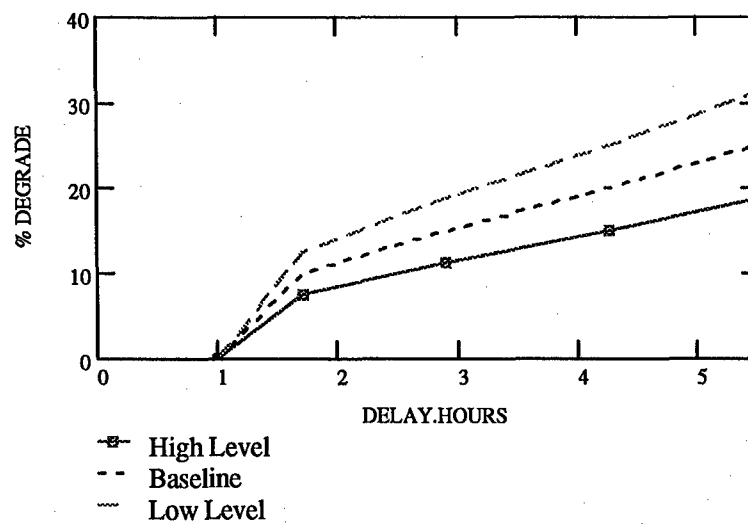


Figure 9. Unit Strength C3 Degrade Curve

3.3.4 Standoff Reconnaissance (SREC)

3.3.4.1 Variables

3.3.4.1.1 SREC Effects Multiplier

THUNDER models attrition of forces using the Attrition Calibration (ATCAL) methodology developed by the U.S. Army Concepts Analysis Agency (TAM Volume Two p. 16). This is accomplished through the impact of direct and indirect fire weapons. The

lethality of these weapons can, understandably enough, be increased through the synergistic effect of intelligence updates on target location. In THUNDER, SREC is performed by JSTARS, an airborne platform equipped with a long-range, air-to-ground surveillance system designed to locate, classify and track ground targets in all weather conditions.

Although still in development during Operation Desert Storm, two JSTARS test aircraft flew 54 combat sorties and supported all mission taskings with a system availability rate of more than 80 percent. One of the two aircraft was in the air every day identifying and targeting Scud missiles and launchers, convoys, trucks, tanks, surface-to-air missile sites and artillery pieces for coalition aircraft (USAF Fact Sheet 91-03).

THUNDER provides the ability to perform SREC against four types of equipment categories: infantry, helicopters, armor and artillery. The effect of SREC is modeled by applying a multiplier to the lethality of the indirect and direct fire weapons. The maximum achievable multiplier is a user input according to the weapon, target and unit posture and is dependent upon the amount of SREC coverage the target's zone/sector received (TAM Volume Two p. 16). The maximum SREC multiplier information is contained in *srec.dat* and an excerpt is shown in Figure 10.

BEGIN.SREC.AC.IDS

1098

END.SREC.AC.IDS

SREC.MAX.EFFECT.MULT.BY.SHOOTER.SREC.EQUIP.CAT.BY.TGT.SREC.EQUIP.C
AT.BY.POSTURE

Weapon	Target	POSTURE						
		BADD	BADH	BADI	STATIC	RADI	RADH	RADD
10001	10001	1.300	1.300	1.300	1.100	1.100	1.200	1.300
	10002	1.300	1.300	1.300	1.100	1.100	1.200	1.300
	10003	1.300	1.300	1.300	1.100	1.100	1.200	1.300
	10004	1.300	1.300	1.300	1.100	1.100	1.200	1.300

Figure 10. SREC Maximum Effect Multiplier – *srec.dat*

This SREC table tells us that a weapon of type 10001 against a target of type 10002 can potentially increase the weapons lethality by 130 percent if the unit's posture is BADD.

3.3.4.2 Design Levels for SREC Multiplier

This thesis will attempt to answer whether or not the effect of SREC is accurately modeled by THUNDER. This is done by evaluating the effect of perturbing the multipliers listed in *srec.dat*. Specifically, the SREC multipliers are varied by plus or minus 25 percent to obtain the high and low levels for the experimental design. The complete SREC Multiplier tables are included in Appendix A.

3.3.5 Air to Air Engagements

3.3.5.1 Variables

3.3.5.1.1 Engagement Probabilities

The probability that an aircraft engages an enemy aircraft given detection occurs is listed in *detect.dat*. *Detect.dat* is somewhat of a misnomer for this data file as the probabilities listed for the blue/red aircraft combinations are not detection probabilities at all, but are only conditioned upon detection and relate the probability an engagement occurs. Not all detections will result in an engagement. The probabilities listed represent, in an aggregate manner, the sensor suite, cockpit visibility, ability to reposition aircraft, etc., which affect the course of an engagement (TAM Volume Two p. 43). An excerpt of this data file is included in Figure 11. This data file is pertinent to this thesis effort in that the engagement probabilities are a function of the airborne early warning (AEW) state. THUNDER can model four different AEW states: no AEW, Blue AEW, Red AEW, and Blue/Red AEW.

DETECT.PROBS.205									
MULT.FACTORS...BLUE...RED									
NO.AEW		1.00		1.00					
BLUE.AEW		1.25		0.75					
RED.AEW		0.75		1.25					
BOTH.AEW		1.00		1.00					
BLUE.KILLER.AC									
2001				2002					
2005				2006					
END.RED.TGT.AC...PROB.DETECT.IF..NO.									
AEW..BLUE.AEW..RED.AEW..BOTH.AEW									
1001	20	20	20	20	20	20	20	20	20
90	90	90	90	90	90	90	90	90	90

1002	60	60	60	60	60	60	60	60	60
90	90	90	90	90	90	90	90	90	90

Figure 11. Engagement Probabilities – *detect.dat*

3.3.5.1.2 AEW Control State

The multipliers listed in Figure 11 (MULT.FACTORS) represent the AEW control state (ACS) of the scenario at the time of detection. These multipliers allow the analyst the ability to model the effectiveness, or ineffectiveness, of different AEW strategies. Additionally operational guidelines are easily be built into this table. For example, if aircraft 1002 benefited more by having Blue AEW present than did aircraft 1001, then without changing the ACS multipliers, which affects every aircraft pairing, the advantage aircraft 1002 has over 1001 is modeled as shown in Table 10. In this situation aircraft 1002 has with the larger engagement probability after the ACS multiplier is applied to the engagement probability for the two aircraft.

Table 10. Capability Increase with Constant ACS Multipliers

	AEW CONTROL STATE			
	No AEW	Blue AEW	Red AEW	Both AEW
1001	20	20	20	20
	90	90	90	90
1002	60	75	60	75
	90	100	90	100

3.3.5.2 Design Levels for AEW Control States

The baseline values for the ACS multipliers listed in Figure 11 were selected to represent the relatively static nature of the battle when neither/both sides have AEW and the offensive advantage obtained when your opponent does not have AEW coverage.

The data from the unmodified *detect.dat* is used as baseline values for the ACS multipliers. The low level and high level values are calculated by decreasing and increasing the baseline values respectively. The multipliers that will be used are listed in Table 11.

Table 11. ACS Multiplier Design Levels

ACS MULTIPLIER	LOW LEVEL		MID LEVEL		HIGH LEVEL	
	BLUE	RED	BLUE	RED	BLUE	RED
NO.AEW	0.75	0.75	1.00	1.00	1.25	1.25
BLUE.AEW	0.313	0.188	1.25	0.75	1.56	0.94
RED.AEW	0.188	0.313	0.75	1.25	0.94	1.56
BOTH.AEW	0.75	0.75	1.00	1.00	1.25	1.25

3.3.6 Intelligence, Surveillance, and Reconnaissance (ISR) Effects

3.3.6.1 Variables

3.3.6.1.1 ISR Levels of Resolution

THUNDER's ISR submodel allows for three distinct levels of ISR resolution described by the model documentation as low-resolution, high resolution, and very high

resolution (TAM V2 p118). In the low resolution mode opposing sides are given perfect intelligence on their adversary. In both the high and very high modes, this intelligence is imperfect and degrades over time. However, through ISR updates the intelligence can be refreshed through information gathered during RECCE and SREC sorties. The difference between the high and very high modes of operation is the level of detail to which the intelligence efforts are directed. In the very high mode intelligence is collected on individual targets, while in the high mode, intelligence gathering efforts are directed towards zone/sectors. Each side's, Blue or Red, ISR level of resolution is user specified in *control.dat* making it possible to model different levels of resolution. For the purposes of this thesis, the high level of ISR resolution provides sufficient detail for the analysis.

3.3.6.1.2 Zone/Sector Perception Degrade

One of the surest ways of forming good combinations in war would be to order movements only after obtaining *perfect information* of the enemy's proceedings.

Jomini, 1862 p. 268

With the recent advances in information technologies the ability of our forces to achieve the *perfect information* idealized by Jomini has been significantly enhanced. With aircraft like the AWACS and JSTARS this level of perfection is all but realized – at least while they are on-station.⁴ Should these aircraft have to go off-station for any reason, then of course, the movement of the Red forces could go undetected and our confidence in their location would be degraded.

⁴ Unfortunately, THUNDER does not model the immense ISR capabilities of the RC-135 Rivet Joint. This awe inspiring aircraft can just about achieve Jomini's ideal single-handedly.

In THUNDER's ISR high resolution mode a general level of perception for each zone/sector is calculated and applied to each target within the zone/sector. This value of perception is either aged as the model runs, or updated with intelligence reports. The perception values are then applied to "truth" values to determine perceived enemy attributes (TAM Vol 2 p. 118).

As should be expected in a dynamic combat environment, this level of perception is not a constant. The input parameters to age the level of perception are contained in *airrules.dat* under the guise of DEGRADE.INTERVAL and DEGRADE.PERCENTAGE. An excerpt from *airrules.dat* is shown in Figure 12.

PERCEPTION.DATA	
INITIAL.PERCENTAGE.OF.GROUND.TRUTH (INT)	75
DEGRADE.INTERVAL (HOURS)	24
DEGRADE.PERCENTAGE (INT)	5
LOCATION.ERRORS.FOR.FIXED.TARGETS	NO
MEAN.ERROR.QUANTITIES	
AMMUNITION.SUPPLIES (PCT)	1
DRY.BULK.SUPPLIES (PCT)	1
EQUIPMENT.SUPPLIES (PCT)	1
POL.SUPPLIES (PCT)	1
WATER.SUPPLIES (PCT)	1
LOG.FACILITY.ISSUE.CAPACITY (PCT)	1
FORCE.RATIO (DEC)	.10
UNIT.STRENGTH (PCT)	0
MESSAGE.PROCESSING (PCT)	100
COUNTS.OF.THINGS (PCT)	50
LENGTHS.OF.THINGS (PCT)	1
LOCATION.OF.THINGS (METERS)	1000
VELOCITY.OF.THINGS (PCT)	1
ARC.THRUPUT (PCT)	1
AIRBASE.MAINTENANCE.CAPABILITY (PCT)	9
STRATEGIC.TARGET.DAMAGE (PCT)	25

Figure 12. Perception data -- *airrules.dat*

The perception of targets within a given zone/sector is aged based upon the number of DEGRADE.INTERVALs, i , that have passed since the last update. More accurately, the amount of perception lost is calculated according to:

$$\text{New Perception} = \text{Old Perception} * \left(1 - \frac{D}{100}\right)^i \quad (23)$$

Where D is the value for DEGRADE.PERCENT from Figure 12.

The perception level is modified when the perception from new information is fused with the old perception level and updated according to:

$$\text{Perception} = \left(C + R - \left(\frac{C * R}{100} \right) \right) \quad (24)$$

Where C is the current perception level and R is the effective report level, which takes the perception update from the sensor and degrades the observation for the passage of time since the information was gathered.

3.3.6.1.3 Mean Error Quantity (MEQ)

Perceived knowledge of ground targets is influenced by three factors (TAM Vol 2 p. 120):

1. The true values for the target attributes.
2. The level of perception for a target's zone/sector.
3. The value of the Mean Error Quantity.

For each target, attribute values are generated randomly from a normal distribution. The "true" value of the component is used for the mean of the normal distribution, and the standard deviation is calculated by using the MEQ in the following equation:

$$StdDev = M * \frac{MEQ}{100}. \quad (25)$$

The one exception to this calculation is with the Force Ratio MEQ. This MEQ is input as a dimensionless number and in this case the standard deviation is set equal to the MEQ value. In the preceding equation U is the uncertainty level for the zone/sector which is computed as:

$$U = \frac{(100 - P)}{100}, \quad (26)$$

where P is the zone/sector perception level.

3.3.6.2 Design Levels for Zone/Sector Perception Degrade

The perception degrade curves are generated using Figure 12 by beginning with an initial perception level of 75%. The baseline value selected for perception degrade was a 25% degrade in the perception of targets within a given zone/sector. Low and high levels were determined by perturbing the perception degrade by $\pm 20\%$ in *airrules.dat*. These curves do not reflect any changes to perception levels from intelligence updates. An equally viable method to investigate the effect of degraded perception over time would be to alter the DEGRADE.INTERVAL value. However, the former methodology was selected and the perception degrade percentages used in this thesis are shown in Table 12.

Table 12. Perception Degrade Design Levels

	LOW LEVEL	BASELINE	HIGH LEVEL
Perception Degrade	45 %	25 %	5 %

3.3.6.3 Design Levels for Mean Error Quantities

The three MEQs amenable to perturbation in this study are force ratio, unit strength, and message capacity. In this analysis, the baseline values are selected according to Table 13. The low and high levels were once again shifted from the baseline values by $\pm 25\%$. From equation 25, a value of zero for an MEQ means we have no variation in a target's perceived state. A value of 100 allows for the largest standard deviation, representing a worst-case ability to precisely determine an enemy's location. The values chosen for the high level represents a high level of uncertainty. Also, these variables are not treated individually. Rather, they are perturbed in a lock-step fashion where all three are set to the same level based upon the specific design point.

Table 13. Design Levels for Mean Error Quantities

	Design Levels		
	Low	Mid	High
Force Ratio	0.55	0.75	0.95
Unit Strength	95	75	55
Msg Processing	95	75	55

3.3.7 Input Variable Summary

Table 14 lists the lower, center, and upper values for all of the different variables that will be perturbed during the course of this study.

Table 14. Input Variable Values

Variable	Lower	Baseline	Upper
AWACS	6	12	18
JSTARS	3	6	9
Mainstay	1	2	3
Integration Degrade	-25%	0%	+25%
C3 Degrade	+25%	0%	-25%
SREC Multiplier	-25%	0%	+25%
ACS Multiplier	-25%	0%	+25%
Perception Degrade	-20%	0%	+20%
Blue Message Capacity	-25%	0%	+25%
Red Message Capacity	-25%	0%	+25%
MEQ	-25%	0%	+25%

3.4 Select Output Metrics

As shown in Figure 13 the next chronological step in this analysis is to choose output measures. THUNDER can produce a mountain of output. It is extremely important to select those output measures pertinent to both the study at hand, and ultimately, to the decision maker.

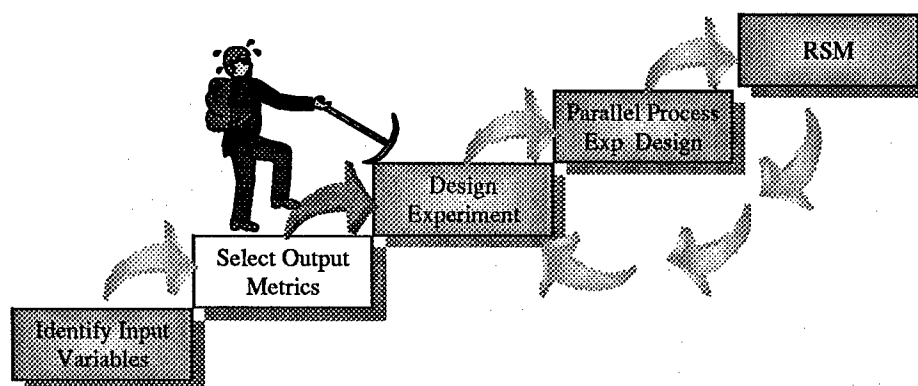


Figure 13. Select Output Metrics

In any complex simulation model, the final outcome is influenced by a myriad of different variables. Accordingly, the perturbation of a single variable is unlikely to

significantly change the larger outcome of the conflict within THUNDER (TAM Vol 3 p. 98). Consequently, considerable effort was expended to identify those measures of effectiveness, providing meaningful insights into the effect of the variable perturbations.

As important as it may be to provide meaningful insights into the effect of the perturbations, the selection of output measures must be tempered by the nature of the question being answered. COs may provide the answers required, but unfortunately, THUNDER doesn't have an output measure called "Halt Invading Armies." Operational Capabilities (OCs) which actually quantify the effect of a given alternative are needed. The ASPVG methodology can be used to break down the COs into the OCs, which are measurable by THUNDER. But which THUNDER output measure best applies to the different COs? Some excellent guidance for selecting OCs is contained in the draft Office of Aerospace Studies (OAS) Analysis of Alternatives (AOA) Guide. Several of the guidelines contained therein are listed below (OASP 97-1, 1997):

1. Do the OCs measure battle or engagement outcome (if applicable)?
2. Are the OCs quantifiable?
3. Do the OCs allow for discrimination among alternatives?
4. Can the OCs be linked or related to the COs?
5. Are the OCs evaluated for all alternatives?
6. If weighted OCs are used, do they distort the comparisons? Was sensitivity analysis done on the weights?
7. Do the OCs relate to the alternatives being evaluated?

Using these criteria as a basis and evaluating the metrics used by Grier, we derived the MOEs used in this study (see Grier 1997: 67):

1. Exchange Ratio – defined as the total number of Air-to-Air (AA) and Surface-to-Air (SA) kills for Blue divided by those for Red.
2. Total number of Red Tanks, Artillery, Armored Personnel Carriers, Trucks, Helicopters and Infantry Vehicles destroyed.
3. Days to achieve Air Superiority – defined as the day when Red sorties drop below 10% of day 1 levels.
4. Days to halt FLOT movement – this measure tracks the number of days required before Blue transitions from losing to gaining ground.

Appendix B provides a complete listing of all nine COs, OOs, OTs, and OCs from which the above MOEs were derived.

3.5 Design Experiment

As shown in Figure 14 the next step along the journey is to design the experiment to provide the maximum amount of information with the minimum number of runs.

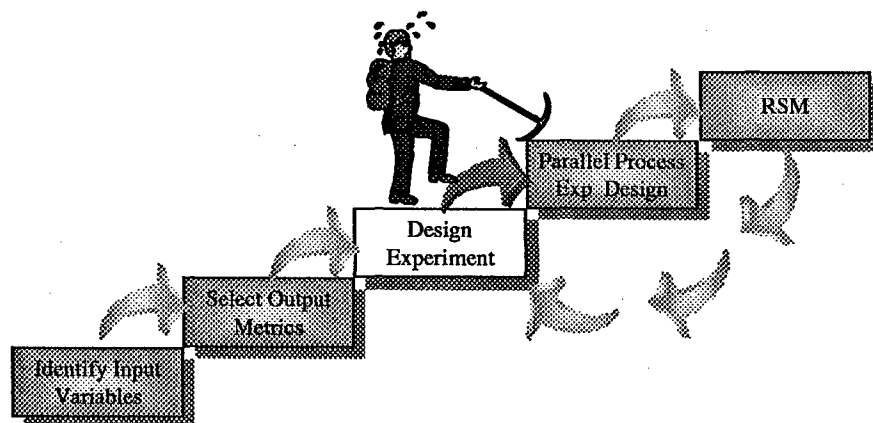


Figure 14. Experimental Design

With 11 input variables, this objective is accomplished most efficiently with a PB_{III} design (see Table 2). As a Resolution III design, we obtain the unaliased single factor effects needed to make a screening decision. As discussed previously, multiple replications at each design point will be required to provide an average value for each of the effectiveness measures. Thirty replications are performed at each design point.

The next section outlines the method used to process the multiple THUNDER replications. What is necessary at this point though, is an understanding of the methodology used to develop and prepare the experimental design for implementation. As shown in Figure 15, the PB_{III} design used for the screening experiment was generated by JMP® and then processed by Excel into the form required by the parallelization script and the master data files. This process was not automated, but is relatively straightforward to implement. Also, if more than one design is envisioned this process will easily pay for the time required to understand the mechanics involved.

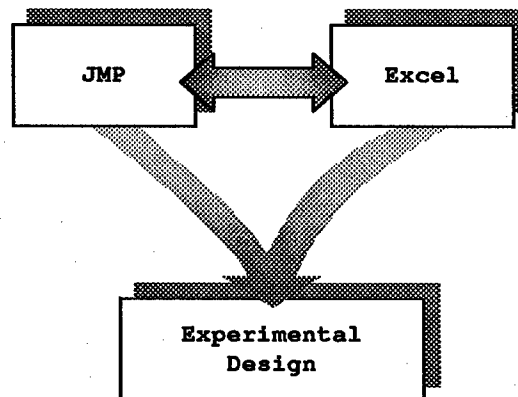


Figure 15. Experimental Design Generation

The process simply involves exporting the experiment matrix of values, (-1, 0, 1), from JMP® into Excel. Then, in Excel, this matrix is mapped to another matrix using standard if-then-else structures as illustrated below:

```
if ( $x_k < 0$ ) then  $x_k = \text{ACS\_LOW\_}$ 
else if ( $x_k > 0$ ) then  $x_k = \text{ACS\_HIGH\_}$ 
else  $x_k = \text{ACS\_BASE\_}$ 
```

This transformed matrix now contains the flags used during the decomposition phase of the parallelization script. A final step added a column onto the beginning of this matrix containing the design point designators used by the parallelization script to build and name subdirectories for each design point. The full matrix of flags used in the Plackett-Burman validation design is included in Appendix D as Table 43.

3.6 Parallel Processing of Experimental Design

At this point, we have reached the usual study *bottleneck*. The input variables are determined, the output measures are defined and the appropriate experimental design is developed for performing the C2 sensitivity analysis. Now, as shown in Figure 16 our task involves developing a methodology for performing the simulations runs in a parallel environment. The methodology discussed in the next several pages and the Perl script developed to implement this methodology set about duplicating the process to remove the source of our analytical bottleneck, using THUNDER in an experimental design.

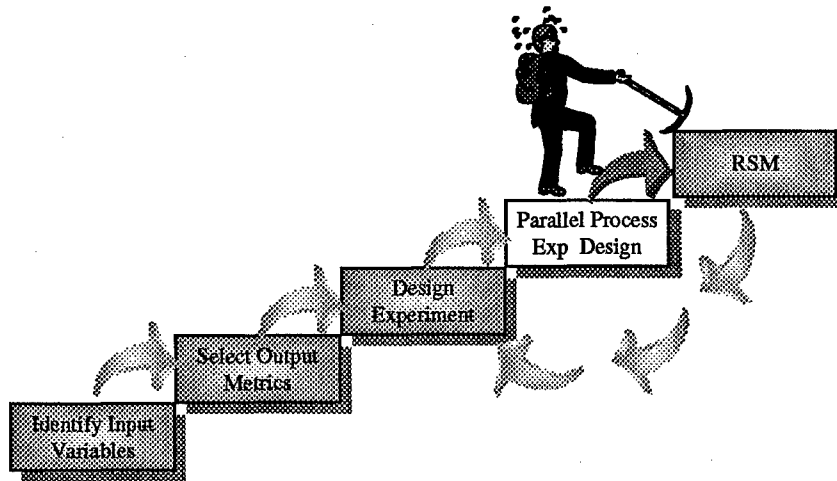


Figure 16. Parallel Processing

This step was in fact the alternative impetus behind this thesis. In support of the SIMAF Proof of Concept project we were tasked with providing a methodology for providing an efficient means of accomplishing the experimental design necessary for an AoA thesis effort requiring THUNDER to evaluate alternative notional air superiority scenarios (Siegener, 1998).

One of our overarching goals in support of the SIMAF project was to demonstrate the analytical performance enhancements made possible from the parallelization of data gathering efforts using MSRC organic resources. These resources include Cray, IBM, and SGI mainframe computers as well as a network of SGI workstations in the MSRC's Science and Visualization (SCI/VIS) laboratory. The mainframe computers are used exclusively for batch mode processing and the SCI/VIS computers allow for the interactive processing required by our parallelization methodology.

Building on the methodology of Skordos (Skordos, 1994: 5) discussed in Chapter 2, the logical flow of our parallelization efforts can be summarized by Figure 17. Flowing

from left to right around the direction arrow we can take an experimental design, modify the necessary data files, create the THUNDER run environment, execute model runs on multiple machines and multiple CPUs and finally prepare the output data for analysis with any commercially available statistical package. Appropriately enough, the script which accomplishes all of this is titled MMMP for Multiple Machine Multiple Processors. A complete listing of the computer code is contained in Appendix C.

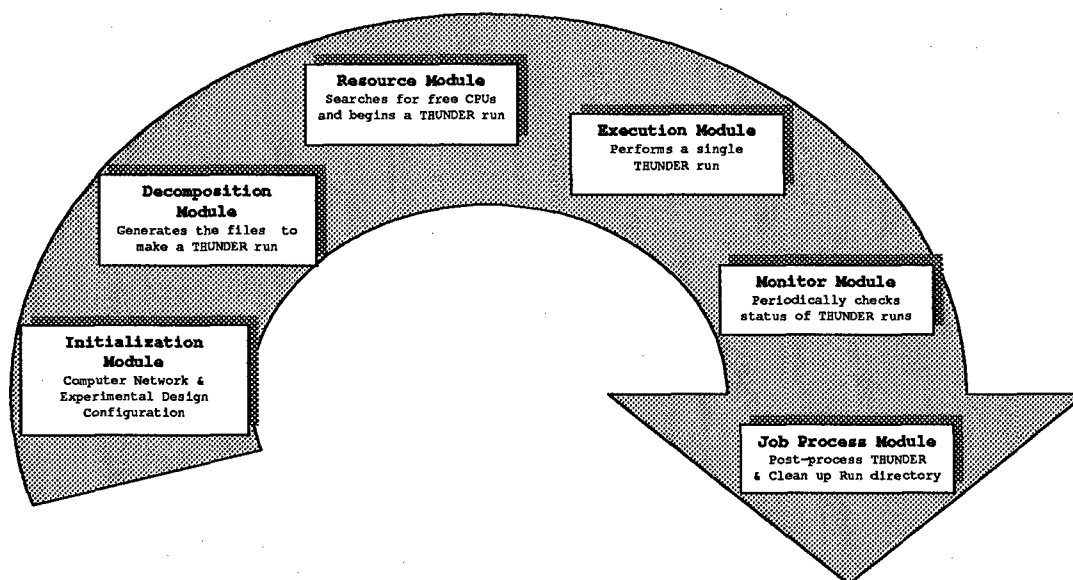


Figure 17. MMMP Parallelization Methodology

The design of MMMP and its operation in a parallel operating environment relies heavily upon the availability of a homogeneous network of workstations, a common file system and the ability to remotely execute commands on remote machines. The first requirement is achieved through the use of the SCI/VIS lab, which contains ten SGI workstations with a total of 21 central processing units (CPUs), ranging in capability from a R4400 to a R10000. All ten of these SCI/VIS computers are interconnected via a

Network File System (NFS), satisfying our second criteria. A NFS allows the system administrator to mount a remote computer's directory structure onto the local computer. More importantly for an analyst, with a NFS'd computer network, every computer has access to the same file structure and is essentially no different than having the entire system mounted on the local computer. Finally, our ability to remotely execute commands throughout the network is made possible through a UNIX command, rsh (remote shell), which allows us to execute the THUNDER runs on the network machines without needing to log in interactively.

There are essentially three different tasks the MMMP script accomplishes. First, MMMP does an initialization job to read in the computer network information and build the cases and status directories and enter the *monitor* subroutine loop. Second, when monitor has spawned a job onto a remote machine, it performs an execute job. The tasks performed during an execute job include: creating the necessary run directory, executing THUNDER and processing the data upon run completion. Finally, MMMP performs a monitor job periodically (user defined) looking for completed jobs or jobs to kill (if they have hung and/or exceed a maximum time limitation). Having completed these tasks, *monitor* searches for free machines and launches jobs on available machines. When all available machines are busy the script goes into a sleep mode for a period of time, only to awaken and repeat the process repeatedly until all jobs are completed.

3.6.1 Initialization Module

This section has no pretensions about providing the reader with a full understanding of how to operate in a UNIX environment. Rather, we will endeavor to

explain those aspects of this environment that provided the most complications during our *climb to the top*. UNIX in a Nutshell, an O'Reilly & Associates, Inc. publication, can help fill in any knowledge gaps remaining after reading this section.

As was mentioned previously, the ability to spawn THUNDER onto remote machines without interactive intervention was a requirement for experimental design parallelization. The goal of course, is to set up the experimental design, launch MMMP, and then have data in a format that is ready to be analyzed the next day. This launch and leave capability is provided by the rsh command which connects a local machine to the specified hostname and executes the specified command (UNIX man page rsh(1)). For example, *rsh \$hostname MMMP* launches a copy of the MMMP script on the remote machine *\$hostname*. This of course, assumes the host machine is configured to allow for remote execution of commands. For our computer network at the MSRC this involved creating a *.personal.rhosts* file containing the official name of the computers we wanted to remotely access (Notice this is a hidden file, as indicated by the *dot* prefix). The location and name of this file varies from network to network, therefore the best advice available is to check with the system administrator for local requirements.

Another requirement is the remote machine must know where to look for the command to execute. With a remote execution, the host computer searches for the specified command to execute in the locations designated by the global variable *\$path*. The first occurrence of the command is executed, so do not leave old versions of a script lying around or it may be launched instead, with unexpected results. If the remote

machine is unable to locate the script, then an addition to the *.cshrc* file is required. For this project, we needed to include the following in our *.personal .cshrc* file:

```
set path = ($path ${HOME}/TT64/bin)
```

```
set path = ($path ${HOME}/MMMPDIR)
```

The first tells the computer where to look for THUNDER, and the second tells it where to look for our script.

Another script essential to the operation of MMMP is Perl. The first line of MMMP, *#!/usr/local/bin/perl*, uses the “shebang” notation which tells the operating system to associate MMMP with the Perl interpreter located in */usr/local/bin*. Perl is never in the same place twice, and the system administrator must be consulted to find the path to the correct version of Perl.

3.6.2 Decomposition Module

The previous section, while not a specific subroutine of the MMMP script, nonetheless provides ample opportunity for frustration if not properly established before MMMP execution. MMMP works as advertised, but if the necessary initialization is not accomplished beforehand, launching MMMP only result in a “permission denied” error message.

MMMP is based upon the directory structure shown in Figure 18 with the MMMP directory located at the top level. Below the MMMP directory are five subdirectories, only one of which needs to be populated prior to launching the script.

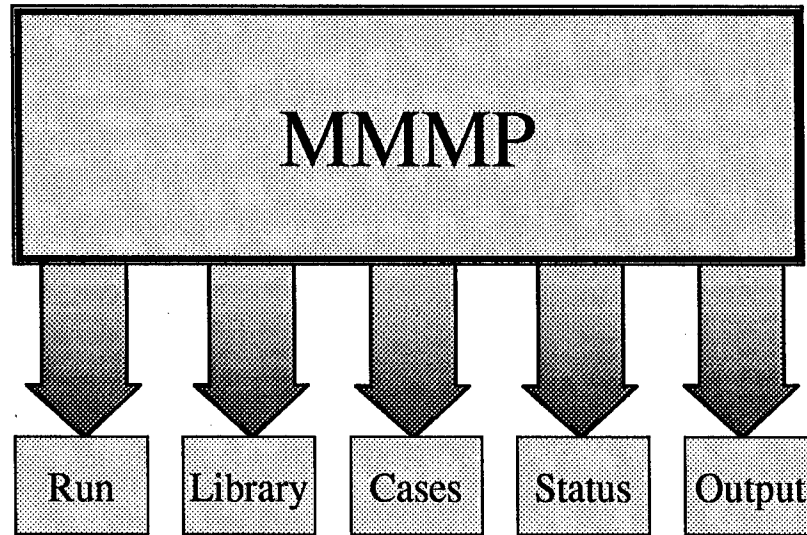


Figure 18. MMMP Directory Structure

For each individual case/rep being executed a separate subdirectory within the run directory is created by the subroutine *mk_run_dir* from which THUNDER can be launched. In each of these run directories *mk_run_dir* builds the four directories required by THUNDER. In addition it brings in the other files need to operate THUNDER.

One of these files is *seedval.dat*. This file is used to control the random number streams within THUNDER. The *seedval.dat* file is built by another subroutine, *mk_new_seed*. This subroutine gives the user the option of creating independent replications of THUNDER or using common random numbers. The ability to use common random numbers is enabled simply enough by copying the same library copy of *seedval.dat* into each run directory. For independent replications, *mk_new_seed* changes the random number seed values within this file before it is copied into the run directory by *mk_run_dir*. The application of common random numbers as discussed in the previous chapter and advocated by Law and Kelton to a complex stochastic model like THUNDER

would be the subject of an interesting thesis topic, but was not attempted during this effort. Instead we used the method of independent replications to allow us to use classical statistical techniques when analyzing our output.

The library directory is the repository of files used by MMMP during the course of execution to initialize and create the separate run environments. Files required in the library directory include:

1. Master data files
2. Random number seed file
3. *THUNDER.CTL*
4. *ttgraph.cfg*
5. *control.dat*
6. *DESIGN.CTL* – (see Cases directory discussion)

The output directory is empty at program inception. As replications of a design point are completed, user specified files (*MMMP.CTL*) are consolidated in a storage directory within the output directory (e.g., *S_Case_10*). There will be one storage directory created for each design point.

The cases directory is built by MMMP during the decomposition phase of the program. It is composed of subdirectories, one each for every design point containing the modified data files associated with that point. Modifying and creating these directories manually would have been a formidable task. Fortunately, with the use of cpp preprocessor directives, there is a much simpler way to accomplish this task. Figure 19 illustrates what happens to the library's master data files as they are accessed by the *mk_CASES_dir* subroutine.

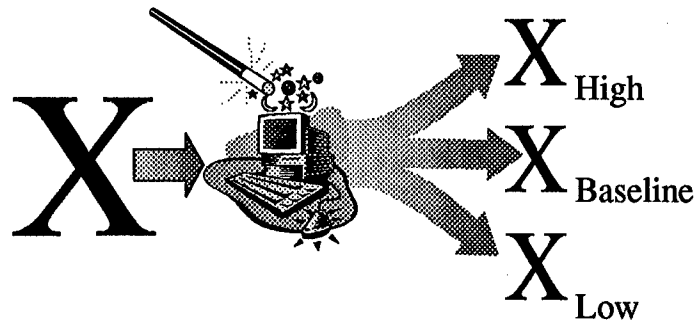


Figure 19. Data File Preprocessing

The values in the different data files that need to be modified have been replaced by unique variables (X). Then, almost as if by magic, by using the cpp preprocessor directives this variable is transformed into the high, baseline or low level as needed for each design point. To implement the cpp preprocessor directives, the master THUNDER data files are be modified so that a unique variable is substituted for the THUNDER parameter value in those positions requiring modification for the design point. Then the flags read in from *DESIGN.CTL* determines if this variable is set to its high, baseline, or low value. An example of how we implemented the cpp preprocessor directives and modified a data file is included at the end of the MMMP script in Appendix C. For further information about how to implement the cpp command refer to the UNIX cpp help (manpage for cpp).

3.6.3 Resource Module

This module as well as the execution module, resides within the *monitor* subroutine but are broken out here and in the next section for clarification. THUNDER is processor intensive. One THUNDER job is about all a CPU can handle. Send a CPU two

runs of THUNDER simultaneously and you will have brought the CPU to its *knees*. To prevent this, three separate checks are required before a run of THUNDER is launched on any given machine, as illustrated by Figure 20.

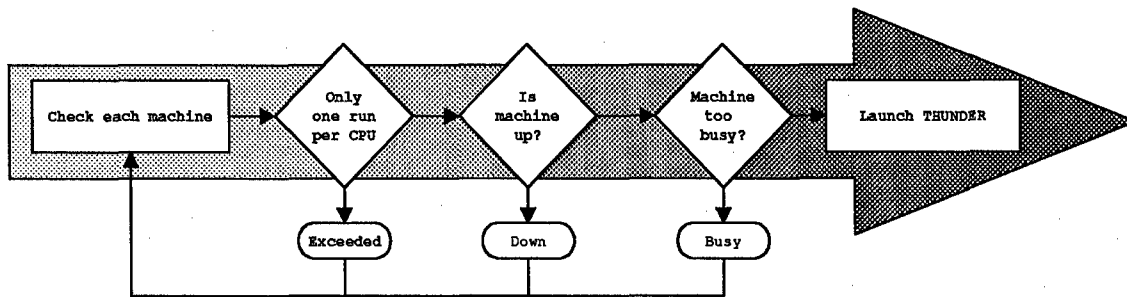


Figure 20. Resource Module Methodology

The different figures shown above are representative of the processes MMMP goes through in evaluating the suitability of machine to accept a THUNDER run. First, a machine is found. In *read_machine*, MMMP reads in the *machine.dat* file from the MMMP directory. This is done during every monitor period to allow the user the opportunity to take machines out of the cycle if needed. Assuming this file exists and is not empty, the subroutine continues.

The first suitability check performed is on the number of jobs currently executing on the machine. The limit is one job for each CPU. Coincident with a THUNDER execution is the creation of a file in the status directory with a filename that uniquely identifies the machine, case and rep associated with the model run (e.g. Run_\$machine_Case_\$case_Rep_\$rep). The *find_machine* subroutine searches through this directory for files beginning with Run and compares the search results for a given machine with the number of CPUs available on that machine (CPU information was read

in during the *read_machine* subroutine). As jobs complete or fail, this label is changed to Complete or Fail respectively.

The next two checks take advantage of UNIX's *rup* command. Executing the command *rup \$machine* would result in the following output:

```
$machine (up or down) 59 days, 20:34 load average 0.15, 0.09, 0.14
```

The *find_machine* subroutine then parses this output to determine first, if the machine is even up and secondly, if so what its load average is. This last bit of information appears to be a subjective area with no solid guidance concerning what load level limit should be observed before launching a THUNDER run. However we observed that if the load was much above 0.8/CPU THUNDER executed rather slowly.

3.6.4 Execution Module

Assuming there is a THUNDER job remaining and an available machine, the *monitor* subroutine remotely launches a run of THUNDER. The remote machine is told what case and rep to do as well as where to do the work. This last bit is rather important, since upon *rsh*'ing to a machine the user will begin in their home directory. Since MMMP is built to operate in the MMMP directory the computer needs to be told that is where to look for the MMMP executable. Having successfully initiated an execution job, the *execute* subroutine spawns a run of THUNDER in the *spawn_a_run* subroutine. It is within *spawn_a_run* that the actual operation of THUNDER takes place. Here, *mk_run_dir* is called upon to build the THUNDER specific operating environment, execute each of the commands necessary to perform a single THUNDER job and then finally to call the *process_job* subroutine to manage the model's output.

3.6.5 Monitor Module

As was mentioned previously, MMMP purposefully enters an endless monitoring loop (Figure 21), breaking out of this loop only temporarily to spawn, or kill a job. When all jobs are completed, a permanent exit condition is created and all that is left for MMMP to do is post-process the data for exportation into Excel.

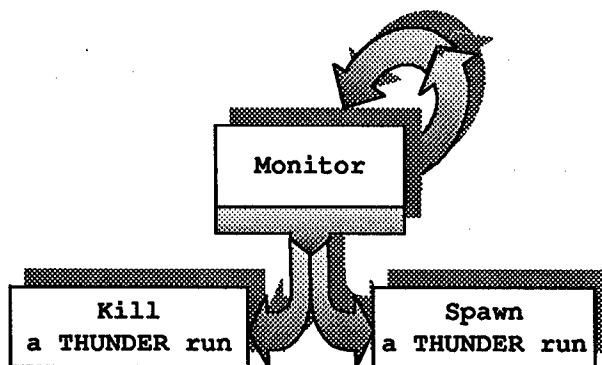


Figure 21. Monitor Subroutine Options

3.6.6 Job Process Module

As was discussed in the initialization and execution module section, knowing where to look for a script is important, but perhaps even more important when working with a model like THUNDER is knowing what to do with its mountains of output. Generally speaking, a 30-day war with THUNDER requires in excess of 100 Mb of storage capacity. This means that our initial Plackett-Burman screening design with 12 design points and 30 replications per design point would require storage in excess of 36 Gb, or approximately 25,000 3.5" floppy disks!

While this did not seriously impact the MSRC, MMMP was written with memory management practices in mind for applications upon more ordinary networks. To control

the amount of memory required after a run of THUNDER is complete, MMMP allows the user to either retain the entire run directories, or to specify which files to save, tar or compress. The default option in MMMP is to remove the run directories. A minimal save is recommended if space is at a premium. This should include as a minimum, those files that would allow further analysis should interest in an unanticipated MOE suddenly become important. We chose to save the *seedval.dat*, *control.dat*, *ttgraph.rpt* and all summary reports for each individual run.

The final step required is to prepare the output for use by Excel. Specifically this step converts the THUNDER output file, *ttgraph.rpt*, into a comma delimited file that is easily imported into Excel.

The preceding sections briefly discussed the operation of MMMP. Now, as shown in Figure 22, the grunt work is done and the real work of analyzing the output and forming conclusions and recommendations begins.

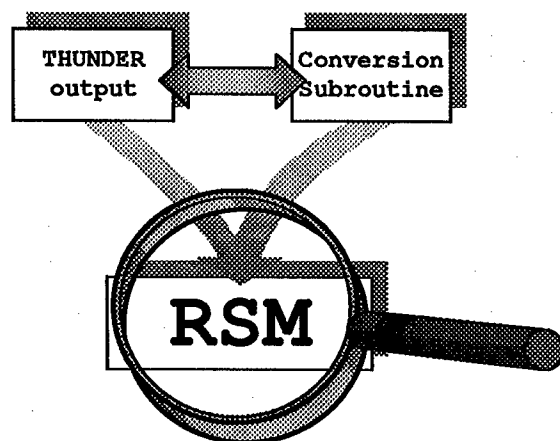


Figure 22. Job Process Module Overview

To do our output analysis, which is discussed in the next section, we had to first get the data off the mainframe computer and onto the PC. One option was to use the previously mentioned script and transform the *ttgraph.rpt* output file into comma separated files. Another more direct option at getting at the MOE information needed for this thesis was to use scripts written specifically for this purpose. The scripts written to perform this task are titled:

1. *get_moe* – The master script which calls all other scripts.
2. *get_FLOT* – Extracts the days to halt Red advance measure.
3. *get_redloss* – Extracts the total number of Red aircraft losses.
4. *get_blueloss* – Extracts the total number of Blue aircraft losses.
5. *get_airsup* – Extracts the day air superiority was achieved.
6. *get_tahait* – Extracts the total number of Red ground targets destroyed.

Assuming the user has specified *ttgraph.rpt* be saved, then upon program completion in the data output storage directories for each design point there is a *ttgraph.rpt* for each replication. The *get_moe* script when launched from the output directory accesses each design point's storage location, goes through every *ttgraph.rpt* contained therein and creates a separate MOE file for each design point. At this point it is a simple task to use the *mk_file_csv* script and transform this into a format recognizable by Excel. Now we have reached the point where the data from one phase of the THUNDER runs has been post-processed into matrix form. Next, RSM analysis is performed on this data to determine a regression equation that adequately fits the THUNDER data.

3.7 Response Surface Methodology

At this point we have almost reached the pinnacle (see Figure 23) in our analysis efforts. This is not to say this upcoming step of the project is trivial. In fact considerable

effort is required to complete this particular step. What makes this step noticeably more difficult than the previous steps is that there is an *art* to making the tactical decisions necessary for the development of a metamodel. There is no single right way to do something. This of course means the possibility exists (and happened more than once) to make a wrong turn into uncharted territory. Also, this is not a single step, but rather an iterative step where designs hypothesized are modified until some termination criteria is satisfied.

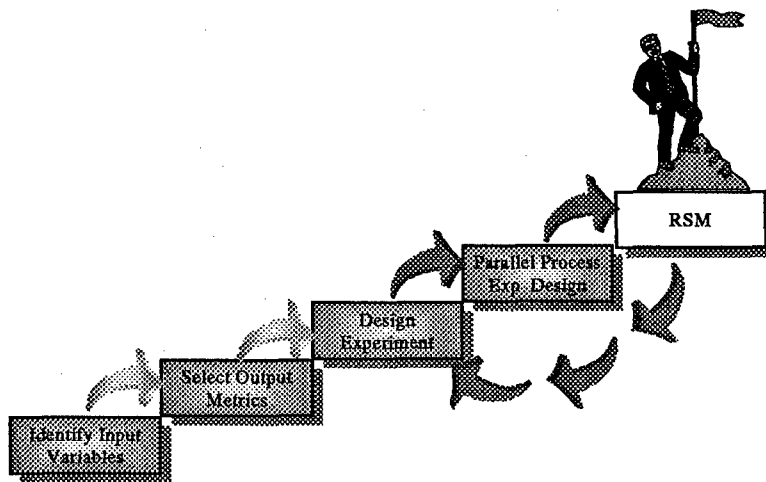


Figure 23. Response Surface Methodolgy

To keep our efforts focused we modified the RSM flow diagram from Chapter 2 into a methodology that would allow us to perform the necessary sensitivity analysis. The methodology is depicted in Figure 24. In general terms, our sensitivity analysis fits the most parsimonious metamodel, which adequately describes the behavior of THUNDER relative to the modeled C2 inputs. We do this by first making the assumption that not all of the C2 variables investigated significantly impact the outcome of a THUNDER scenario. Working under this assumption, we distill the 11 selected variables down to a

subset of significant variables using a Plackett-Burman screening design. A Resolution V design was constructed and performed to assess the true nature of the main and two-factor interactions. From the results of this experiment a metamodel describing the behavior of THUNDER is constructed. Finally, since our hypothesized metamodel is built with a subset of the original variables we perform a validation experiment to ensure we have accurately captured the essence of THUNDER relative to study objectives.

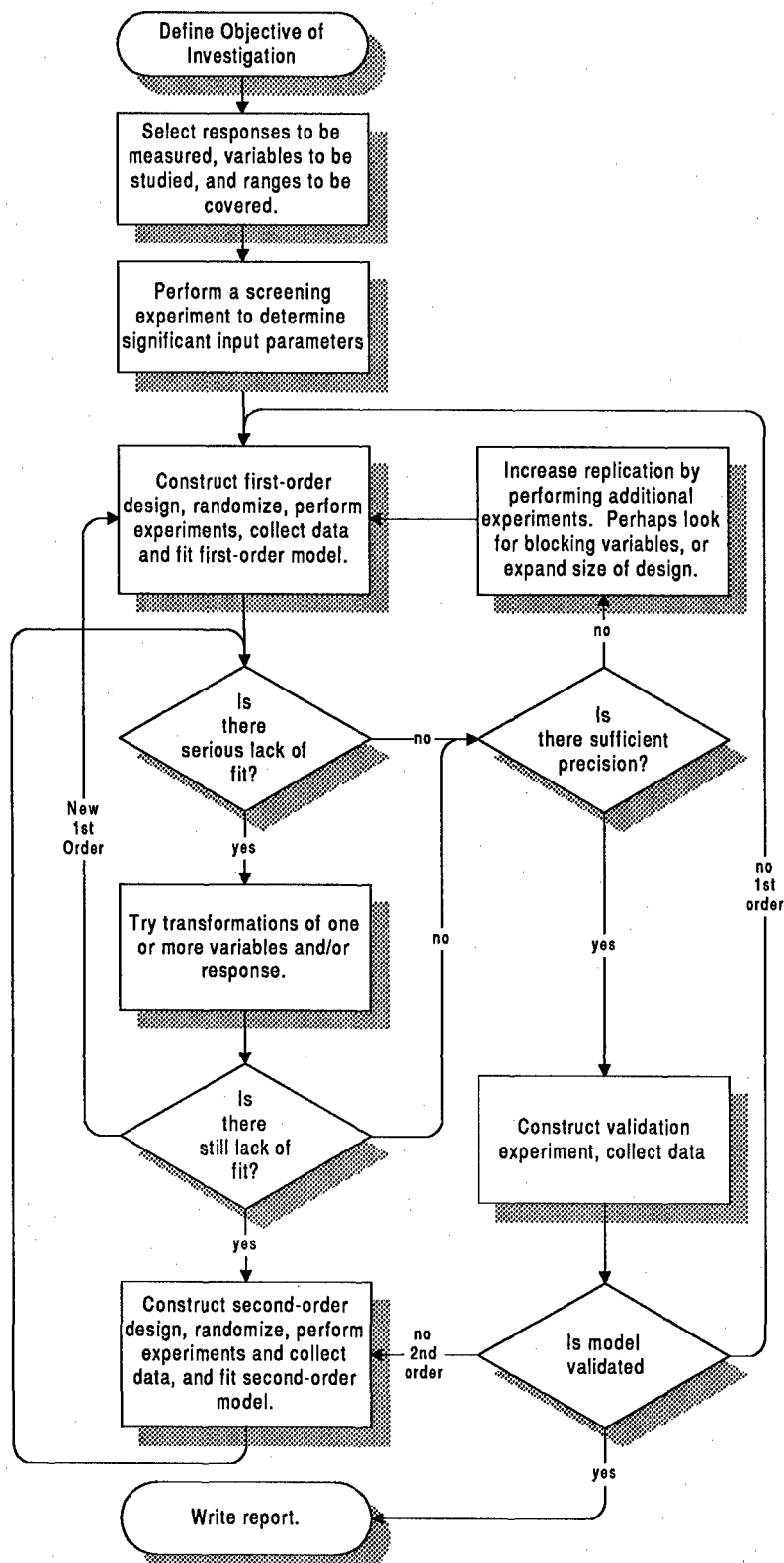


Figure 24. Sensitivity Analysis Flow Diagram

3.7.1 Effects Screening

Careful experimental design is required for the experiment to yield the expected results. To distill the 11 selected variables down to a subset of significant variables using a Plackett-Burman screening design we used two methods, both accepted, one more formal than the other. The formal test involved evaluating JMP®'s Orthogonal t-Test. Variables were considered significant under this test if their "Prob>|t|" value was less than .05 (95% significance level).

The other method used was to evaluate the normal probability plot of the variable estimates to assess the importance of a variable's contribution to the MOEs. The normal probability plots generated by JMP® show the effect estimates on the vertical axis and normal quantiles on the horizontal axis. If all effects are due to random noise, they will lie on a straight line with slope σ , standard error (JMP® Manual, 1995:190). Any points deviating from this line are considered significant.

3.7.2 Resolution V Design

A Resolution V designs provide substantial amounts of information in the most efficient manner possible. Specifically, no main effects or two-factor interactions are aliased with any other main effect or two-factor interaction. But two-factor interactions can be aliased with three-factor interactions. The metamodels generated from the Resolution V design will then be validated for predictive ability.

3.7.3 Metamodel Validation

Validation is an often-overlooked step, yet is very important. What we attempt to accomplish by validating the metamodel is to show there is a correspondence, or consistency between the metamodel and simulation model. This of course assumes prior simulation model validation.

As discussed in Chapter 2, there are several methods for validating a metamodel. We will generate another set of data within the design space to check the consistency between the metamodel and actual simulation results. We perform another Plackett-Burman experiment, this time using the levels listed in Table 15.

Table 15. Validation Experiment Variable Levels

Variable	Lower	Baseline	Upper
AWACS	9	12	15
JSTARS	4	6	7
Mainstay	1	2	2
Integration Degrade	-12.5%	0%	12.5%
C3 Degrade	12.5%	0%	-12.5%
SREC Multiplier	-12.5%	0%	12.5%
ACS Multiplier	-12.5%	0%	12.5%
Perception Degrade	-10.0%	0%	10.0%
Blue Message Capacity	-12.5%	0%	12.5%
Red Message Capacity	-12.5%	0%	12.5%
MEQ	-12.5%	0%	12.5%

Where possible we use coded values of $\pm \frac{1}{2}$ for our design levels. The only two variables where this was not possible were the JSTARS and Mainstay aircraft, and their levels were set according to the table. The coded levels for the variables are shown in Appendix D, Table 44.

The validation data set is generated by varying all 11 variables rather than just those deemed significant to check the practical effect of forming the metamodel with a subset of variables. Remember that it was assumed that not all variables would be significant in the final outcome of a campaign level model. If this is a good assumption, then the levels of the insignificant variables should not affect the predictive ability of the metamodel.

The results of the validation experiment are measured by their predictive ability. As discussed in Chapter 2 the two measures for the predictive ability of a proposed regression equation are the *MSPR* and the *MAPE*.

$$MSPR = \frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{n} \quad MAPE = \frac{100}{n} * \sum_{i=1}^n \frac{|Y_i - \hat{Y}_i|}{|Y_i|} \quad (27)$$

where: Y_i is the value of the response variable in the i th validation case (for this thesis, Y_i will be the 30 rep mean MOE value), $\hat{Y}_i$ is the predicted value for the i th validation case based on the model building data set, and n is the number of cases in the validation set.

4. Results

4.1 RSM Objectives

The preceding chapter discussed the methodology used for this research project.

This chapter presents the results.

The objectives of our analysis were:

- 1) Perform a Plackett-Burman screening experiment to determine which, if any of the 11 C2 input parameters warrant further investigation using a 95 % significance level.
- 2) Perform a 2_v^{5-1} fractional factorial design and generate parsimonious metamodels to determine the relationships between the C2 variables and MOEs.
- 3) Validate the Metamodels.

4.1.1 Overview

The following sections present the results of our analysis. Each successive section covers one of the objectives defined above. The analysis is performed using JMP® PC software (SAS Institute, Inc.). Appendix E contains data normality analysis and Appendix F includes detailed regression analysis output. Finally, to avoid any confusion the variables will hereafter be referred to by their proxy as listed in Table 16.

Table 16. Variable Identification

Variable	Proxy
AWACS	AWACS
JSTARS	JSTARS
Mainstay	MNSTY
Integration.Degrade	INT
C3 Degrade	C3DEG
SREC Multiplier	SREC
ACS Multiplier	ACS
Perception Degrade	PRCPT
Blue Message Capacity	BLUEMSG
Red Message Capacity	REDMSG
MEQ	MEQ

4.2 OBJECTIVE 1 – Plackett-Burman Screening Results

The Plackett-Burman Screening Design optimized the identification of the significant variables in this study. In all, 11 variables were analyzed at Resolution III using only 12 design points. The next most efficient design would have been a 2_{III}^{11-7} fractional factorial design requiring 16 design points to generate the same information as the Plackett-Burman design. Due to the stochastic nature of THUNDER each design point was replicated independently 30. The average value for each MOE is reported as the MOE value for that design point.

The coded design matrix and 30 rep average MOE values are listed in Table 17. Before doing any analysis of the THUNDER output though, we need to ensure the normality assumption holds. Appendix E shows the results of the normality assessment for each of the four MOEs. For the Plackett-Burman experiment, JMP® provided three checks of normality: the Normal Quantile Plot, a histogram of the data with a normal

curve superimposed and a formal test for normality, the Shapiro-Wilk test. All tests indicate the normality assumption holds at the 95% level.

Table 17. Coded Design Matrix and Average MOE Results for Plackett-Burman Screening Experiment

	C2 Input Variables											MOEs			
DESIGN POINT	ACS	AWACS	BLUEMSG	C3DEG	ENT	JSTARS	MNSTY	MEO	PRCPT	REDMSG	SREC	XRATIO	TAHAIT	FLOT	AIRSUP
1	1	1	1	1	1	1	1	1	1	1	1	1.14	60816.58	10.57	21.88
2	-1	1	-1	1	1	1	-1	-1	-1	1	-1	1.12	60291.21	10.66	24.53
3	-1	-1	1	-1	1	1	1	-1	-1	-1	1	1.12	61304.15	11.03	24.20
4	1	-1	-1	1	-1	1	1	1	-1	-1	-1	1.15	60390.88	10.59	22.74
5	-1	1	-1	-1	1	-1	1	1	1	-1	-1	1.10	60475.27	10.76	25.36
6	-1	-1	1	-1	-1	1	-1	1	1	1	-1	1.08	60509.88	10.72	25.53
7	-1	-1	-1	1	-1	-1	1	-1	1	1	1	1.10	61349.88	10.68	24.94
8	1	-1	-1	-1	1	-1	-1	1	-1	1	1	1.19	61659.73	10.94	22.51
9	1	1	-1	-1	-1	1	-1	-1	1	-1	1	1.11	60998.85	10.64	22.26
10	1	1	1	-1	-1	-1	1	-1	-1	1	-1	1.15	61448.94	10.59	22.97
11	-1	1	1	1	-1	-1	-1	1	-1	-1	1	1.12	62176.33	10.83	23.56
12	1	-1	1	1	1	-1	-1	-1	1	-1	-1	1.03	60955.90	10.75	23.50

4.2.1 MOE Normal Probability Plots

An analysis of the normal probability plots provide the first subjective test for ascertaining which of the C2 variables significantly affect the selected MOEs. The normal probability plots generated by JMP® show the effect estimates on the vertical axis and normal quantiles on the horizontal axis. If an effect is due to random noise, it will lie on a straight line with slope σ , standard error (JMP® Manual, 1995: 190). Therefore, any point deviating from this line may be considered significant. Figure 25 shows the normal probability plots of the four MOEs. The FLOT and Exchange Ratio plots do not have any points that deviate from normal significantly and initial indications were that none of the

C2 variables significantly impacted these two MOEs. On the other hand, the Air Superiority and Red TAHAIT plots show points deviating from normal: SREC and ACS for Air Superiority; SREC, BMSG, PRCPT and JSTAR for Red TAHAIT. These variables are considered significant based upon this test. The next four subsections will analyze JMP®'s formal t-test for variable significance.

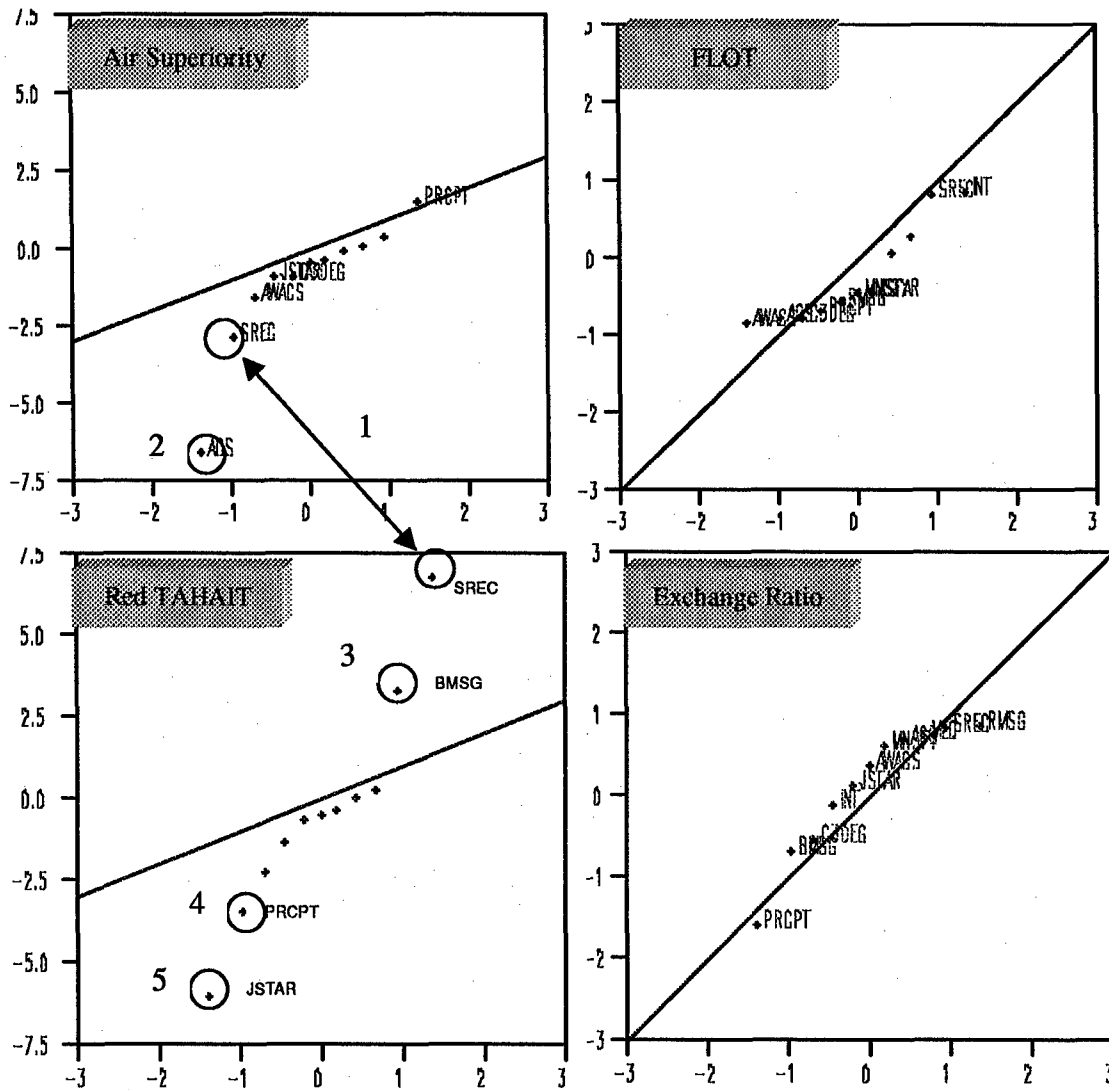


Figure 25. MOE Normal Probability Plots – Five Significant Variables

4.2.2 MOE #1: Days Needed to Achieve Air Superiority (AIRSUP)

Using JMP®, the results of the screening experiment for the first MOE are presented in Table 18. The parameter estimates indicate that ACS and SREC are the only two statistically significant variables at an α level of 0.05 (Prob>|t| less than 0.05). These variables are included in the Resolution V experimental design to build the final metamodel. This finding mirrors the analysis of the normal probability plots.

Table 18. Air Superiority Plackett-Burman Screening Results

Transformed Parameter Estimates				
Term	Original	Orthog Coded	Orthog t-Test	Prob> t
Intercept	23.66500	23.66500	151.4560	<.0001
ACS	-1.02167	-1.02167	-6.5387	<.0001
AWACS	-0.23833	-0.23833	-1.5253	0.1554
BMSG	-0.05833	-0.05833	-0.3733	0.7160
C3DEG	-0.14000	-0.14000	-0.8960	0.3894
INT	-0.00167	-0.00167	-0.0107	0.9917
JSTAR	-0.14167	-0.14167	-0.9067	0.3840
MNSTY	0.01667	0.01667	0.1067	0.9170
MEQ	-0.06833	-0.06833	-0.4373	0.6703
PRCPT	0.24667	0.24667	1.5787	0.1427
RMSG	0.06167	0.06167	0.3947	0.7006
SREC	-0.44000	-0.44000	-2.8160	0.0168

4.2.3 MOE #2: Days Needed to Halt FLOT (FLOT)

The results of the screening experiment for the second MOE are presented in Table 19. From this table it can be seen that at the 95% level there are no variables that are significant.

Table 19. FLOT Plackett-Burman Screening Results

Term	Transformed Parameter Estimates			
	Original	Orthog Coded	Orthog t-Test	Prob> t
Intercept	10.72976	10.72976	165.4639	<.0001
ACS	-0.04965	-0.04965	-0.7656	0.4600
AWACS	-0.05397	-0.05397	-0.8322	0.4230
BMSG	0.01855	0.01855	0.2861	0.7801
C3DEG	-0.04963	-0.04963	-0.7653	0.4602
INT	0.05595	0.05595	0.8629	0.4066
JSTAR	-0.02790	-0.02790	-0.4302	0.6754
MNSTY	-0.02850	-0.02850	-0.4395	0.6688
MEQ	0.00553	0.00553	0.0853	0.9336
PRCPT	-0.04323	-0.04323	-0.6667	0.5187
RMSG	-0.03658	-0.03658	-0.5641	0.5840
SREC	0.05317	0.05317	0.8200	0.4296

4.2.4 MOE #3: Red TAHAIT Destroyed

The results of the screening experiment for the third MOE are presented in Table 20. Analyzing the parameter estimates for this MOE reveal that BMSG, JSTAR, PRCPT and SREC are the statistically significant variables at an α level of 0.05. Here we see that SREC is significant in two different MOEs.

Table 20. TAHAIT* Plackett-Burman Screening Results

Term	Transformed Parameter Estimates			
	Original	Orthog Coded	Orthog t-Test	Prob> t
Intercept	61031.47	61031.47	1173.582	<.0001
ACS	13.68	13.68	0.2630	0.7974
AWACS	3.06	3.06	0.0589	0.9541
BMSG	170.50	170.50	3.2785	0.0074
C3DEG	-34.67	-34.67	-0.6667	0.5187
INT	-114.33	-114.33	-2.1984	0.0502
JSTAR	-312.88	-312.88	-6.0163	<.0001
MNSTY	-67.18	-67.18	-1.2919	0.2229
MEQ	-26.69	-26.69	-0.5132	0.6180
PRCPT	-180.41	-180.41	-3.4691	0.0052
RMSG	-18.76	-18.76	-0.3608	0.7251
SREC	352.79	352.79	6.7838	<.0001

*TAHAIT = Tank, Armored Personnel Carriers, Helicopters, Artillery, Infantry, Trucks.

4.2.5 MOE #4: Exchange Ratio

The results of the screening experiment for the final MOE are presented in Table 21. As was the case with the FLOT MOE at the 95% level, there are no variables that can be considered significant.

Table 21. Exchange Ratio Plackett-Burman Screening Results

Term	Transformed Parameter Estimates			
	Original	Orthog Coded	Orthog t-Test	Prob> t
Intercept	1.117147	1.117147	72.7911	<.0001
ACS	0.010401	0.010401	0.6777	0.5120
AWACS	0.005594	0.005594	0.3645	0.7224
BMSG	-0.010232	-0.010232	-0.6667	0.5187
C3DEG	-0.008343	-0.008343	-0.5436	0.5975
INT	-0.001849	-0.001849	-0.1205	0.9063
JSTAR	0.001957	0.001957	0.1275	0.9008
MNSTY	0.009334	0.009334	0.6082	0.5554
MEQ	0.011233	0.011233	0.7319	0.4795
PRCPT	-0.024218	-0.024218	-1.5780	0.1429
RMSG	0.013304	0.013304	0.8668	0.4045
SREC	0.012769	0.012769	0.8320	0.4231

The conclusion reached from aggregating the results of the individual t-tests for each MOE concurred with assessment made from the normal probability plots. Both tests indicated that SREC, ACS, BMSG, PRCPT and JSTAR should be included for further investigation.

4.3 Fractional Factorial Results

The five variables determined to be significant from the Plackett-Burman screening experiment are used in a Resolution V fractional factorial design. The configuration of this experiment required 20 design points in order to obtain the unaliased main and two factor interactions estimates as well as check for curvature. JMP® was used to generate

the Resolution V experimental design, Excel to convert this design into the form needed by MMMP, and then about one day later the results from all 600 runs were ready for analysis. The final outcomes of these 18,600 days of warfare are listed in Table 22. Each of the 20 different scenarios was simulated for 31 days and each scenario was replicated 30 times.

Table 22. Coded Design Matrix and Average MOE Results for Resolution V Fractional Factorial Experiment

 Design Pt	Resolution V C2 Variables					MOEs			
	ACS	BLEMSG	JSTAR	PRCPI	SREC	XRATIO	TAHAIT	FIOT	AIRSUP
1	-1	-1	-1	-1	1	1.112	61900.74	10.79	29.69
2	-1	-1	-1	1	-1	1.106	60694.2	10.66	30.32
3	-1	-1	1	-1	-1	1.127	60638.71	10.61	29.09
4	-1	-1	1	1	1	1.067	60730.09	10.6	29.23
5	-1	1	-1	-1	-1	1.114	61285.97	10.59	29.18
6	-1	1	-1	1	1	1.077	60891.43	10.61	29.97
7	-1	1	1	-1	1	1.077	61598	10.8	29.03
8	-1	1	1	1	-1	1.091	60076.63	10.74	30.09
9	1	-1	-1	-1	-1	1.169	61155.31	10.8	27.69
10	1	-1	-1	1	1	1.118	60968.31	10.54	27.43
11	1	-1	1	-1	1	1.132	61537.87	10.77	26.9
12	1	-1	1	1	-1	1.134	59950.59	10.74	28.68
13	1	1	-1	-1	1	1.148	61643.43	10.69	27.89
14	1	1	-1	1	-1	1.167	60612.2	10.63	29.46
15	1	1	1	-1	-1	1.139	60764.97	10.71	28.17
16	1	1	1	1	1	1.117	60880.66	10.63	27.54
17	0	0	0	0	0	1.133	60898.76	10.63	28.26
18	0	0	0	0	0	1.139	60852.43	10.72	28.26
19	0	0	0	0	0	1.123	60762.57	10.67	28.89
20	0	0	0	0	0	1.129	60794.73	10.69	28.74

Stepwise linear regression was performed on the output for each MOE. The output of this process was then iteratively used in a least squares regression analysis to generate a parsimonious metamodel for each MOE. Appendix F contains detailed regression results for each MOE.

4.3.1 MOE #1: Days Needed to Achieve Air Superiority

As was necessary with the Plackett-Burman analysis, we needed to first check for normality. This time, in addition to the three tests available during the Plackett-Burman analysis we were able to analyze a residual plot to check for model adequacy. Any discernable pattern within the residual plot could indicate a departure from normality. The residual plot for this MOE showed no significant pattern and all other normality tests were adequate so we felt safe in concluding the normality assumption was not violated.

Appendices G and H contain the detailed normality analysis.

With this assumption satisfied we are able to form the air superiority metamodel and continue with the analysis. Table 23 contains JMP® generated parameter estimates for this MOE.

Table 23. Air Superiority Metamodel Parameter Estimates

Parameter Estimates						
Term	Estimate	Std Error	t Ratio	Prob> t	Lower 95%	Upper 95%
Intercept	28.7255	0.057407	500.38	<.0001	28.599148	28.851852
ACS	-0.8025	0.064183	-12.50	<.0001	-0.943766	-0.661234
BLUEMSG	0.14375	0.064183	2.24	0.0467	0.0024841	0.2850159
ACS*BLUEMSG	0.15125	0.064183	2.36	0.0380	0.0099841	0.2925159
JSTAR	-0.18125	0.064183	-2.82	0.0165	-0.322516	-0.039984
PRCPT	0.3175	0.064183	4.95	0.0004	0.1762341	0.4587659
SREC	-0.3125	0.064183	-4.87	0.0005	-0.453766	-0.171234
PRCPT*SREC	-0.235	0.064183	-3.66	0.0037	-0.376266	-0.093734

The air superiority metamodel is shown below:

$$\hat{y} = 28.7255 - 0.8025 x_{ACS} + 0.14375 x_{BLUEMSG} + 0.15125 x_{ACS} * x_{BLUEMSG} \\ - 0.18125 x_{JSTAR} + 0.3175 x_{PRCPT} - 0.3125 x_{SREC} - 0.235 x_{PRCPT} * x_{SREC} \quad (28)$$

Looking at Table 24 we see that the R^2 and R^2_{ADJ} for this model are quite high, indicating this particular metamodel accounts for a high degree of variability in the observed responses.

Table 24. Air Superiority Metamodel Summary of Fit

Summary of Fit	
RSquare	0.95752
RSquare Adj	0.926625
Root Mean Square Error	0.256731
Mean of Response	28.7255
Observations (or Sum Wgts)	20

The last check we need to make for this metamodel is for lack of fit. As we pointed out in Chapter 2 our hope is to *fail to reject* the null hypothesis that our model is sufficient. Therefore, we would hope to see a value in the ANOVA table for F^* that is relatively small or where the $Prob > F$ is larger than the desired significance level.

From Table 25 we see that this metamodel is sufficient at the $\alpha = 0.1$ level and there is no need to expand the experiment to look for quadratic terms.

Table 25. Air Superiority Metamodel Lack of Fit

Lack of Fit				
Source	DF	Sum of Squares	Mean Square	F Ratio
Lack of Fit	8	0.40574500	0.050718	0.4766
Pure Error	3	0.31927500	0.106425	Prob>F
Total Error	11	0.72502000		0.8212

4.3.2 MOE #2: Days Needed to Halt FLOT (FLOT)

The parameter estimates for this MOE are listed in Table 26 and result in the following metamodel:

$$\begin{aligned} \hat{y} = & 10.681 - 0.01687 x_{ACS} * x_{BLUEMSG} + 0.01812 x_{JSTAR} + \\ & + 0.02687 x_{BLUEMSG} * x_{JSTAR} - 0.03812 x_{PRCPT} - 0.01562 x_{ACS} * x_{PRCPT} + \\ & + 0.01562 x_{BLUEMSG} * x_{PRCPT} + 0.01562 x_{JSTAR} * x_{PRCPT} - 0.02812 x_{ACS} * \\ & * x_{SREC} - 0.04562 x_{PRCPT} * x_{SREC} \end{aligned} \quad (29)$$

Table 26. FLOT Metamodel Parameter Estimates

Parameter Estimates						
Term	Estimate	Std Error	t Ratio	Prob> t	Lower 95%	Upper 95%
Intercept	10.681	0.00650	1640.9	<.0001	10.6665	10.6955
ACS*BLUEMSG	-0.01687	0.00728	-2.32	0.0429	-0.03309	-0.00066
JSTAR	0.01812	0.00728	2.49	0.0320	0.00191	0.03434
BLUEMSG*JSTAR	0.02687	0.00728	3.69	0.0042	0.01066	0.04309
PRCPT	-0.03812	0.00728	-5.24	0.0004	-0.05434	-0.02191
ACS*PRCPT	-0.01562	0.00728	-2.15	0.0573	-0.03184	0.00059
BLUEMSG*PRCPT	0.01562	0.00728	2.15	0.0573	-0.00059	0.03184
JSTAR*PRCPT	0.01562	0.00728	2.15	0.0573	-0.00059	0.03184
ACS*SREC	-0.02812	0.00728	-3.86	0.0031	-0.04434	-0.01191
PRCPT*SREC	-0.04562	0.00728	-6.27	<.0001	-0.06184	-0.02941

Reviewing the normality tests from Appendices G and H indicates the model appears adequate and the data is normally distributed. As was the case with air superiority, we see that the R^2 and R^2_{ADJ} for this model (Table 27) are very high indicating

this particular metamodel accounts for a majority of the variability in the observed responses.

Table 27. FLOT Metamodel Summary of Fit

Summary of Fit	
RSquare	0.923508
RSquare Adj	0.854666
Root Mean Square Error	0.02911
Mean of Response	10.681
Observations (or Sum Wgts)	20

A final check is required to see if investigation for quadratic effects is warranted.

Table 28 presents the results of the lack of fit test for model sufficiency. With such a high *Prob > F* value there is no indication there are quadratic effects unaccounted for by the metamodel.

Table 28. FLOT Metamodel Lack of Fit

Lack of Fit				
Source	DF	Sum of Squares	Mean Square	F Ratio
Lack of Fit	7	0.004198	0.000600	0.4209
Pure Error	3	0.0042750	0.001425	Prob>F
Total Error	10	0.008473		0.8441

4.3.3 MOE #3: Red TAHAIT Destroyed

The parameter estimates for this MOE are listed in Table 29 and result in the following metamodel:

$$\begin{aligned} \hat{y} = & 60931.88 - 185.879 x_{JSTAR} - 357.555 x_{PRCPT} + 310.7468 x_{SREC} + \\ & + 103.7181 x_{JSTAR} * x_{SREC} + 46.7831 x_{BLUEMSG} * x_{JSTAR} \end{aligned} \quad (30)$$

Table 29. Red TAHAIT* Metamodel Parameter Estimates

Parameter Estimates						
Term	Estimate	Std Error	t Ratio	Prob> t	Lower 95%	Upper 95%
Intercept	60931.88	23.052	2643.2	<.0001	60882.438	60981.322
JSTAR	-185.879	25.773	-7.21	<.0001	-241.1569	-130.6019
PRCPT	-357.555	25.773	-13.87	<.0001	-412.8331	-302.2781
SREC	310.7468	25.773	12.06	<.0001	255.4693	366.0244
JSTAR*SREC	103.7181	25.773	4.02	0.0013	48.44061	158.9956
BLUEMSG*JSTAR	46.7831	25.773	1.82	0.0910	-8.49438	102.0606

*TAHAIT = Tank, Armored Personnel Carriers, Helicopters, Artillery, Infantry, Trucks.

Analyzing the normality tests for this MOE indicates the model appears adequate and the data is normally distributed. As was the case with the previous two MOEs, we see that the R^2 and R^2_{ADJ} for this model (Table 30) are very high indicating this particular metamodel accounts for a majority of the variability in the observed responses.

Table 30. Red TAHAIT Metamodel Summary of Fit

Summary of Fit	
RSquare	0.96693
RSquare Adj	0.955119
Root Mean Square Error	103.0921
Mean of Response	60931.88
Observations (or Sum Wgts)	20

A final check for quadratic effects is now required. Table 31 presents the results of the lack of fit test for model sufficiency. The $Prob > F$ value, while not as high as had been seen with the other MOEs, still indicates a linear fit is adequate and there is no need to investigate for quadratic effects. As tempting as it may be to try and explain this relatively low F value, the methodology used for our sensitivity analysis given in Figure 24

shows that since there isn't serious lack of fit and there is sufficient precision, we can proceed to the validation stage for this metamodel.

Table 31. Red TAHAIT Metamodel Lack of Fit

Lack of Fit				
Source	DF	Sum of Squares	Mean Square	F Ratio
Lack of Fit	11	137803.05	12527.6	3.4201
Pure Error	3	10988.70	3662.9	Prob>F
Total Error	14	148791.75		0.1699

4.3.4 MOE #4: Exchange Ratio

The parameter estimates for this MOE are listed in Table 32 and result in the following metamodel:

$$\hat{y} = 1.12095 - 0.02206 x_{ACS} - 0.00794 x_{JSTAR} - 0.0088 x_{PRCPT} + 0.0056 x_{BLUEMSG} * x_{PRCPT} - 0.0124 x_{SREC} \quad (31)$$

Table 32. Exchange Ratio Metamodel Parameter Estimates

Parameter Estimates						
Term	Estimate	Std Error	t Ratio	Prob> t	Lower 95%	Upper 95%
Intercept	1.12095	0.0022	502.36	<.0001	1.116	1.12573
ACS	0.02206	0.0025	8.84	<.0001	0.0167	0.02741
JSTAR	-0.00794	0.0025	-3.18	0.0067	-0.013	-0.0026
PRCPT	-0.0088	0.0025	-3.53	0.0033	-0.014	-0.0035
BLUEMSG*PRCPT	0.0056	0.0025	2.23	0.0427	0.0002	0.0109
SREC	-0.0124	0.0025	-4.99	0.0002	-0.018	-0.007

Reviewing the normality tests from Appendices G and H indicates the model appears adequate and the data is normally distributed. As has been the case throughout this experiment, we see that the R^2 and R^2_{ADJ} for this model (Table 33) are high indicating

this particular metamodel accounts for a good deal of the variability in the observed responses.

Table 33. Exchange Ratio Metamodel Summary of Fit

Summary of Fit	
RSquare	0.903205
RSquare Adj	0.868635
Root Mean Square Error	0.009979
Mean of Response	1.12095
Observations (or Sum Wgts)	20

A final check is required to see if investigation for quadratic effects is warranted.

Table 34 presents the results of the lack of fit test for model sufficiency. As was the case with Red TAHAIT, the *Prob > F* value indicates there is no serious lack of fit at the $\alpha = 0.05$ level, but the strength of the statement is less than what it was for FLOT or air superiority.

Table 34. Exchange Ratio Metamodel Lack of Fit

Lack of Fit				
Source	DF	Sum of Squares	Mean Square	F Ratio
Lack of Fit	11	0.00125814	0.000114	2.5230
Pure Error	3	0.00013600	0.000045	Prob>F
Total Error	14	0.00139414		0.2417

4.4 Analysis of Results

Multiple regression analysis was performed on each MOE. Significant variables were selected and regression accomplished with those variables. Metamodel adequacy and sufficiency were then addressed with the statistical measures discussed in Chapter 2

and using the methodology discussed in Chapter 3. The results of this analysis are consolidated in Table 35.

Table 35. MOE Response Surface Analysis

Response Surface Analysis				
Term	AIRSUP	FLOT	TAHAIT*	XRATIO
Intercept	28.7255	10.681	60931.88	1.12095
ACS	-0.8025			0.02206
JSTAR	-0.18125	0.01812	-185.879	-0.00794
BLUEMSG	0.14375			
SREC	-0.3125		310.7468	-0.0124
PRCPT	0.3175	-0.03812	-357.555	-0.0088
ACS*PRCPT		-0.01562		
ACS*BLUEMSG	0.15125	-0.01687		
BLUEMSG*PRCPT		0.01562		0.0056
BLUEMSG*JSTAR		0.02687	46.7831	
ACS*PRCPT		-0.01562		
JSTAR*PRCPT		0.01562		
JSTAR*SREC			103.7181	
ACS*SREC		-0.02812		
PRCPT*SREC	-0.235	-0.04562		
RSquare	0.95752	0.923508	0.96693	0.903205
RSquare Adj	0.926625	0.854666	0.955119	0.868635
Root Mean Square Error	0.256731	0.02911	103.0921	0.009979
Mean of Response	28.7255	10.681	60931.88	1.12095
Lack of Fit -- Prob>F	0.8212	0.8441	0.1699	0.2417

*TAHAIT = Tank, Armored Personnel Carriers, Helicopters, Artillery, Infantry, Trucks.

4.4.1 MOE #1: Days Needed to Achieve Air Superiority

Intuitively we expected to see a significant impact from the ACS variable, since this variable represents the status of red and blue airborne early warning systems: AWACS and Mainstay aircraft. This insight comes from experience during the Gulf War which showed that having an AWACS on-station has a synergistic effect on the outcome

of the air war. However, the metamodel for this MOE only shows a reduction in the number of days to achieve air superiority of 2.79% for the ACS parameter.

4.4.2 MOE #2: Days Needed to Halt FLOT (FLOT)

In the development of the design levels for message capacity (red and blue) and C3DEG it was felt that these would act as *big knobs*, significantly influencing this MOE. After all, these variables directly affect the ground war. However, REDMSG and C3DEG did not make the cut after the Plackett-Burman screening experiment. Then in the Resolution V experiment, BLUEMSG was only significant in combination with other effects as an interaction term. It was also assumed *a priori* that JSTARS would have an impact on the outcome of the ground war. Statistically, this turned out to be true. However, the impact proved to be a 1.7% *increase* in the time needed to halt the FLOT. Not what would be expected based upon the performance of JSTARS during the Gulf War.

4.4.3 MOE #3: Red TAHAIT Destroyed

The metamodel for this MOE also exhibited some surprising results. The parameter estimates for JSTAR and PRCPT were - 185.879 and - 357.555 respectively. Both of these estimates indicate we have a reduced ability to destroy Red targets. Contrary to these estimates was the estimate for SREC at 310.7468. This value at least makes sense. Increase standoff reconnaissance effects and our ability to inflict damage on Red is increased. However, the counterintuitive result is that JSTAR had a negative impact on this MOE and yet the JSTARS are the aircraft performing SREC. In addition,

the interaction term between JSTAR and SREC had a positive impact on TAHAIT.

Clearly, something is going on with THUNDER and its interaction with these variables that warrants further investigation. Application of this methodology to a more realistic classified database is warranted to check for consistency in parameter estimates.

4.4.4 MOE #4: Exchange Ratio

With this MOE, as was the case with air superiority, it was assumed *a priori* that the ability to provide airborne guidance to aircraft as modeled by ACS would have a significant impact upon the resultant exchange ratio. However, this did not prove to be the case. An improvement in this MOE would be a reduction in its value (i.e., we see less Blue and more Red aircraft dead). We see by the parameter estimate for ACS though, that we actually realize a 2 % increase in this measure.

4.5 Metamodel Validation

Two measures often used to assess the predictive ability of a proposed metamodel are the *MSPR* and *MAPE*. The former giving an expectation for the amount of variance to be expected when using the metamodel and the latter expresses by percentage the average error when using the metamodel. Recall from chapter 2 the equations for MSPR and MAPE:

$$MSPR = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n} \quad MAPE = \frac{100}{n} * \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{|y_i|} \quad (32)$$

According to Neter et al., if the MSPR is fairly close to the model's MSE then the regression equation is not seriously biased and gives an appropriate indication of the

predictive ability of the model (Neter, 1996: 436). To provide an unbiased evaluation of the metamodel's predictive ability we first put both the MSPR and the metamodel's root MSE (RMSE) on the same playing field. Since we are replicating THUNDER 30 times for each design point, the target parameter used in this investigation is the MOE's true, but unknown mean, μ . Assuming our estimator is unbiased, then RMSE is equal to $(\sigma^2/n)^{1/2}$. Therefore, for comparison to RMSE we will use $RMSPR = (MSPR/n)^{1/2}$ as the measure of our metamodel's predictive ability.

A metamodel is useful to a decision-maker if it can accurately predict outcomes of the simulation model throughout the appropriate design space (see Figure 26). As stated in Chapter 3 we performed an additional Plackett-Burman experiment at half-levels. We did not want to waste any THUNDER data. The validation experiment performed at the half-levels serves to assess the predictive ability of the metamodels within the design space and the data obtained from the screening experiment serves to check adequacy of the metamodel at the perimeter of the design space.

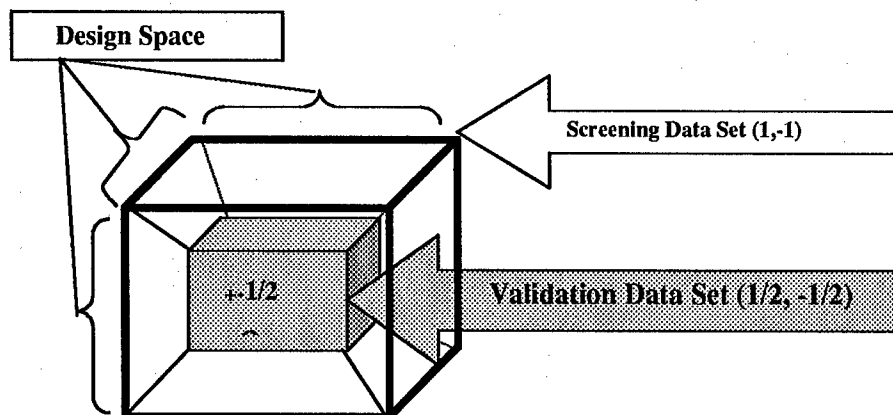


Figure 26. Validation of Metamodel Throughout Design Space

A comparison of the metamodel's prediction to THUNDER's output from the screening experiment is shown in Table 36. A similar comparison for the data obtained from the validation experiment is shown in Table 37. Calculations for RMSPR and MAPE for both the screening data set and validation data set are found in Appendix D.

Table 36. Plackett-Burman Screening Data vs. Metamodel Comparison

AIRSUP		FLOT		TAHAIT		XRATIO	
THUNDER	Metamodel	THUNDER	Metamodel	THUNDER	Metamodel	THUNDER	Metamodel
26.82	27.81	10.57	10.61	60816.58	60849.69	1.14	1.08
29.27	29.11	10.66	10.60	60291.21	60642.31	1.12	1.16
28.84	28.94	11.03	10.81	61304.15	61564.80	1.12	1.13
27.74	27.21	10.59	10.73	60390.88	60642.31	1.15	1.12
29.73	30.58	10.76	10.64	60475.27	60599.96	1.10	1.15
29.90	30.20	10.72	10.77	60509.88	60020.76	1.08	1.14
29.00	29.49	10.68	10.60	61349.88	61014.02	1.10	1.12
27.41	27.41	10.94	10.81	61659.73	61729.13	1.19	1.11
26.97	27.22	10.64	10.56	60998.85	60756.13	1.11	1.06
27.45	28.16	10.59	10.66	61448.94	61221.50	1.15	1.12
28.44	29.31	10.83	10.75	62176.33	61635.56	1.12	1.14
28.80	29.26	10.75	10.64	60955.90	60506.39	1.03	1.12

Table 37. Plackett-Burman Validation Data vs. Metamodel Comparison

AIRSUP		FLOT		TAHAIT		XRATIO	
THUNDER	Metamodel	THUNDER	Metamodel	THUNDER	Metamodel	THUNDER	Metamodel
27.19	28.32	10.67	10.65	60613.94	60871.60	1.12	1.10
29.23	28.97	10.80	10.68	60746.18	60868.24	1.13	1.14
28.55	28.84	10.67	10.72	61013.1	61229.15	1.13	1.13
27.73	28.09	10.58	10.71	60516.88	60868.24	1.12	1.12
29.20	29.59	10.68	10.65	60599.77	60771.82	1.12	1.14
29.33	29.47	10.66	10.69	59989.65	60526.28	1.12	1.13
28.67	29.16	10.68	10.65	61265.03	61013.42	1.11	1.13
27.90	28.08	10.59	10.72	61558.62	61370.97	1.15	1.11
27.50	28.10	10.58	10.64	60479.31	60856.01	1.12	1.10
28.23	28.49	10.73	10.68	61144	61098.18	1.14	1.12
28.23	29.03	10.78	10.70	61564.22	61339.78	1.11	1.13
28.73	28.93	10.58	10.65	60565.88	60740.63	1.15	1.12

At this point we have to ask the question, "Are going to fish or cut bait?" Is the RMSPR fairly close to the model's RMSE? If the metamodel cannot predict the model's

outcome, we need a new metamodel. Table 38 shows the RMSPR and MAPE for the screening data set. The RMSPR results indicates the metamodel can be used with confidence to predict THUNDER's performance at the perimeter of the design space. In addition, as you can see from the MAPE the metamodel is off by no more than 4 % for all of the MOEs.

Table 38. Comparison of RMSE, MSPR and MAPE for Plackett-Burman Screening Experiment Data Set

THUNDER MOE	RMSE	RMSPR	MAPE
AIRSUP	0.257	0.242	1.68
FLOT	0.029	0.05	0.91
TAHAIT	103.092	136.97	0.46
XRATIO	0.00998	0.02	4.08

Table 39 shows the RMSPR and MAPE for the validation data set. Once again, the RMSPR results indicate the metamodel can be used with confidence to predict THUNDER's performance within the design space. Also, the MAPE indicates the metamodel is off by no more than 2% for all of the MOEs.

Table 39. Comparison of RMSE, MSPR and MAPE for Plackett-Burman Validation Experiment Data Set

THUNDER MOE	RMSE	RMSPR	MAPE
AIRSUP	0.257	0.217	1.51
FLOT	0.029	0.034	0.64
TAHAIT	103.092	116.524	0.4005
XRATIO	0.010	0.009	1.68

Based upon the results evident from the previous two tables there is every indication to believe the metamodels obtained are an adequate representation of the simulation model.

5. Conclusions and Recommendations

This research had two objectives. The first was to develop a methodology to demonstrate the parallel processing capability provided by ASC's MSRC and application of said methodology to the SIMAF Proof of Concept project. Our second objective was to provide a sensitivity analysis of THUNDER to the modeled C2 inputs. This chapter provides an executive level summary of conclusions resulting from this thesis effort. Recommendations for possible future research are also.

5.1 Conclusions

5.1.1 Parallelization

MMMP, the script written to process THUNDER replications, provides the analyst with a *launch and leave* capability. With a comparatively minimal amount of preparatory work, MMMP takes the burden of performing all of the necessary grunt work on its *back*, leaving the analyst time to actually perform the analysis rather than minding the store.

The power of parallelization cannot be overemphasized. The time required to accomplish a single replication of THUNDER is approximately 45 minutes. For this thesis we made approximately 1,560 runs of THUNDER (an initial Plackett-Burman experiment had to be discarded) in slightly less than **3 days**. To accomplish the same number of runs on a single CPU machine would require slightly more than **3 months**.

As discussed in the previous section, during the Resolution V experiment, 600 THUNDER runs were launched by MMMP at the MSRC in less than one day. Unfortunately, at the time there was a need to retain the temporary run directories to examine an apparent peculiarity with the operation of THUNDER. At this point, what was peculiar about THUNDER is not important, but rather what happened by failing to use MMMP's default option of eliminating the temporary directories is worth noting. In all, these directories grew in size to approximately 75 gigabytes, shutting down the NFS where MMMP was operating.

It must be emphasized that THUNDER generates mountains of output, enough to fill even seemingly *unlimited* storage areas. If retaining the temporary directories is required for some reason, then pay close attention should to disk utilization or a concerned system administrator may give you a call.

5.1.2 C2 Sensitivity Analysis

This thesis provided an exploratory investigation into the sensitivity of THUNDER to the modeled C2 inputs. The five-step *mountain climbing* methodology presented in Figure 5, discussed and demonstrated by this thesis provided an efficient approach to analyzing the sensitivity of THUNDER to C2 parameters. This same methodology applies to any simulation model with any set of input parameters.

The use of a Plackett-Burman Resolution III experimental design allowed us to identify which of the 11 input variables actually had a statistical impact upon THUNDER. The decision to investigate only the significant variables reduced the number of input variables from 11 to 5. This reduced the number of design points necessary to obtain the

same Resolution V information from 128 to 16 and eliminated the need for 3,360

THUNDER runs. A significant savings!

A rather surprising result of this research was that all four MOEs measured were relatively insensitive to the perturbation of the C2 variables. In as much as we would intuitively expect to see an impact, perhaps even a statistically significant impact, on the final outcome by perturbing C2, the findings of this thesis can be explained by one of three possibilities. The first is the metamodels developed were faulty and could not accurately predict the outcome of THUNDER for these measures. The second possibility is the unclassified database used for this thesis was too far removed from reality to provide an accurate assessment of THUNDER's sensitivities. Finally, THUNDER does not accurately model the C2 process.

Since the metamodels were rigorously validated the first possibility is unlikely. Therefore, the explanation would most likely be found somewhere within the final two possibilities. However, before anything can be said about THUNDER's ability to model C2, the second possibility should be explored and this process repeated on a classified database.

5.2 Recommendations for Further Study

Work on this thesis has suggested several avenues for explaining some of the counterintuitive results uncovered during this investigation. First, the same methodology could be employed on a classified database. If this topic is undertaken, consideration might be given to increasing the variable design space since it was difficult to determine if the results of this study were driven by the unclassified database or a restricted variable

domain. Also, a larger set of MOEs would allow a multivariate look at the response of THUNDER to the modeled input parameters thereby providing more insight into the underlying factors that ultimately drive the model.

As pointed out by Webb, there is no specific guidance available to the analyst to aid in the selection of output measures (Webb 1994: 5-5). For theater-level campaign models the selection of input variables is necessarily model dependent. However, cross-model commonality in the form of output metric standardization is one area which would be of significant benefit to the simulation modeling community. The UJTL has certainly endeavored to answer this need, but has provided too much for too many. An analysis framework similar to the checklist developed by the ASPVG, decomposing CINC theater-level campaign objectives down into model-measurable operational capabilities would tremendously enhance cross-model verification and validation efforts. Research to develop this analytical framework could benefit the entire simulation community.

One final area might prove to be both interesting and very rewarding. For this thesis the THUNDER runs were accomplished independently with different random number streams between replications. THUNDER though, has the provision for allowing different random number streams to be assigned to different events. This coupled with the fact that MMMP has the capability of ensuring each replication is begun with the same set of random number streams suggests research into whether or not common random numbers could reduce the variance of a large scale stochastic model.

Appendix A. THUNDER Data Files

SREC Maximum Effect Multiplier Design Tables Screening Design

Table 40. SREC Multiplier – Low Level

WEAPON	TARGET	BADD	BADH	BADI	STATIC	RADI	RADH	RADD
10001	10001	0.975	0.975	0.975	0.825	0.825	0.900	0.975
	10002	0.975	0.975	0.975	0.825	0.825	0.900	0.975
	10003	0.975	0.975	0.975	0.825	0.825	0.900	0.975
	10004	0.975	0.975	0.975	0.825	0.825	0.900	0.975
10002	10001	0.825	0.825	0.825	0.825	0.825	0.825	0.825
	10002	0.825	0.825	0.825	0.825	0.825	0.825	0.825
	10003	0.825	0.825	0.825	0.825	0.825	0.825	0.825
	10004	0.825	0.825	0.825	0.825	0.825	0.825	0.825
10003	10001	0.975	0.975	0.975	0.825	0.825	0.900	0.975
	10002	0.975	0.975	0.975	0.825	0.825	0.900	0.975
	10003	0.975	0.975	0.975	0.825	0.825	0.900	0.975
	10004	0.975	0.975	0.975	0.825	0.825	0.900	0.975
10004	10001	0.975	0.975	0.975	0.825	0.825	0.900	0.975
	10002	0.975	0.975	0.975	0.825	0.825	0.900	0.975
	10003	0.975	0.975	0.975	0.825	0.825	0.900	0.975
	10004	0.975	0.975	0.975	0.825	0.825	0.900	0.975

Table 41. SREC Multiplier – Baseline

WEAPON	TARGET	BADD	BADH	BADI	STATIC	RADI	RADH	RADD
10001	10001	1.30	1.30	1.30	1.10	1.10	1.20	1.30
	10002	1.30	1.30	1.30	1.10	1.10	1.20	1.30
	10003	1.30	1.30	1.30	1.10	1.10	1.20	1.30
	10004	1.30	1.30	1.30	1.10	1.10	1.20	1.30
10002	10001	1.10	1.10	1.10	1.10	1.10	1.10	1.10
	10002	1.10	1.10	1.10	1.10	1.10	1.10	1.10
	10003	1.10	1.10	1.10	1.10	1.10	1.10	1.10
	10004	1.10	1.10	1.10	1.10	1.10	1.10	1.10
10003	10001	1.30	1.30	1.30	1.10	1.10	1.20	1.30
	10002	1.30	1.30	1.30	1.10	1.10	1.20	1.30
	10003	1.30	1.30	1.30	1.10	1.10	1.20	1.30
	10004	1.30	1.30	1.30	1.10	1.10	1.20	1.30
10004	10001	1.30	1.30	1.30	1.10	1.10	1.20	1.30
	10002	1.30	1.30	1.30	1.10	1.10	1.20	1.30
	10003	1.30	1.30	1.30	1.10	1.10	1.20	1.30
	10004	1.30	1.30	1.30	1.10	1.10	1.20	1.30

Table 42. SREC Multiplier – High Level

WEAPON	TARGET	BADD	BADH	BADI	STATIC	RADI	RADH	RADD
10001	10001	1.625	1.625	1.625	1.375	1.375	1.500	1.625
	10002	1.625	1.625	1.625	1.375	1.375	1.500	1.625
	10003	1.625	1.625	1.625	1.375	1.375	1.500	1.625
	10004	1.625	1.625	1.625	1.375	1.375	1.500	1.625
10002	10001	1.375	1.375	1.375	1.375	1.375	1.375	1.375
	10002	1.375	1.375	1.375	1.375	1.375	1.375	1.375
	10003	1.375	1.375	1.375	1.375	1.375	1.375	1.375
	10004	1.375	1.375	1.375	1.375	1.375	1.375	1.375
10003	10001	1.625	1.625	1.625	1.375	1.375	1.500	1.625
	10002	1.625	1.625	1.625	1.375	1.375	1.500	1.625
	10003	1.625	1.625	1.625	1.375	1.375	1.500	1.625
	10004	1.625	1.625	1.625	1.375	1.375	1.500	1.625
10004	10001	1.625	1.625	1.625	1.375	1.375	1.500	1.625
	10002	1.625	1.625	1.625	1.375	1.375	1.500	1.625
	10003	1.625	1.625	1.625	1.375	1.375	1.500	1.625
	10004	1.625	1.625	1.625	1.375	1.375	1.500	1.625

NUMBER.OF.C3.TYPES 6

1 "BLUE UNIT C3 FAC"

SIDE..MSG.CAP (MSG/HR) ..TGT.SEP.FN

1 150 20001

MESSAGE.DEMAND.BY.POSTURE

.BADD....BADH....BADI....STATIC..RADI....RADH....RADD
100 90 80 50 60 80 100

Figure 27.. C3 Message Capacity -- *typec3.dat*

```

1098 "JSTARS E-8"
DAY.IN.THEATER..AUTH.QTY.SORT/DAY..AC.MAX.SORT/DAY
    1.00          1.00          1.50
END.PROFILE

10981 "JSTARS"
SIDE..SUP.CMD.ID..TYPE.AC.ID..AUTH.QTY..AR.PRIORITY...RECCE.PRIORITY
    1      1200      1098      1      1      0
MOB.ID..DISP.AB.ID..SERV.KIT.ID..SORT.PROF.ID..MISSION.CLASS
    1029      1020      1098      1098      JSTARS
..DCA..ODCA..HVAA..BARC..FSWP..EAIR...STI...CAS...BAI...INT...OCA..OTBM..DTBM
    0      0      0      0      0      0      0      0      0      0      0      0
.DSED..SSUP..CSUP..ESUP..SJAM..CJAM..EJAM..RECC..SREC...AEW...AAR..LIFT..XXXX..RESV
    0      0      0      0      0      0      0      0      0      0      0      0      100

1003 "AWACS E-3"
DAY.IN.THEATER..AUTH.QTY.SORT/DAY..AC.MAX.SORT/DAY
    1.00          1.00          1.50
END.PROFILE

10301 "AWACS"
SIDE..SUP.CMD.ID..TYPE.AC.ID..AUTH.QTY..AR.PRIORITY...RECCE.PRIORITY
    1      1200      1003      1      1      0
MOB.ID..DISP.AB.ID..SERV.KIT.ID..SORT.PROF.ID..MISSION.CLASS
    1029      1020      1003      1003      AWACS
..DCA..ODCA..HVAA..BARC..FSWP..EAIR...STI...CAS...BAI...INT...OCA..OTBM..DTBM
    0      0      0      0      0      0      0      0      0      0      0      0
.DSED..SSUP..CSUP..ESUP..SJAM..CJAM..EJAM..RECC..SREC...AEW...AAR..LIFT..XXXX..RESV
    0      0      0      0      0      0      0      0      0      70      0      0      100

20000 "MAINSTAY"
SIDE..SUP.CMD.ID..TYPE.AC.ID..AUTH.QTY..AR.PRIORITY...RECCE.PRIORITY
    2      2200      2006      0      0      0
MOB.ID..DISP.AB.ID..SERV.KIT.ID..SORT.PROF.ID..MISSION.CLASS
    2014      2013      2006      2006      AWACS
..DCA..ODCA..HVAA..BARC..FSWP..EAIR...STI...CAS...BAI...INT...OCA..OTBM..DTBM
    0      0      0      0      0      0      0      0      0      0      0      0
.DSED..SSUP..CSUP..ESUP..SJAM..CJAM..EJAM..RECC..SREC...AEW...AAR..LIFT..XXXX..RESV
    0      0      0      0      0      0      0      0      0      70      0      0      100

```

Figure 28. *squadron.dat* excerpt

```

@   These IADS variables have been divided into two
groups,
@   one involving comms ability, and the other involving SAM
There are
@   firing ability. 5 possible comms ability states:
@
@ State      Num SOCs      Num IOCs      Jammers Present
@ 1          0             0             N/A
@ 2          >0            >0             N
@ 3          >0            0             N
@ or         0             >0             N
@ 4          >0            0             Y
@ or         0             >0             Y
@ 5          >0            >0             Y
@
@   There are six possible firing ability states:
@
@ State      Surviving      Surviving
@ State      EW            Acq            Weasels Present
@ 1          >50%          >50%          N
@ 2          >50%          >50%          Y
@ 3          >50%          <50%          N
@ or         <50%          >50%          N
@ 4          >50%          <50%          Y
@ or         <50%          >50%          Y
@ 5          <50%          <50%          N
@ 6          <50%          <50%          Y
@
@
@ Probability of Integration Table
@
@ Comm Ability      Firing Ability State
@ State      1      2      3      4      5      6
@-----|-----
@ 1      | 0.0    0.0    0.0    0.0    0.0    0.0
@ 2      | 1.0    .50    .75    .35    .50    .20
@ 3      | .50    .25    .40    .20    .25    .10
@ 4      | .25    .10    .20    .10    .10    .05
@ 5      | .75    .40    .50    .30    .40    .15

```

Figure 29. Comm/Firing Ability State Calculation -- *adsector.dat*

Appendix B. Campaign Objectives

CO # 1. HALT INVADING ARMIES

OO # 1.1 Delay/destroy/disrupt lead elements of armored advance

OT # 1.1.1 Destroy/damage advancing armored vehicles

OC # 1.1.1.1 Percent of tanks killed

OC # 1.1.1.2 Percent APCs killed

OC # 1.1.1.3 Time to stop RED advancement

OC # 1.1.1.4 Distance FLOT moved

OT # 1.1.2 Destroy/damage accompanying support vehicles

OC # 1.1.2.1 Percent trucks killed

OT # 1.1.3 Mine/cut key attack routes

OC # 1.1.3.1 Number of choke points killed

OC # 1.1.3.2 Number of choke points repaired

OC # 1.1.3.3 Percent potential RED attack routes closed

OO # 1.2 Delay/damage reinforcing forces and supplies in the rear

OT # 1.2.1 Mine/cut roads and railbeds

OC # 1.2.1.1 Number of choke points killed

OC # 1.2.1.2 Number of rail bridges killed

OC # 1.2.1.3 Number of trans shipment points killed

OT # 1.2.2 Destroy/damage armored and other vehicles in convoys or on trains

OC # 1.2.2.1 Total number of moving tanks killed

OC # 1.2.2.2 Total number of moving APCs killed

OC # 1.2.2.3 Total number of re-supply tanks killed

OC # 1.2.2.4 Total number of re-supply APCs killed

OT # 1.2.3 Disrupt field logistics sites, transportation nodes, assembly areas

OC # 1.2.3.1 Number of logistics sites destroyed

OC # 1.2.3.2 Number of assembly nodes killed

OT # 1.2.4 Drop bridges, block tunnels and other choke points

OC # 1.2.4.1 Number of bridges destroyed

OC # 1.2.4.2 Number of tunnels destroyed

OO # 1.3 Provide fire support to forces in close contact with enemy ground forces

OT # 1.3.1 Destroy/disable/pin armored vehicles near line of contact

OC # 1.3.1.1 Percent of tanks near line of contact disabled/killed

- OC # 1.3.1.2 Percent of APCs near line of contact disabled/killed
- OT # 1.3.2 Disable/pin dismounted troops near line of contact
 - OC # 1.3.2.1 Percent of infantry near line of contact killed
 - OC # 1.3.2.2 Percent of infantry near line of contact wounded in action
- OT # 1.3.3 Destroy/suppress artillery and multiple rocket launchers
 - OC # 1.3.3.1 Percent of self-propelled artillery killed
 - OC # 1.3.3.2 Percent of towed artillery killed
 - OC # 1.3.3.3 Percent of MLRS killed

CO # 2. MARSHAL AND SUSTAIN IN-THEATER ASSETS

OO # 2.1 Airlift personnel and materiel into and within distant theaters

- OT # 2.1.1 Airlift forces and critical support into distant theater
 - OC # 2.1.1.1 Percent TPFDD arrived in theater on time
- OT # 2.1.2 Airlift forces and critical support within theater
 - OC # 2.1.2.1 Total troops moved in theater
 - OC # 2.1.2.2 Total cargo moved in theater
- OT # 2.1.3 Airdrop troops and equipment covertly in hostile territory
 - OC # 2.1.3.1 Percent of SOF missions successfully inserted

OO # 2.2 Refuel aircraft in flight

- OT # 2.2.1 Refuel aircraft flying to and from distant theaters
 - OC # 2.2.1.1 Total number of aircraft refueled
 - OC # 2.2.1.2 Percent of total scheduled receivers refueled
- OT # 2.2.2 Refuel aircraft moving to attack enemy forces
 - OC # 2.2.2.1 Total number of aircraft refueled
 - OC # 2.2.2.2 Percent of total scheduled receivers refueled
- OT # 2.2.3 Refuel aircraft on station or CAP
 - OC # 2.2.3.1 Total number of CAP aircraft refueled
 - OC # 2.2.3.2 Percent of total scheduled CAP receivers refueled
 - OC # 2.2.3.3 Average time to refuel once on station

OO # 2.3 Recover personnel in distress

- OT # 2.3.1 Rescue downed aircrews and other personnel in hostile territory
 - OC # 2.3.1.1 Percent of downed aircrew recovered
 - OC # 2.3.1.2 Success rate of recovering special ops teams
- OT # 2.3.2 Medevac wounded personnel to medical facilities
 - OC # 2.3.2.1 Total number of patients medevaced

OO # 2.4 Train and maintain in-theater force elements

OT # 2.4.1 Maintain equipment for high-intensity operations (generate sorties)

OC # 2.4.1.1 BLUE mean repair time (MRT) by platform\

OC # 2.4.1.2 BLUE aircraft battle damage repair time (ABORT)

OC # 2.4.1.3 Average BLUE mission preparation time

OC # 2.4.1.4 Average weapon upload/checkout time

OC # 2.4.1.5 Total sorties generated by BLUE

OC # 2.4.1.6 Total number of mission aborts for maintenance

OT # 2.4.2 Train and exercise personnel

OC #

OT # 2.4.3 Provide for the morale and welfare of personnel

OC #

OO # 2.5 Secure bases

OT # 2.5.1 Secure base perimeters

OC #

OT # 2.5.2 Defeat attacks by special ops forces

OC #

CO # 3. EVICT HALTED ARMIES FROM FRIENDLY TERRITORY

OO # 3.1 Degrade and overrun defensive positions

OT # 3.1.1 Destroy/damage armored and other vehicles in defensive positions

OC # 3.1.1.1 Number of Tanks in defensive positions killed

OC # 3.1.1.2 Number of APCs in defensive positions killed

OT # 3.1.2 Disable dismounted troops

OC # 3.1.2.1 Number of infantry in defensive positions killed

OC # 3.1.2.2 Number of infantry in defensive positions wounded in action

OT # 3.1.3 Neutralize obstacles (mines, fortifications)

OC #

OT # 3.1.4 Mine/cut key routes of retreat

OC # 3.1.4.1 Percentage of key routes of retreats blocked

OO # 3.2 Delay/damage reinforcing forces and supplies in the rear (same as I.B above)

OT # 3.2.1 Mine/cut roads and railbeds

OC # 3.2.1.1 Number of choke points killed

OC # 3.2.1.2 Number of rail bridges killed

OC # 3.2.1.3 Number of trans shipment points killed

OT # 3.2.2 Destroy/damage armored and other vehicles in convoys or on trains

OC # 3.2.2.1 Total number of moving tanks killed

OC # 3.2.2.2 Total number of moving APCs killed

OC # 3.2.2.3 Total number of re-supply tanks killed

OC # 3.2.2.4 Total number of re-supply APCs killed

OT # 3.2.3 Disrupt field logistics sites, transportation nodes, assembly areas

OC # 3.2.3.1 Number of logistics sites destroyed

OC # 3.2.3.2 Number of assembly nodes killed

OT # 3.2.4 Drop bridges, block tunnels and other choke points

OC # 3.2.4.1 Number of bridges destroyed

OC # 3.2.4.2 Number of tunnels destroyed

OO # 3.3 Provide fire support to forces in close contact with enemy forces

OT # 3.3.1 Destroy/disable/pin armored vehicles near line of contact

OC # 3.3.1.1 Percent of tanks near line of contact disabled/killed

OC # 3.3.1.2 Percent of APCs near line of contact disabled/killed

OT # 3.3.2 Disable/pin dismounted troops near line of contact

OC # 3.3.2.1 Percent of infantry near line of contact killed

OC # 3.3.2.2 Percent of infantry near line of contact wounded in action

OT # 3.3.3 Destroy/suppress artillery and MLRS

OC # 3.3.3.1 Percent of self-propelled artillery killed

OC # 3.3.3.2 Percent of towed artillery killed

OC # 3.3.3.3 Percent of MLRS killed

CO # 4. GAIN, MAINTAIN AIR SUPERIORITY

OO # 4.1 Defeat air attacks

OT # 4.1.1 Destroy/disrupt aircraft and helicopters in flight

OC # 4.1.1.1 RED aircraft lost due to BLUE air

OC # 4.1.1.2 Percent of RED aircraft shot down

OC # 4.1.1.3 RED/BLUE exchange ratio

OT # 4.1.2 Destroy/disrupt cruise missiles in flight

OC # 4.1.2.1 Percent of cruise missile intercepted in flight

OT # 4.1.3 Disrupt sensors on aircraft and weapons

OC # 4.1.3.1 Losses with EW assets/Losses without EW assets ratio

OT # 4.1.4 Execute passive defense measures in threatened areas

OC # 4.1.4.1 Total number of BLUE aircraft destroyed on the ground

OO # 4.2 Suppress generation of air sorties

OT # 4.2.1 Crater/mine/damage airfield runways and taxiways

- OC # 4.2.1.1 Percent of RED airfield operable
- OC # 4.2.1.2 Percent time RED airfield is operable
- OT # 4.2.2 Destroy/damage aircraft in the open or in revetments
 - OC # 4.2.2.1 Number of RED aircraft destroyed in the open
 - OC # 4.2.2.2 Number of RED aircraft destroyed in revetments
 - OC # 4.2.2.3 Ground kills per total kills
- OT # 4.2.3 Destroy/damage aircraft in hardened shelters
 - OC # 4.2.3.1 Number of RED aircraft destroyed in hardened shelters
 - OC # 4.2.3.2 Number of RED hardened shelters destroyed
- OT # 4.2.4 Destroy/damage airbase support facilities
 - OC # 4.2.4.1 Percent of RED support facilities destroyed by BLUE air
- OT # 4.2.5 Deny attack helicopter forward area refuel/replenishment points (FARRP)
 - OC # 4.2.5.1 Percentage of RED FARPS destroyed by BLUE air
- OO # 4.3 Suppress surface-based air defenses**
 - OT # 4.3.1 Destroy/damage fixed SAM launchers
 - OC # 4.3.1.1 Number of TELs killed
 - OC # 4.3.1.2 Number of ACQ radars killed
 - OC # 4.3.1.3 Number of fire control radars killed
 - OC # 4.3.1.4 Total number of BLUE aircraft lost to enemy SAMs
 - OT # 4.3.2 Destroy/damage mobile SAM launchers and AAA
 - OC # 4.3.2.1 Number of TELARs killed
 - OC # 4.3.2.2 Number of mobile ACQ radars killed
 - OC # 4.3.2.3 Number of mobile fire control radars killed
 - OC # 4.3.2.4 Total number of BLUE aircraft lost to enemy SAMs
 - OT # 4.3.3 Destroy/disrupt tracking and engagement radars
 - OC # 4.3.3.1 Percent of EW/GCI sites operable
- OO # 4.4 Defeat attacking ballistic missiles**
 - OT # 4.4.1 Destroy ballistic missiles in flight (active defense)
 - OC # 4.4.1.1 Percent of BMs intercepted by Blue air
 - OT # 4.4.2 Execute passive defense measures in threatened areas
 - OC # 4.4.2.1 Total number of BLUE losses due to BM attack
- OO # 4.5 Suppress the generation of ballistic missile launches**
 - OT # 4.5.1 Damage/destroy TELs in the field and disrupt operations
 - OC # 4.5.1.1 Percent of BM TELs in field operable
 - OT # 4.5.2 Damage/destroy TELs in garrisons and assembly areas
 - OC # 4.5.2.1 Percent of TELs in garrison and assembly areas operable

OT # 4.5.3 Damage/destroy fixed TBM launchers

OC # 4.5.3.1 Percent of TBMs operable

OT # 4.5.4 Destroy TBM storage areas

OC # 4.5.4.1 Total number of TBM storage areas destroyed

CO # 5. GAIN, MAINTAIN SEA CONTROL

OO # 5.1 Sink/disable surface combatants and disrupt their operations

OT # 5.1.1 Sink/disable ships at sea and in port

OC # 5.1.1.1 Percent of surface combatants killed

OT # 5.1.2 Mine ports, choke points, and anchorages

OC # 5.1.2.1 Percent of port processing capacity remaining overall

OT # 5.1.3 Disrupt shipborne sensors

OC #

CO # 6. GAIN, MAINTAIN SPACE CONTROL

OO # 6.1 Sustain operations of friendly space-based assets

OT # 6.1.1 Redeploy space assets as needed and sustain constellations on orbit

OC #

OT # 6.1.2 Launch satellites on a timely basis

OC # 6.1.2.1 Average delay of BLUE satellite launches

OO # 6.2 Protect friendly space-based assets in the face of enemy attack

OT # 6.2.1 Destroy/disrupt ASATs in flight

OC # 6.2.1.1 Percent of RED ASATs destroyed in flight

OT # 6.2.2 Evade ASATs

OC # 6.2.2.1 Percent of BLUE satellites operable

OO # 6.3 Suppress enemy space-based capabilities

OT # 6.3.1 Destroy/damage satellites in orbit

OC # 6.3.1.1 Percent of satellites operable

OT # 6.3.2 Destroy/damage launch facilities, tracking stations, and other fixed sites

OC # 6.3.2.1 Percent of launch facilities, tracking stations and fixed sites operable

OT # 6.3.3 Destroy/damage mobile space surveillance and tracking radars

OC # 6.3.3.1 Percent of mobile space surveillance and tracking radars operable

OT # 6.3.4 Disrupt links

OC # 6.3.4.1 Percent of links operable

CO # 7. GAIN, MAINTAIN INFORMATION DOMINANCE

OO # 7.1 Provide "eyes, ears, and voice" of commanders

OT # 7.1.1 Provide timely, accurate information on enemy activities, force disposition

OC #

OT # 7.1.2 Provide timely, accurate assessment of battle results

OC #

OT # 7.1.3 Provide timely, accurate reports on friendly force disposition

OC #

OT # 7.1.4 Provide timely, accurate reports on the weather

OC #

OT # 7.1.5 Provide timely, accurate dissemination of commanders' intent

OC #

OO # 7.2 Degrade command and control of enemy forces

OT # 7.2.1 Destroy/damage command bunkers

OC # 7.2.1.1 Percent of command bunkers operable

OC # 7.2.1.2 Data transmission rate

OT # 7.2.2 Destroy/damage mobile command posts

OC # 7.2.2.1 Number of C3 antennae killed

OC # 7.2.2.2 Number of C3 vans killed

OC # 7.2.2.3 Data transmission rate

OT # 7.2.3 Disrupt enemy communications

OC # 7.2.3.1 Average communication data transmission rate

OT # 7.2.4 Destroy/disrupt airborne C2, and surveillance platforms

OC # 7.2.4.1 Number of mainstays killed

OC # 7.2.4.2 Average time mainstay is on station

OT # 7.2.5 Destroy/disrupt ground-based radars and other sensors

OC # 7.2.5.1 Percent of EW/GCI sites operable

OO # 7.3 Sow confusion in enemy situation awareness

OT # 7.3.1 Effect deceptions/false targets to mask friendly deployments and assets

OC # 7.3.1.1 Number of decoys killed

OT # 7.3.2 Disseminate disinformation to enemy commanders and forces

OC #

CO # 8. DENY POSSESSION AND USE OF WEAPONS OF MASS DESTRUCTION

OO # 8.1 Damage/deny facilities for producing and storing WMD

OT # 8.1.1 Destroy production plants

OC # 8.1.1.1 Percent of production plant destroyed

OT # 8.1.2 Destroy weapon storage sites or deny access

OC # 8.1.2.1 Percent of WMD storage facilities destroyed

OO # 8.2 Defeat attacking ballistic missiles

OT # 8.2.1 Destroy ballistic missiles in flight (active defense)

OC # 8.2.1.1 Percent of BMs intercepted by BLUE air

OT # 8.2.2 Execute passive defense measures in threatened areas

OC # 8.2.2.1 Total number of BLUE losses due to BM attack

OO # 8.3 Suppress the generation of ballistic missile launches

OT # 8.3.1 Destroy/damage TELs in the field and disrupt operations

OC # 8.3.1.1 Percent of BM TELs in field operable

OT # 8.3.2 Destroy/damage TELs in garrisons and assembly areas

OC # 8.3.2.1 Percent of TELs in garrison and assembly areas operable

OT # 8.3.3 Destroy/damage fixed TBM launchers

OC # 8.3.3.1 Percent of TBMs operable

OT # 8.3.4 Destroy Tactical Ballistic Missile (TBM) storage areas

OC # 8.3.4.1 Total number of TBM storage areas destroyed

OO # 8.4 Defeat air attacks

OT # 8.4.1 Destroy/disrupt aircraft and helicopters in flight

OC # 8.4.1.1 RED aircraft lost due to BLUE air

OC # 8.4.1.2 Percent of RED aircraft shot down

OC # 8.4.1.3 RED/BLUE exchange ratio

OT # 8.4.2 Destroy/disrupt cruise missiles in flight

OC # 8.4.2.1 Percent of cruise missiles intercepted in flight

OT # 8.4.3 Disrupt sensors on aircraft and weapons

OC # 8.4.3.1 Losses with EW assets/Losses without EW assets ratio

OT # 8.4.4 Execute passive defensive measures in threatened areas

OC # 8.4.4.1 Total number of BLUE aircraft destroyed on the ground

OO # 8.5 Suppress generation of air sorties

OT # 8.5.1 Crater/mine/damage airfield runways and taxiways

OC # 8.5.1.1 Percent of RED airfield operable

OC # 8.5.1.2 Percent time RED airfield is operable

OT # 8.5.2 Destroy/damage aircraft in the open or in revetments

OC # 8.5.2.1 Number of RED aircraft destroyed in the open

OC # 8.5.2.2 Number of RED aircraft destroyed in revetments

OC # 8.5.2.3 Ground kills per total kills

OT # 8.5.3 Destroy/damage aircraft in hardened shelters

OC # 8.5.3.1 Number of RED aircraft destroyed in hardened shelters

OC # 8.5.3.2 Number of RED hardened shelters destroyed

OT # 8.5.4 Destroy/damage airbase support facilities

OC # Percent of RED support facilities destroyed by BLUE air

OT # 8.5.5 Deny attack helicopter forward area refuel/replenishment points

OC # 8.5.5.1 Percentage of RED FARPS destroyed by BLUE air

CO # 9. SUPPRESS NATIONAL CAPACITY TO WAGE WAR

OO # 9.1 Disrupt national POL system

OT # 9.1.1 Disrupt/damage POL refineries and storage facilities

OC # 9.1.1.1 POL production rate

OC # 9.1.1.2 Percent POL storage facilities destroyed

OT # 9.1.2 Sever key petroleum pipelines

OC # 9.1.2.1 POL transfer rate

OC # 9.1.2.2 Percent pipelines destroyed

OT # 9.1.3 Disrupt off-load sites at ports and transshipment points

OC # 9.1.3.1 Percentage of RED transshipment points operable

OT # 9.1.4 Disrupt/damage POL control facilities

OC # 9.1.4.1 Percent POL control facilities operable

OO # 9.2 Disrupt national power generation system

OT # 9.2.1 Disrupt/disable power plants and hydroelectric facilities

OC # 9.2.1.1 Percent of power plants/hydroelectric facilities operable

OC # 9.2.1.2 Average total power output

OT # 9.2.2 Disrupt/disable substations and transformers

OC # 9.2.2.1 Number of substations killed

OC # 9.2.2.2 Number of trans shipment points killed

OT # 9.2.3 Sever power lines

- OC # 9.2.3.1 Number of power lines destroyed
- OT # 9.2.4 Disable/destroy alternative "stand-alone" power sources
 - OC # 9.2.4.1 Number of stand alone power sources
- OT # 9.2.5 Disrupt/disable grid control facilities
 - OC # 9.2.5.1 Number of grid control facilities killed
- OO # 9.3 Disrupt national communications system**
 - OT # 9.3.1 Disrupt/disable key telephone switching centers
 - OC # 9.3.1.1 Number of key telephone switching centers killed
 - OT # 9.3.2 Sever landlines
 - OC # 9.3.2.1 Number of land lines killed
 - OT # 9.3.3 Disrupt/damage key communications nodes
 - OC # 9.3.3.1 Number of fixed C2 nodes killed
 - OC # 9.3.3.2 Number of mobile C2 nodes killed
- OO # 9.4 Disrupt national transportation system**
 - OT # 9.4.1 Disrupt airports, seaports, and transshipment points
 - OC # 9.4.1.1 Percent of airports, seaports and transshipment points operable
 - OT # 9.4.2 Disrupt railroad marshaling yards
 - OC # 9.4.2.1 Percent of marshaling yards destroyed
 - OT # 9.4.3 Mine/cut roads, railroads, and waterways
 - OC # 9.4.3.1 Number of roads cut
 - OC # 9.4.3.2 Number of waterways destroyed
 - OC # 9.4.3.3 Number of railroads destroyed
 - OT # 9.4.4 Drop bridges and block choke points
 - OC # 9.4.4.1 Percent of bridges destroyed
 - OC # 9.4.4.2 Percent of choke points open
 - OT # 9.4.5 Disrupt/damage network control and navigation facilities
 - OC # 9.4.5.1 Number of network control facilities destroyed
 - OC # 9.4.5.2 Number of navigation facilities destroyed
- OO # 9.5 Damage/disrupt war-supporting industry**
 - OT # 9.5.1 Destroy defense-related plants and equipment
 - OC # 9.5.1.1 Percent of defense related plants operable
 - OC # 9.5.1.2 Percent reduction in farm production
 - OC # 9.5.1.3 Percent reduction in vehicle production
 - OC # 9.5.1.4 Percent reduction in metal production
 - OC # 9.5.1.5 Percent of research learning centers destroyed
 - OT # 9.5.2 Disrupt flow of war-supporting imports

OC # 9.5.2.1 Percent of inventory destroyed

OO # 9.6 Disrupt political direction of enemy society, economy, war effort

OT # 9.6.1 Destroy/damage key directing organs and leadership cadres

OC # 9.6.1.1 Number of offices destroyed

OT # 9.6.2 Destroy leadership and internal security facilities

OC # 9.6.2.1 Number of internal security facilities killed

OC # 9.6.2.2 Number of leaders killed

OT # 9.6.3 Disseminate disinformation among leadership and population

OC #

OO # 9.7 Reduce motivation of enemy troops to resist friendly action

OT # 9.7.1 Disseminate disinformation, warning of impending attacks

OC #

OT # 9.7.2 Create belief that operating combat equipment will bring harm

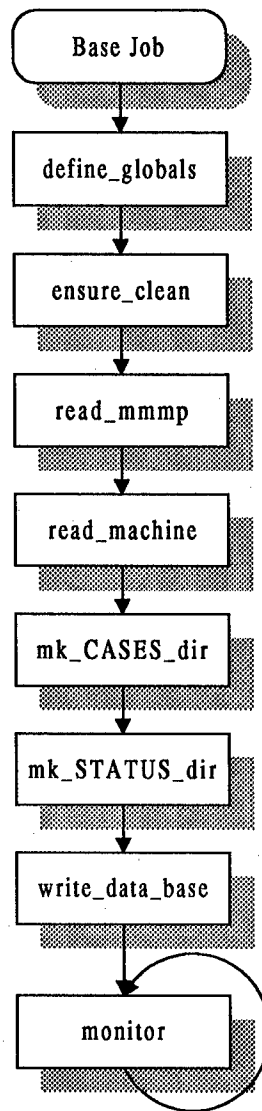
OC #

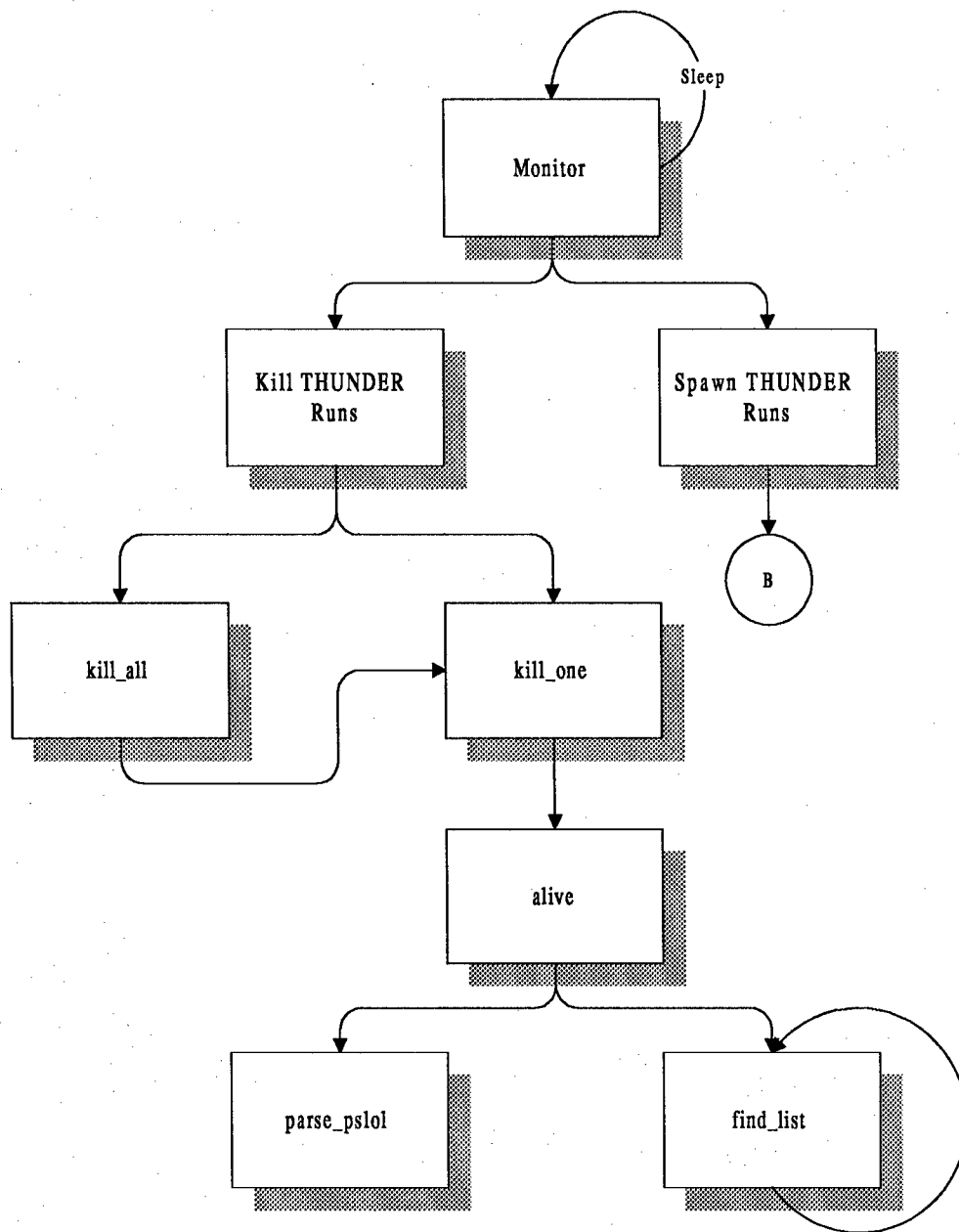
OT # 9.7.3 Create belief that reinforcements and supplies not forthcoming

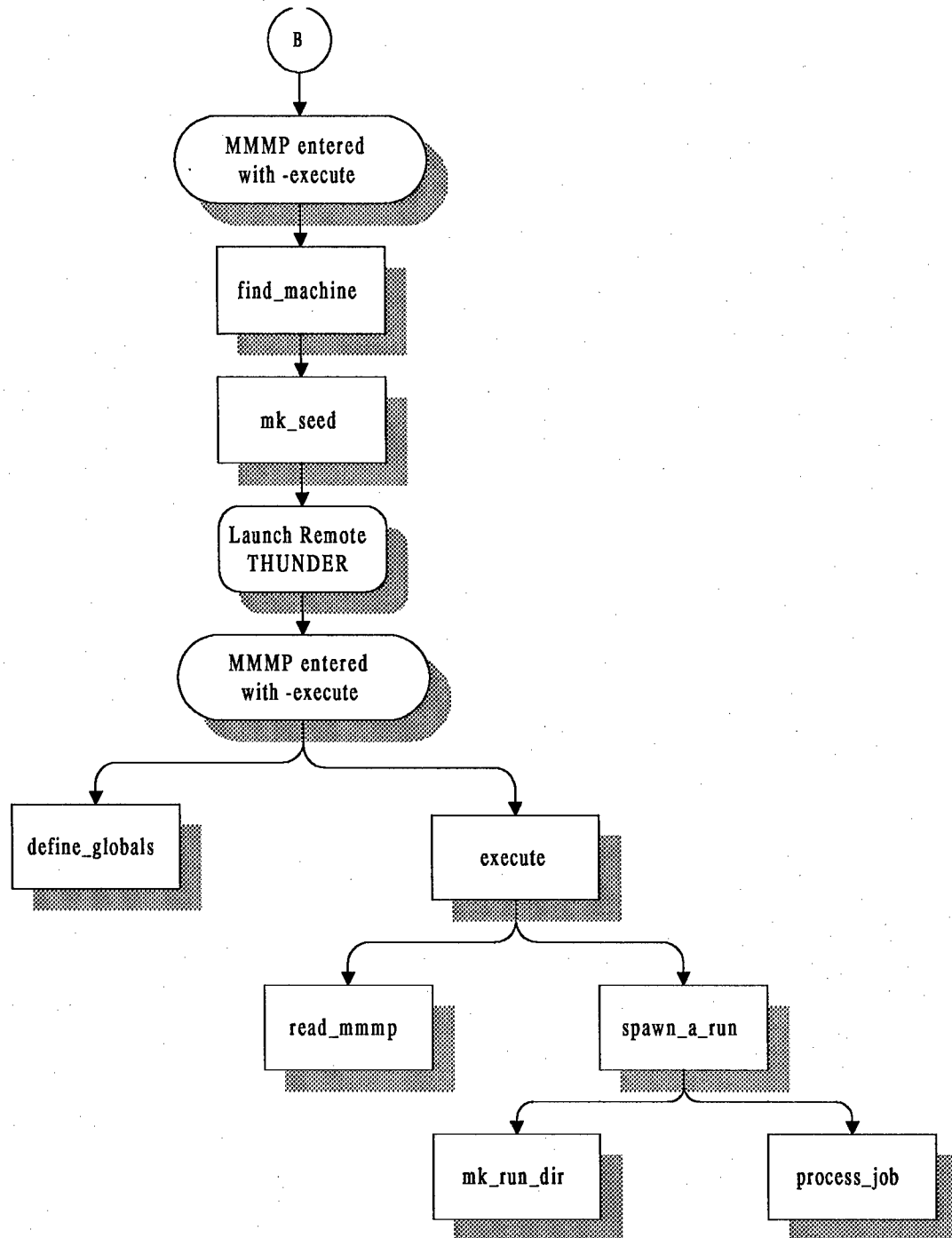
OC #

Appendix C. Perl Scripts

This Appendix contains the scripts used for this thesis.







MMMP

```
#!/usr/local/bin/perl

# The first line of this program must provide the location of Perl.
# If you have trouble consider trying this alternative approach to indicating
# to the machine where Perl is located. Substitute ! for exclamationpoint and
# remove the # signs on the eval and if 0 lines

#exclamationpoint/bin/sh -- # -*- perl -*- -w
#eval 'exec perl -S $0 ${1+"$@"}'
# if 0;

#-----
#
#                               Work to Do
#-----
# in mk_CASES_dir
# for some reason I can't get the following line to work. I need to be able
# to dynamically figure out how many columns there are in the array
#   for ($j=1; $j<=$#{Design_array[$j]};$j++)
#
#-----
#                               &chk_mach_type;
# build a subroutine that is more robust in determining the
# local operating system and paths for necessary information
#-----
#-----
#                               General Comments
#-----
# Comments is this code are preceeded by a # sign. If there appears to be
# any code that has been commented out I was probably trying to debug the
# script and didn't get around to removing it, or left it in to provide hints
# where you may need to throw in debug prints when you modify this script.
# If you feel confident that you understand what is going on, then
# feel free to modify the code to suit your needs.
# My convention is to bring the curly braces out where I can see them.
# There were too many times where I had trouble debugging this script because
# of a missing brace. So I put them where they are more noticeable.
#-----

# This program is intended to start and continue running THUNDER runs
# on multiple cpus on multiple machines.

# If this is a base execution:
#   o Read in the data about computers
#   o Write the database
#   o Schedule a "monitor" job.
#
# If this is a monitor job:
#   o Read in the database
#   o Gather up the runs that are finished.
#   o Kill runs that exceed max runtime
#   o Look around to see what computers are free
#   o Launch the runs on free computers
#   o Update the database
#   o Schedule a "monitor" job.
#
# If this is a run job
```

```

# o Read in the data from command line
# o Create the directories
# o Copy over the executables and other information
# o Start the thunder run.
# o If it ends correctly, write little database file and process job
# o If it dies ugly, write error database file
#
# MMMPDB format, name of associative array : JOBS:
# runs_to_do num_runs_to_do
# runs_succeeded num_runs_successfully_completed
# runs_failed num_runs_failed_to_complete
# runs_launched num_runs_launched total
# max_run_time max_run_time (in seconds)
# max_failures max_failures
# machine_name num_launched on this machine
# ...
# Run:machine_name:run_number time_when_launched
# ...
#
# DONEJOB format:
# machine_name.run_number [ succeed | fail ]
#
# MMMP.CTL format:
# runs_to_do
# max_run_time
# max_failures allowed
# max_ouevrage allowed
# location_of_executables
# location_of_control_files
#
#-----
#
# THE SCRIPT
#
#-----
# Tells UNIX which permission bits to disallow when creating files and dirs
# umask 077 would result in permissions -rw----- upon creation of a file
# umask 000 places no limitations on permission granting authority for this
# script. In the script I will place limits as necessary via the MODE option
#-----
umask 000;

#the following is an attempt to fix an error occuring in monitor.pm where
#I was getting $0 = ./MMMP
$0 =~ s-.*/-- ;#trying to strip off leading . and /

use File::Copy;
use File::Basename;
use Cwd;

#-----
# Find out what type of machine we are currently operating on.
#-----
#&chk_mach_type; # not built yet

#-----
# Process command line options. Should really use Getopts.
#-----
$MONITOR = $EXECUTE = $CLEANUP = $HELP = $VERBOSE = 0;
$KILLER = $FORCE = 0;
$EDIR = getcwd;
$EDIR =~ s/\/tmp_mnt//; # to strip off the pesky /tmp-mnt from AFIT Zoo
#-----
while( defined($ARGV[0]) && $ARGV[0] =~ /^-/ ) # Get args

```

```

{
    _ = shift ;
    if( /-b.* / ) { $BUG++; next; } # run in debug mode
    if( /-c.* / ) { $CLEANUP++; next; } # Clean this dir
    if( /-C.* / ) { $CLEANUP++; $FORCE++; next; } # Clean up, forced
    if( /-e.* / ) { $EXECUTE++;
        $run_case = shift ;
        $run_rep = shift ;
        $JOBNUM = shift ;
        $EDIR = shift ;
        next; } # Execution mode
    if( /-k.* / ) { $KILLER++; # Killer mode
        $KJOB = shift ;
        next; }

    if( /-h.* / ) { $HELP++; next; } # Help
    if( /-m.* / ) { $MONITOR++; next; } # Monitor mode
    if( /-n.* / ) { $NRMTMP++; next; } # Save temp files
    if( /-r.* / ) { $CRN++; next; } # Use CRN
    if( /-v.* / ) { $VERBOSE++; next; } # Print things
    if( /-t.* / ) { $JUSTTESTING++; shift; next; } # Testing syntax
    die "\nBad arg: $_\n\nFor help type '$0 -help'.\n" ;
}

#-----
# Change to the execution directory. $EDIR will be set by
# rsh'ed commands above, or it will still be cwd.
#-----

chdir $EDIR ;
#print "Main: EDIR = $EDIR\n";

#-----
# Define global variables.
#-----
&define_globals;

#-----
# The Remote Shell arguments are used throughout.
#-----
$rsh_argmnts = ' -v ' if ( $VERBOSE );
$rsh_argmnts = ' -n ' if ( $NRMTMP );
$rsh_argmnts = ' -b ' if ( $BUG );
$rsh_argmnts = ' -r ' if ( $CRN );
$rsh_argmnts = ' -v -b ' if ( $VERBOSE && $BUG );
$rsh_argmnts = ' -v -n ' if ( $VERBOSE && $NRMTMP );
$rsh_argmnts = ' -v -r ' if ( $VERBOSE && $CRN );
$rsh_argmnts = ' -n -c ' if ( $NRMTMP && $CRN );
$rsh_argmnts = ' -b -c ' if ( $BUG && $CRN );
$rsh_argmnts = ' -n -b ' if ( $NRMTMP && $BUG );
$rsh_argmnts = ' -v -n -b ' if ( $VERBOSE && $NRMTMP && $BUG );
$rsh_argmnts = ' -v -n -r ' if ( $VERBOSE && $NRMTMP && $CRN );
$rsh_argmnts = ' -v -b -r ' if ( $VERBOSE && $BUG && $CRN );
$rsh_argmnts = ' -n -b -r ' if ( $NRMTMP && $BUG && $CRN );
$rsh_argmnts = ' -v -n -b -r ' if ( $VERBOSE && $NRMTMP && $BUG && $CRN );

#-----
# None of these come back, but put the exit 0 there.
#-----
&cleanup, exit 0 if $CLEANUP;
&monitor, exit 0 if $MONITOR;
&execute, exit 0 if $EXECUTE;
&helpopt, exit 0 if $HELP;
&spawn_a_run, exit 0 if $EXEC_JOB;

#-----

```

```

# If we get here, we must be at a base job.
#-----
&ensure_clean;
&read_mmmp;
&read_machine;
&mk_CASES_dir;
&mk_STATUS_dir;

#-----
# the following JUSTTESTING option can be used to set up directories
#-----
exit(0) if ($JUSTTESTING);

&writ_data_base;
&mk_log("$this_mach: ".time." Starting MMMP job at ", scalar localtime);
print STDOUT "\tEverything checks out\n" if ($VERBOSE);
#-----
# The next while loop is the heart of the script. It keeps the
# script going until all case/rep runs of the model have been made
#-----
$DONE = &monitor;
while ( !($DONE) )
{
    print " ***** $this_mach is sleeping *****\n";
    sleep $MONTIME;
    $DONE = &monitor;
}
print "\n", scalar localtime, "$0 Main complete\n";

exit(0);

#-----
#
# THE SUBROUTINES
#
#-----

#-----
#
# sub alive
#
#
# To avoid having to do this two different ways, this is passed
# either return_type = 1 or return_type = 2.
# return_type = 1 gives a list ( @totalist )
# return_type = 2 gives a hash array (
#-----

sub alive
{
    &mk_log ("$this_mach: ".time." joblisting reached");#debugg
    my( $amachine, $acase, $arep) = @_;
    $user_name = getpwuid $<;
    $i++;
    if ($amachine eq $this_mach)
    {
        @psout = `ps -fj -u $user_name`;
    }
    else
    {
        @psout = `rsh $amachine ps -fj -u $user_name`;
    }
}

```

```

# Get the ps list of lists
@apslol = &parse_pslol( @psout );

# Go through the list of lists, and push the csh job.  add rest of jobs
@totalist = ();
for $i ( 0 .. $#apslol )
{
    if ( $apslol[$i][2] =~ /csh.*-e $acase $arep/)
    {
        push @totalist, $apslol[$i][0];
        push @totalist, &find_list( $apslol[$i][0], @apslol );
        last;
    }
}
return @totalist;
}

#-----
#
# CLEANUP
#
# Cleans up the execution directory, removing:
#   Database files
#   seedval files for THUNDER
#   xmon file
#   Directories containing the runs
#
# If FORCE is true (because called with -C vice -c), then it
#   does not ask before removing all the directories.
#
# List_Dir is created in spawn_a_run.pm from $dir_label="Dir: ".$run_label;
# run_dir are the different directories created for each separate run
#-----

sub cleanup
{
    print "cleanup reached\n"; #debug

    # dbmopen(%JOBS, "$MMMPDB", 0644);
    # &kill_all;
    # dbmclose(%JOBS);
    # print glob "$MMMPDB.*" "\n";
    unlink MMMPDB.pag if (-e MMMPDB.pag);

    unlink MMMPDB.dir if (-e MMMPDB.dir);

    unlink <DONEJOB.*> if (-e <DONEJOB.*>);
    unlink $DONEfile if (-e $DONEfile);
    unlink $XMON if (-e $XMON);

    # unlink $ACTIVE if (-e $ACTIVE);
    # unlink <seedval_*.dat> if (-e <seedval_*.dat>);
#-----
# This is an attempt to chdir into CASES_dir and remove
#-----
    unlink <$caseslocation/seedval_*.dat> if (-e <$caseslocation/seedval_*.dat>);
    # cannot decide if I want this here or in main
    &read_mmmp;

    #system("rm -rf $statuslocation");
    system("rm -rf $statuslocation") if (-d "$statuslocation");
    #and warn "Cannot remove run_dir ($runlocation) ($!)";
}

```

```

#try to have this automated by having user specify what to delete
system("rm -rf $caseslocation/seedval*");

# remove the output case directories
system("rm -rf $outlocation/S_Case_*");

# Get the case run directories of interest
opendir( DIRLOC, "$runlocation") or die "Cannot opendir mmmplacement($!)";
@List_Dirs = grep { /^Dir_/ } readdir(DIRLOC);
closedir( DIRLOC );

# Go through them one at a time
DIRLOOP: for ($dircount=0; $dircount<=$#List_Dirs; $dircount++)
{
    $run_dir = $List_Dirs[$dircount];
    if ( !($FORCE) )
    {
        print STDERR "Remove directory ($run_dir) [y=yes] (n=no,a=all,q=quit)? ";
        $inval = <STDIN>;
    }
    else
    {
        $inval = "a";
    }

    # If desired, get rid of this directory
    if ($inval =~ /^y/ or $inval =~ /^$/ )
    {
        # operating system dependent
        system("rm -rf $run_dir")
        and warn "Cannot remove run_dir ($run_dir) ($!)";
    }

    # Want to get rid of all the rest:
    elsif ($inval =~ /^a/) {

        # Print out the rest to make sure
        if ( !($FORCE) )
        {
            print STDERR "You sure you want to delete the following:\n";
            for ($rest=$dircount; $rest<= $#List_Dirs; $rest++)
            {
                print STDERR "\t$List_Dirs[$rest]\n";
            }
            print STDERR "[y=yes] (n=no)? ";
            $inval2 = <STDIN>;
        }
        else
        {
            $inval2 = "y";
        }

        # Really does want to delete. Do it:
        if ($inval2 =~ /^y/ or $inval2 =~ /^$/ )
        {
            for ($rest=$dircount; $rest<= $#List_Dirs; $rest++)
            {
                $run_dir = $List_Dirs[$rest];
                # operating system dependent
                system("rm -rf $runlocation/$run_dir")
                and warn "Cannot remove run_dir ($run_dir) ($!)";
            }
        }
    }
}

```

```

        # All gone, so don't need to loop.
        last DIRLOOP;
    }
    else
    {
        # Bailed out of deleting all, so need to reconsider
        # the one that we got an "all" for
        redo DIRLOOP;
    }
}

# Want to skip this deleting stuff
elsif ($inval =~ /^q/)
{
    last DIRLOOP;
}
}
exit 0;
}

#-----
#
#           Define the global variables used by the subroutines.
#-----

sub define_globals
{
    $|                = 1;                # Ensures flush
    $DONEfile         = "DONE.cat";        # All the DONEJOB info
    $MACH              = "machine.dat";    # Machine information
    $MMMPDB            = "MMMPDB";        # MMMP database file
    $MMMPCTL           = "MMMP.CTL";      # MMMP control file
    $XMON              = "xmon";          # monitor execution history
    $THCTL             = "THUNDER.CTL";   # thunder control file
    $CTLDAT            = "control.dat";   # control.dat
    $SEEDVAL           = "seedval.dat";   # initial seeds
    $period            = 2147483648;      # Random Number
    $SRN_interval      = 1000000;         # Random Number interval
    $MONTIME           = 180;             # Number of seconds
    $DESIGNCTL         = "DESIGN.CTL";    # Design Matirx
    $ACTIVE            = "executing.dat"; # text file of executing jobs
    $user_name         = getpwuid $<;     # User name

    # this will give the location of the perl source
    ($EXECNAME, $MMMPDIR, $suffix) = fileparse($0, "");
    if ($MMMPDIR =~ /^$/ or $MMMPDIR =~ /\.\/\//)
    {
        $MMMPDIR = getcwd;
        # $MMMPDIR =~ s/\/tmp_mnt//; # to strip off the pesky /tmp-mnt at zoo
    }

    # This (might) give the machine name that this is called from
    if (-e "/usr/ucb/hostname")
    {
        $this_mach = `/usr/ucb/hostname`;
        chomp $this_mach;
    }
    elsif ($this_mach = `hostname`)
    {
        chomp $this_mach;
    }
    else
    {

```

```

    $this_mach = '';
}
#print "Define: machine = $this_mach\n";          # debug
#print "Define: The current user is $user_name\n";  # debug
}

#-----
#
#                               If things are not properly set up, die.
#
#-----

sub ensure_clean
{
    die "$XMON exists and should not " if -e $XMON;
    die "$MMMPDB.dir exists and should not " if -e "$MMMPDB".dir";
    die "$MMMPDB.pag exists and should not " if -e "$MMMPDB".pag";
    die "$MMMPCTL does not exist and should " if !(-e $MMMPCTL);
    die "$MACH does not exist and should " if !(-e $MACH) ;
    #die "nextseed does not exist and should" if !(-e "$MMMPDIR/nextseed.sim");
}

#-----
#
#                               EXECUTE
#
# Sent here by main, does the execution of a single job.
# Sets up some variables and calls spawn_a_run.
# DONEJOB format:
#   machine_name.run_number      succeed | fail
#-----

sub execute
{
    &mk_log ("$this_mach: ".time." sub execute reached") if ($BUG);
    &read_mmmmp;

    if (defined($this_mach))
    {
        $run_label = "Run_". $this_mach. "_Case_". $run_case .
                     "_Rep_". $run_rep ;
        $DONEJOBfile = "DONEJOB.". $run_label;
    }
    else
    {
        die "\$this_mach is not defined.";
    }

    $exe_return = &spawn_a_run;

    #-----
    # Open donejob file _after_ completion, or the monitor job
    # will find it and won't know that it's not finished.
    #-----
    open ( DJ, ">$EDIR/$DONEJOBfile");
    if ( $exe_return == 0 )
    {
        &mk_log ("$this_mach: ".time." Execute: $run_label successful execution ",
        scalar localtime);
        print DJ "$run_label      succeed  ",time,"\n";
    }
}

```

```

elseif ( $exe_return == -1 )
{
    &mk_log("$this_mach: ".time." Execute: $run_label failed execution, type -
1",
        scalar localtime);
    print DJ "$run_label      fail      ",time," \n";
}
else
{
    &mk_log("$this_mach: ".time." Failed execution of $run_label, type 0
",scalar localtime);
    print DJ "$run_label      fail      ",time," \n";
#####
#####
# New Stuff pulled in from monitor as an attempt to keep track of failures
    # if (&kill_one( $key, $amachine, $acase, $arep))
    #{
        $fail_label = "Fail_."$run_label";
        rename ("$statuslocation/$key","$statuslocation/$fail_label");
        open (FAILFILE, ">>$statuslocation/$fail_label")
            or die "cannot open $statuslocation/$fail_label: $!";
        print FAILFILE scalar localtime, "      fail_label\n";
        close FAILFILE;
    # }
    $skill_label = "Dir_."$run_label";
    print"Execute: kill label = $skill_label\n";
    system("/bin/rm -rf $runlocation/$skill_label") if (-d
"$runlocation/$skill_label");

#I don't really know why this directory is being eliminated elsewhere, this is
#just an attempt to patch this program so I can do some runs done. I need to
# get into this and see if the to kill is working.
#####
#####
    }
    close( DJ );
    &mk_log ("$this_mach: ".time." Execute: on machine $this_mach") if ($BUG);
}

#-----
#
#               find_list
#
#   Returns a list of csh jobs to sub alive
#-----
sub find_list
{
    my( $pjob, @apslol ) = @_;
    my( @returnlist );

    @returnlist = ();
    for $i ( 0 .. $#apslol )
    {
        if ($apslol[$i][1] == $pjob)
        {
            push @returnlist, $apslol[$i][0];
            push @returnlist, &find_list( $apslol[$i][0], @apslol );
        }
    }
    return @returnlist;
}

#-----
#
#               find_machine
#-----

```

```

#
# Search the machines listed in machine.dat to find one that is available
#-----

sub find_machine
{
    &mk_log("$this_mach: ".time." sub find_machine reached") if ($BUG);
    LOOPC: foreach $machine ( @machine_name )
    {
        #&mk_log("$this_mach: ".time." find_machine: trying $machine");

        #-----
        #
        #           Do we really want to use this machine???
        #
        # Have we exceeded the number of CPUs for this machine? -----
        # While a job is active there will be a file called Run_$machine
        # in the STATUS_dir
        #-----
        opendir( STATLOC, "$statuslocation")
            or die "Cannot opendir status location/..(!)";
        @Run_Files = grep { /^Run_/ } readdir(STATLOC);
        closedir( STATLOC );

        # Go through them one at a time
        # print "In find and machine is $machine\n";
        $JOBS{$machine} = 0;
        foreach $file_name (@Run_Files)
        {
            # print "a run file is $file_name\n";
            if ($file_name =~ /^{$machine}/)
            {
                $JOBS{$machine}++;
            }
        }
        next LOOPC if ( $JOBS{$machine} >= $machine_cpus{$machine} );
        $rup_response = `rup $machine`;
        @rup_stuff = split( ' ', $rup_response );

        # Is the machine down? -----
        next LOOPC if !( $rup_stuff[1] =~ /up/ );

        # Is the machine busy? -----
        $rup_stuff[7] =~ s/,//; # Location of machine load w/o
        next LOOPC if ( $rup_stuff[7] >= $machine_cpu_limit{$machine} );
        return $machine;
    }
    return ( 0 );
}

#-----
#
#           find_rsh
#
# Find a particular rsh'ed job
#
#-----

sub find_rsh
{
    my( $machine, $case, $rep ) = @_;

    # Get a list of processes running on this machine
    # `ps -alxww` works for the solaris 2.6 operating system
    # `ps -fl` works for SunOS 5.5.1 and IRIX 6.2 operating systems
    #-----

```

```

$user_name = getpwuid $<;

@psout = `ps -fj -u $user_name`;

# Get the ps list of lists
@apslol = &parse_pslol( @psout );

# Find the rsh
for $i ( 0 .. $#apslol )
{
    if ( $apslol[$i][2] =~ /rsh.*$machine.*-e $case $rep/)
    {
        if ($VERBOSE)
        {
            &mk_log("$this_mach: ".time." rsh Case: $case Rep: $rep on
$machine");
        }
        return( $apslol[$i][0] );
    }
}

if ($VERBOSE)
{
    &mk_log("$this_mach: ".time." Unable to find rsh for Case: $case Rep:
$rep on $machine");
}
return(0);
}

#-----
#                               HELP
#
#                               Print out the help message here.
#-----

sub helpopt
{
    # <<EOH indicates a "here" document and all that follows until the next EOH
    # is considered quoted material. Terminating EOH must appear by itself
    # (unquoted and with no surrounding whitespace) on the terminating line
    $HelpInfo = <<EOH;

    $EXECNAME - MMMP

    $EXECNAME [-b] [-c|C] [-h] [-r] [-v]

    This program ($EXECNAME) is intended to execute an experimental design
    over a network of machines listed in machine.dat.

    User Flags:
        -b          Bug mode. Prints addition information to xmon file.
        -c          Cleanup. delete subdirs, stop jobs, remove files.
        -C          Non-interactive cleanup.
        -h          Help. Print this help message.
        -r          Random Number Deselect. Allows for Common Random
Numbers
        -v          Verbose. Add additional info in xmon about the runs

    Files:

```

```

machine.dat:  Contains the data about what machines to be used
              and the cpu usage allowed on each.  Comments can be started with
              a pound:
              # Example:
              # This is the information about the computers
              # name_of_computer  number_of_cpus  max_limit
                bass                1              0.8
                eel                 4              3.2

MMMP.CTL:  MMMP control file.  Tells $EXECNAME how many runs to do,
           where to do them, where the THUNDER executables are, and information
           about failures.
EOH
print STDERR $HelpInfo;
}

#-----
#                               kill_subs
#
# This group of subroutines handles the killing of launched runs.
#   kill_all:  foreach job in database, calls kill_one
#   kill_one:  Uses system "kill -9" to kill a job
#   find_list: finds all the PIDs for the job being killed
# Kill all the remaining jobs because either:
#   we're finished, or
#   we have too many failures, or
#   we're cleaning up
# This assumes that the JOBS database is already open
#-----

sub kill_all
{
    &mk_log("$this_mach: ".time." ub kill_all reached") if ($BUG);
    while ( ($key, $val) = each(%JOBS) )
    {
        if ($key =~ /^Run_(\S+)_Case_(\w+)_Rep_(\d+)$/ )
        {
            $amachine = $1;
            $acase     = $2;
            $arep      = $3;
            &mk_log("$this_mach: ".time." Kill case $acase rep $arep on machine
$amachine");
            &kill_one( $key, $amachine, $acase, $arep );
        }
    }
}

#-----
#                               kill_one
#
# Kill specific job on a specific machine
#-----

sub kill_one
{
    &mk_log("$this_mach: ".time." sub kill_one reached") if ($BUG);
    my( $akey, $amachine, $acase, $arep ) = @_;

    # First, see if there are any processes on the remote machine
    @alivearray = &alive( $amachine, $acase, $arep );
    if ( $#alivearray != 0 )
    {

```

```

# Kill jobs on the machine in question
foreach (reverse @alivearray)
{
    if ($VERBOSE )
    {
        print STDOUT " Alive array $_ \n";
    }
    # DEBUGGING CODE
    # $sys_return = system "rsh -n $amachine kill -9 $_ > /dev/null 2>&1 &"
;
    $sys_return = system "rsh -n $amachine kill -9 $_ &" ;
    # DEBUGGING CODE
}
sleep 15;

# Check to see if it is dead now.
@alivearray = @alive( $amachine, $acase, $arep );
if ( $#alivearray > 0 )
{
    &mk_log("$this_mach: ".time." Unable to kill $acase, $arep on
$amachine");
    delete( $JOBS{$akey} );
    $JOBS{"runs_failed"}++;
}
}

# Second, see if there is a rsh job on this machine
# This job should finish (and create a DONEJOB file) when we
# kill the above processes. If not, then do it here
$ajob_rsh = &find_rsh( $amachine, $acase, $arep );
if ($VERBOSE)
{
    print STDOUT "$ajob_rsh is remote shell job \n";
}
$run_label = "Run ".$amachine."_Case_".$acase."_Rep_".$arep;
#$run_label = "Run_" . $amachine . ":" . $ajob ;
$DONEJOBfile = $EDIR . "/" . "DONEJOB." . $run_label;
if ( $ajob_rsh != 0)
{
    kill 9, $ajob_rsh;          # kill rshell
    open ( DJ, ">$EDIR/$DONEJOBfile");
    print DJ "$run_label      fail      ",time," \n";
    close( DJ );
}
if ( !(-e $DONEJOBfile) )
{
    open ( DJ, ">$DONEJOBfile");
    print DJ "$run_label      fail      ",time," \n";
    close( DJ );
}
return;
}

```

```

#-----
#
#                               mk_CASES_dir
#
# To use this subroutine some background work was accomplished first:
# 1. An experimental design matrix (-1,0,1) was generated with JMP.
# 2. Using Excel this was converted into a series of "flags" that will
#    be used by the cpp preprocessor to modify the data files that will
#    be used in this particular study. The flag matrix is stored in
#    DESIGN.CTL which needs to be in the library directory -- LIB_dir.
#

```

```

# 3. The location of LIB_dir must be specified in MMMP.CTL which needs
# be in the MMMP directory -- MMMP_dir.
# -----
sub mk_CASES_dir
{
    &mk_log ("$this_mach: ".time." sub mk_CASES_dir reached") if ($BUG);
    #my($i) = 0 ;
    # The LIB_dir directory should initially have the master*.dat files
    # that need to be modified for this experimental design. Separate
    # directories will be created for each of the different design points
    # that contain modified versions of the *.dat files. To see what changes
    # will be made look at one of the master*.dat files. The first section
    # of these files contain the cpp directives that will be processed by
    # this subroutine

    opendir (LIBdir, "$liblocation") or die "cannot open dir ($liblocation)";
    @lib_files = readdir LIBdir;
    closedir LIBdir;
    foreach $lfile (@lib_files)
    {
        #print " a file is $lfile\n";
        if ($lfile =~ /master/)
        {
            # make a listing of the master .dat files for later use
            push @mdatfiles , $lfile;
        }
    }

    # Get the data Flag matrix from DESIGN.CTL in LIB_dir
    open ( DESIGNCTL , "$liblocation/$DESIGNCTL" )
        or die "Cannot open $DESIGNCTL ($!)";
    while (<DESIGNCTL)      # take one line of the design matrix at a time
    {
        push @Design_array, [split];          # split on white space
    }
    close DESIGNCTL;
    chdir $caseslocation or die "cannot chdir into ($caseslocation)";
    if (!( -w $caseslocation))
    {
        die "cannot make subdir in $caseslocation/..";
    }

    # $#Design_array gives the number of rows in the design matrix
    for ($i=0; $i<=$#Design_array; $i++)
    {
        $create_dir = "$Design_array[$i][0]";
        # check to see if the user has already set up a Case Directory with the
        # data files already loaded. If so use those and don't overwrite.
        unless (-d "$create_dir")
        {
            mkdir $create_dir, 0777 or die "cannot make ($create_dir)";
            chdir $create_dir or die "cannot chdir into ($create_dir)";

            # @mdatfiles was created above and holds a lister of master .dat files
            foreach $mdfile (@mdatfiles)
            {
                copy ("$liblocation/$mdfile" , ".$mdfile")
                    or die "cannot cp ($mdfile)";

                # The following requires the master data files in LIB_dir
                # to have the format: master_filename.dat
                $temp_mdfile = $mdfile;
            }
        }
    }
}

```

```

$mdfile =~ s/master_//;
#print"$mdfile\n";
$shortfile = $mdfile;
$mdfile = $temp_mdfile;
#print"$mdfile\n";
$FLAG = " ";

#####
# for some reason I can't get the following line to work. I need
# to be able to dynamically figure out how many columns there are
# in the array I tried ${Design_array[$i]} but that didn't seem
# to work
#####
# for ($j=1; $j<=${Design_array[$j]};$j++)
for ($j=1; $j<=11;$j++)
{
    $FLAG .= " -D$Design_array[$i][$j] ";
}
#####
# if mmp blows up in this vicinity it may be because it can't
# find the location of cpp. If a "which cpp" doesn't find it for
# you, try looking in /usr/ccs/lib/cpp

$templ = "master_test_file";
# The following works on kellie
#system " /usr/ccs/lib/cpp $FLAG $mdfile >$templ";

system " $ccplocation $FLAG $mdfile >$templ";
#I tried to do the following section of code with this command
#system "-v '^ *$|^#' $templ > $shortfile"
#but couldnt get the egrep regular expression to work properly
open ( TEMP1, ".$templ" ) or die "cannot open $templ: $!";
#print "$shortfile\n";
open (MDFILE, ">.$shortfile")
    or die "cannot open $shortfile: $!";
while (<TEMP1>)
{
    if (! /^#/ )    # strip off the cpp directives
    {
        print MDFILE;
    }
}
close MDFILE;
close TEMP1;

# no excess baggage allowed
system "rm $mdfile $templ";

}

chdir $caseslocation or die "cannot chdir into ($caseslocation)";

}

chdir $EDIR;
}

#-----
#                               mk_STATUS_dir
#
# Create a directory to track the status of THUNDER runs
#-----

```

```

sub mk_STATUS_dir
{
    &mk_log ("$this_mach: ".time." sub mk_STATUS_dir reached") if ($BUG);
    $oldpwd = cwd or die "cannot get current working directory";
    chdir $mmmplocation or die "cannot chdir into ($mmmplocation)";
    chdir ".." or die "cannot chdir into ($mmmplocation/..)";
    $newpwd = cwd or die "cannot get new working directory";
    $newpwd =~ s/\/tmp_mnt//; # to strip off the pesky /tmp-mnt from AFIT ZOO
    if ( !(-w $newpwd) )
    {
        die "cannot make subdir in $mmmplocation/..";
    }
    #will need to be cleaned up in sub clean
    unless ( -d $statuslocation )
    {
        mkdir $statuslocation, 0777 or die "cannot make dir ($statuslocation)";
    }
    $stat_rep = 1;
    LOOPD: foreach $case ( @cases ) #iterate over cases
    {
        #print " case is $case\n";
        LOOPE: while( $stat_rep <= $NREPS ) #iterate over reps
        {
            #print "NREP is $NREPS and $stat_rep is stat_rep\n";
            $STATFILE = "Case ".$case."_Rep_". "$stat_rep";
            open (STATFILE, ">>$statuslocation/$STATFILE")
                or die "cannot open $statuslocation/$STATFILE: $!";
            print STATFILE scalar localtime, " $STATFILE\n";
            close STATFILE;
            $stat_rep++;
        } #end of LOOPE
        $stat_rep = 1; #start with rep 1 on the new case.
    } #end of LOOPD
    chdir $oldpwd or die "cannot chdir into ($oldpwd)";
}

#
#-----
#
#               mk_log
#
# Since there can be many, many files that want to print to XMON, you don't
# want to have a routine keeping it open for long. So, every time you want to
# print, call this routine.
#
#-----

sub mk_log
{
    my( @tobeprinted ) = @_;
    open( XMON, ">>$EDIR/$XMON" );
    # flock( XMON, 2); # Get an exclusive lock
    print XMON "@tobeprinted\n";
    close( XMON );
}

#
#-----
#
#               mk_run_dir
#
# Create all directories and files needed for this run of THUNDER
#
#-----

```

```

sub mk_run_dir
{
    &mk_log (" $this_mach: ".time." mk_run_dir reached") if ($BUG);

    $dir_label = "Dir_" . $run_label;

    chdir $mmmplocation      or die "cannot chdir into ($mmmplocation)";
    $newpwd = cwd            or die "cannot get new working directory";
    $newpwd =~ s/\/tmp_mnt//; # to strip off the pesky /tmp-mnt
    if ( !(-w $newpwd))
    {
        die "cannot make subdir in $mmmplocation/..";
    }

    unless ( -d $star_dir )
    {
        #mkdir $star_dir, 0777 or die "cannot make dir ($star_dir)";
    }

    unless ( -d "$runlocation/$dir_label" )
    {
        mkdir "$runlocation/$dir_label" , 0777
            or die "cannot make subdir ($dir_label)";
        chdir "$runlocation/$dir_label"
            or die "cannot chdir into ($dir_label)";
        mkdir "Backmods", 0777 or die "cannot make backmods";
        mkdir "Backup", 0777 or die "cannot make backup";
        mkdir "Reports", 0777 or die "cannot make reports";
        mkdir "Graphics", 0777 or die "cannot make graphics";

        #-----
        #transfer a copy of THUNDER.CTL from caseslocation dir into working dir
        # in addition we need to modify the data line, so THUNDER will know
        # where to look for the dat files for this design point
        #-----
        open (TH,"$liblocation/$THCTL") or die "can't open $THCTL to read : $!";
        open (NEWTHCTL, ">./$THCTL")
            or die "cannot open $dir_label/$THCTL: $!";
        while (defined($line1 = <TH>))
        {
            #read in a line from the baseline THUNDER.CTL
            chomp $line1;
            $line1 =~ s?dummy_string?${caseslocation}/Case_${run_case}?;
            print NEWTHCTL "$line1\n";
        }
        close NEWTHCTL;
        close TH;
        #
        # copy (" $liblocation/control.dat","control.dat")
        # or die "cannot cp control.dat";

        # Need following in run directory in order to get ttgraph.rpt
        copy (" $liblocation/ttgraph.cfg","ttgraph.cfg")
            or die "cannot cp ttgraph.cfg";

        #-----
        # By doing the following we can get the a more understandable report
        # title on the THUNDER output - ttgraph.rtp Will help data processing
        #-----
        open (CDT,"$liblocation/$CTLDAT") or die "cannot open $CTLDAT: $!";
        open (NEWCDT, ">./$CTLDAT") or die "cannot create $CTLDAT: $!";
        while (defined($line2 = <CDT>))
        {
            #this does not appear to work -- will have to work on this later
            chomp $line2;
            if ($line2 =~ /\^s*REPORT.TITLE\s+(S+)/)

```

```

        {
            $line2 =~ s/$1/"Case: $run_case Rep: $run_rep"/;
        }

        print NEWCDT "$line2\n";
    }
    close NEWCDT;
    close CDT;
}#end of unless

#The following is to prevent the interactive prompt given by THUNDER
unlink input.dat if (-e "$dir_label/input.dat");

if (-e "$caseslocation/seedval_$JOBNUM.dat")
{
    copy "$caseslocation/seedval_$JOBNUM.dat", "seedval.dat"
        or die "could not copy seedvalue file to $dir_label";
}
else
{
    die "Could not find the seedvalue file ";
}
}#end of mk_run_dir

#-----
#
#                               mk_seed
#
# Set the random input streams for THUNDER. This calls the simscript
# program nextseed with 1 parameter for the first execution, and
# with 2 parameters for those following.
#-----

sub mk_seed
{
    &mk_log ("$this_mach: ".time." sub mk_seed reached") if ($BUG);
    $jobnumber = $JOBNUM;

    if ($jobnumber == 1 || $CRN )
    {
        #print"$liblocation/$SEEDVAL\n";
        copy("$liblocation/$SEEDVAL", "$caseslocation/seedval_$jobnumber.dat")
            or die "cannot cp seedval file : $!";
        return;
    }

    # make new seedvalue file by adding $RN_interval (from sub define) to the
    # previous values
    $lastnumber = $jobnumber - 1;

    unless (-e "$caseslocation/seedval_$jobnumber.dat" )
    {
        open ( SEED , "$caseslocation/seedval_$lastnumber.dat")
            or die "cannot open seedvalues file $lastnumber in $caseslocation: $!";
        open (NEWSEED, ">$caseslocation/seedval_$jobnumber.dat")
            or die "cannot create newseed file in $MMMPDIR: $!";
        LOOP15: while (defined($input=<SEED>))
        {
            chomp $input;

            if ($input =~ /\^s*(\d+)\s+(\d+)/)
            {

```

```

        $temp = $2 + $RN_interval;
        if ($temp > $period )
        {
            $temp = $temp - $period;
        }
        $printval = "        ".$1"."        ".$temp";
        print NEWSEED "$printval\n";
    next LOOP15;
}
print NEWSEED "$input\n";
}
close SEED;
close NEWSEED;
return;
}
}

#-----
#
# monitor
#
# Monitor job subroutine.
# 1. kill jobs that have taken too long
# 2. finish if we're done or have too many failures
# 3. Launch a THUNDER run if there is an available CPU
#-----

sub monitor
{
    my ( $j ) = 0;
    &mk_log ("$this_mach: ".time." sub monitor reached") if ($BUG);
    &read_machine;
    # Read this each time
    # so we can get changes
    # to the machine file

    &mk_log("$this_mach: ".time." Monitor executed at ", scalar localtime);

    #-----
    # Open database, get all the jobs
    #-----
    dbmopen(%JOBS,"$MMMPDB",0644);

    if ( $VERBOSE )
    {
        &mk_log("$this_mach: ".time." Monitor: Data from JOBS database:");
        @sorted_keys = sort keys %JOBS;
        foreach $key_val (@sorted_keys)
        {
            &mk_log("$this_mach: ".time." $key_val\t\t$JOBS{$key_val}");
        }
    }

    #-----
    # Get the jobs that are done
    #-----
    LOOP6: foreach $donejob ( <DONEJOB.*> )
    {
        if ($VERBOSE)
        {
            &mk_log("$this_mach: ".time." $donejob donejob file name");
        }

        open( DJ , $donejob);
        $indata = <DJ>;
        chop( $indata );
    }
}

```

```

close( DJ );
unlink $donejob;

($machine_and_job, $succeed_or_fail, $run_end_time)
    = split(' ', $indata);

if ($VERBOSE)
{
    &mk_log("$this_mach: ".time." Data in $donejob: ($indata)");
}

# Should be an entry in JOBS that corresponds to this machine/job
if ( !( $JOBS{ $machine_and_job } ) )
{
    &mk_log("$this_mach: ".time." Error! ($machine_and_job) not in
database!");
    next LOOP6;
}

open( DJ0, ">>$DONEfile");
printf DJ0 ("%40s %15d %20d seconds \n", $indata,
$JOBS{$machine_and_job},
    $run_end_time-$JOBS{$machine_and_job});
close(DJ0);

#-----
# Deal with the jobs that succeeded or fail
#-----
if ( $succeed_or_fail =~ /succeed/ )
{
    &mk_log("$this_mach: ".time." $machine_and_job succeeded at ",scalar
localtime);
    $JOBS{"runs_succeeded"}++;
}
else
{
    &mk_log("$this_mach: ".time." $machine_and_job failed:
($succeed_or_fail)");
    $JOBS{"runs_failed"}++;
}
delete $JOBS{ $machine_and_job };
}#end of LOOP6

if ($VERBOSE)
{
    $pass = $JOBS{"runs_succeeded"};
    &mk_log("$this_mach: ".time." Runs that have succeeded : $pass");
    $pass = $JOBS{"runs_launched"};
    &mk_log("$this_mach: ".time." Runs that have launched : $pass");
}

#-----
# Kill the jobs that have exceeded the time limit
#-----
while ( ($key, $started_time) = each(%JOBS) )
{
    if ($key =~ /^Run_(\S+)_Case_(\w+)_Rep_(\d+)/ )
    {
        $machine = $1;
        $acase    = $2;
        $arep     = $3;
        #print"Monitor: $key\n";
        #print"Monitor: $machine\n";
        # print"Monitor: $acase\n";
    }
}

```

```

# print "Monitor: $arep\n";
if ($started_time =~ /^\\d+$/ )
{
    $runtime_used = (time - $started_time);
    if ( $runtime_used > $JOBS{"max_run_time"})
    {
        &mk_log("$this_mach: ".time." $key exceeded runtime limit; ",
            "($runtime_used > $JOBS{max_run_time})");
        &kill_one( $key, $amachine, $acase, $arep);
        &kill_one( $key, $amachine, $acase, $arep);

        $fail_label = "Fail_". "$key";
        rename (" $statuslocation/$key", "$statuslocation/$fail_label");
        open (FAILFILE, ">>$statuslocation/$fail_label")
            or die "cannot open $statuslocation/$fail_label: $!";
        print FAILFILE scalar localtime, "    fail_label\n";
        close FAILFILE;
        $dir_kill = "Dir_". "$key";
        system("/bin/rm -rf $runlocation/$dir_kill") if (-d "$runlocation/$dir_kill");
    }
}
else
{
    &mk_log("$this_mach: ".time." Something wrong with the time ",
        " ($started_time). Killing $key");
    &kill_one( $key, $amachine, $acase, $arep);
}
}

#-----
# If we're finished, stop.
# if ($runs_succeeded >= $JOBS{"runs_to_do"} ) {
#-----
if ( $JOBS{"runs_succeeded"} >= $JOBS{"runs_to_do"} )
{
    &mk_log("$this_mach: ".time." We're done! ", scalar localtime);
    &kill_all;
    dbmclose( %JOBS );
    unlink $ACTIVE if (-e $ACTIVE);
    return( 1 );
}

#-----
# If too many failures, kill all remaining and quit
#-----
if ( $JOBS{"runs_failed"} > $JOBS{"max_failures"} )
{
    &mk_log("$this_mach: ".time." Too many failures, bailing out! ", scalar
localtime);
    &kill_all;
    dbmclose( %JOBS );
    unlink $ACTIVE if (-e $ACTIVE);
    return( 1 );
}

#-----
# Go through all the machines and find one with an idle cpu
#-----

$JOBNUM = $JOBS{"runs_launched"};
$rep = 1;

```

```

LOOPD: while (1)
{
    $machine = &find_machine;
    &mk_log ("$this_mach: ".time." The machine returned from find_machine is
$machine")
    if ($BUG);
    if ( $machine eq 0 )
    {
        last LOOPD; # no machine available time to wait
    }

    opendir( STATLOC, "$statuslocation") or die "Cannot opendir status
location/..($!)";
    @Run_Files = grep { /^Run_/ } readdir(STATLOC);
    closedir( STATLOC );
    opendir( STATLOC, "$statuslocation") or die "Cannot opendir status
location/..($!)";
    @Comp_Files = grep { /^Complete_/ } readdir (STATLOC);
    closedir( STATLOC );
    # Go through them one at a time

    LOOPA: foreach $case ( @cases ) #iterate over cases
    {
        LOOPB: while( $rep <= $NREPS )#iterate over reps
        {

            # Is this case/rep combo currently executing on another machie?--
            foreach $run_name (@Run_Files)
            {
                if ($run_name =~ /Case_${case}_Rep_${rep}/)
                {
                    $rep++;
                    next LOOPB;
                }
            }
            # Has this case/rep combo completed executing on another machine
            foreach $comp_name (@Comp_Files)
            {
                #print "Monitor: comp_name is $comp_name\n";
                if ($comp_name =~ /Case_${case}_Rep_${rep}/)
                {
                    $rep++;
                    next LOOPB;
                }
            }

            #-----
            # We are now ready to launch a THUNDER run.
            # First build a seedval.dat file for this run.
            #-----
            $JOBNUM++;
            #used to generate RN seedval.dat file
            &mk_seed( $JOBNUM );

            # Add the job to the database-----
            $job_label = time;
            $job_key = "Run_".$machine."_Case_".$case."_Rep_".$rep;
            $JOBS{ "$job_key" } = "$job_label";
            &mk_log("$this_mach: ".time." Monitor: $job_key launched at $JOBS{$job_key}")
                if( $BUG );

            # Increment total runs launched and jobs launched on this machine
            $JOBS{"runs_launched"}++;
            $JOBS{$machine}++;
        }
    }
}

```

```

        #&mk_log("$this_mach: ".time." rsh arguments $rsh_argmnts") if
($BUG);
        # Launch the job and process return-----
        print"$this_mach launched $0 on $machine: Case_$case Rep_$rep JOB_$JOBNUM
$EDIR\n";
        # $sys_return = system "rsh $machine $0 $rsh_argmnts -e $JOBNUM
$EDIR $case $rep &";
        $sys_return = system "rsh $machine $0 $rsh_argmnts -e $case $rep
$JOBNUM $EDIR &";
        $sys_return &= 0xffff;
        if ( $sys_return == 0 )
        {
            &mk_log("$this_mach: ".time." In Monitor launched job $JOBNUM
on $machine", "at time ", scalar localtime);
            # &mk_log("$this_mach: ".time." Case $case and rep $rep \n");
            $RUNFILE= $job_key;
            open (RUNFILE, ">$statuslocation/$RUNFILE")
                or die "cannot open $statuslocation/$RUNFILE: $!";
            print RUNFILE scalar localtime, " $RUNFILE\n";
            close RUNFILE;
            next LOOPD;
        }
        else
        {
            &mk_log("$this_mach: ".time." Tried launching job $JOBNUM on
$machine, ",
                "giving response $sys_return, at time ",
                scalar localtime);
            next LOOPD;
        }
        $rep++;
    }#end of LOOPB
    $rep = 1; #start with rep 1 on the new case.
}#end of LOOPA
last LOOPD;
}#end of LOOPD
# This produces an ASCII file that has all the active jobs
# and their machine name
unlink $ACTIVE if (-e $ACTIVE);
undef %active_jobs if defined %active_jobs;
$j = 0;
foreach $akey ( keys %JOBS )
{
    if ($akey =~ /^Run_(\w+)/ )
    {
        $active_jobs{$j} = $1;
        $j++;
    }
}
@sorted_active_jobs = sort keys %active_jobs;
open(ACTIV,">$ACTIVE");
foreach (@sorted_active_jobs)
{
    print ACTIV " $_ $active_jobs{$_} \n";
}
close( ACTIV );
dbmclose( %JOBS );

$nexttime = scalar localtime( time + $MONTIME );
&mk_log("$this_mach: ".time." ...Finished Monitor. (Next at $nexttime)");
return ( 0 );
}

#-----

```

```

#                                     parse_pslol
#
# this subroutine returns a list of lists that contain the
#                                     PID PPID CMD
# for each of the executing commands. Must be passed
# the output of ps -fl (5.5 or 6.2) or ps -alxww (2.6)
#-----

sub parse_pslol
{
    &mk_log ("$this_mach: ".time." sub parse_pslol reached") if ($BUG);
    my( @ps_passed ) = @_;

    my( @pslol );

    # We don't know (or really care) what sort of machine this is
    # but we need to get PID and PPID from the ps. Assume that
    # there is a PID and PPID in the very first line. Find them
    # and get the appropriate information.

    $comd_loc = index $ps_passed[0], "COMMAND"; # solaris 2.6 version
    $comd_loc = index $ps_passed[0], "CMD"; # 5.5 or 6.2

    @firstline = split( ' ', $ps_passed[0] );
    for (0..$#firstline)
    {
        if ( $firstline[$_] =~ /^PID$/ )
        {
            $pid_loc = $_;
        }
        if ( $firstline[$_] =~ /^PPID$/ )
        {
            $ppid_loc = $_;
        }
    }

    if ($VERBOSE)
    {
        &mk_log("$this_mach: ".time." pid_loc is $pid_loc ppid_loc is $ppid_loc");
    }

    shift @ps_passed;
    $i = 0;
    foreach $psline ( @ps_passed )
    {
        $cmd = substr $psline, $comd_loc;
        chomp( $cmd );
        @linearray = split ' ', $psline;
        $pslol[$i] = [ $linearray[$pid_loc],
                      $linearray[$ppid_loc],
                      $cmd ];
        $i++;
    }

    return @pslol;
}

#-----
#                                     process_job
#
# This subroutine will be done after the THUNDER run is complete
#-----

```

```

sub process_job
{
    &mk_log ("${this_mach}: ".time." process_job reached") if ($BUG);
    $comp_label = "Complete_"."$run_label";
    rename ("${statuslocation}/${run_label}", "${statuslocation}/${comp_label}");

    open (COMPFILE, ">>${statuslocation}/${comp_label}")
        or die "cannot open ${statuslocation}/${comp_label}: $!";
    print COMPFILE scalar localtime, "    comp_label\n";
    close COMPFILE;

    # save files to case dir
    $case_dir = "$outlocation/S_Case_$run_case";
    unless( not @sav_files ) #@sav_files is from &read_mmmp
    {
        if(!( -d $case_dir ))
        {
            mkdir $case_dir, 0755;
        }
        opendir THISDIR, ".";
        @allfiles = readdir THISDIR;
        closedir THISDIR;
        # In MMMP.CTL the user specified which files should be saved
        for($k = 0; $k <= $#sav_files; $k++)
        {
            # Search for the specified files in current directory to save
            @savf = grep /$sav_files[$k]/, @allfiles;
            for($fyle = 0; $fyle <= $#savf; $fyle++)
            {
                $savf_rep = "$savf[$fyle]". "_Case_".
                    "${run_case}."_ "_Rep_". "${run_rep}";
                rename $savf[$fyle] , $savf_rep;
                system " cp $savf_rep $case_dir 2>> $outfile";
            }
        }
        #system " cp $sav_files $case_dir 2>> $outfile";
    }

    # selected compress files
    if( @compf and @tar_files )
    {
        opendir( D, '.' ) or die "cannot open dir: $dir_label\n" ;
        @dfiles = readdir D ;
        @files = ( ) ;
        foreach $c (@compf)
        {
            push( @files, grep( /^$c/, @dfiles ) ) ;
        }
        print "\nSelected compress files: @files\n";
        system "compress @files 2>&1" ;
    }

    # archive output-----
    for $x (@tar_files){ $x .= "*" ; } # need if file compressed
    $cmd = "tar cvf $tar_dir/$case.tar @tar_files >
        $tar_dir/$case.tar.list" ." 2>&1" ;

    # tar all files-----
    if( @tar_files )
    {
        print "\t$cmd\n" if $BUG ;
        system "$cmd" ;
        $tarbad++, print "\ntar unsuccessful, status: $?\n" if $? ;
    }
}

```

```

    }
    chdir ($runlocation) or die "cannot cd to $runlocation\n"; # to remove $EDIR
    system "/bin/rm -rf $dir_label" unless ($starbad or $NRMTMP); #del work dir
}# end of process

```

```

#-----
#
#                               read_machine
#
# Get the information from the machine.dat file
#
# Note: this information is read each time the monitor is run, so
# that changes in the machine.dat file will be incorporate in the
# next iteration.
#-----

```

```

sub read_machine
{
    &mk_log ("${this_mach}: ".time." sub read_machine reached") if ($BUG);
    my( $i ) = 0;
    open( MACH, "$MACH") or die "Cannot open $MACH ($!)";
    LOOP4: while ( <MACH> )
    {
        chomp;
        next LOOP4 if (/^\s*$/);          # Skip white space
        next LOOP4 if (/^\#/);           # Skip comments

        ( $name, $ncpus, $nlimit ) = split(' ');

        if ($name !~ /\S*$/)
        {
            &mk_log("${this_mach}: ".time." Problem with machine ($name)");
            next LOOP4;
        }

        if ($ncpus !~ /\d*$/)
        {
            &mk_log("${this_mach}: ".time." Problem with ncpus ($ncpus) for machine
($name)");
            next LOOP4;
        }

        if ($nlimit !~ /\d*\.\d*$/)
        {
            &mk_log("${this_mach}: ".time." Problem with nlimit ($nlimit) for machine
($name)");
            next LOOP4;
        }

        $machine_cpus{ $name } = $ncpus;
        $machine_name[ $i++] = $name;
        $machine_cpu_limit{ $name } = $nlimit;
    }# end of LOOP4
    close( MACH );
}

```

```

#-----
#
#                               read_mummp
#
# This script is controlled, oddly enough, by a control file which governs
# some very basic information needed to run this script and carry out the

```

```

# intended experimental design. The script can be renamed at any time, but
# this control file name is read into the variable $MMMPCTL in &define global.
# This variable can be changed of course and operation of the script will be
# unaffected.
#
# Get the information from the MMMP.CTL file.
#
# @cases      : Contains the different cases
# @INP_$case  : Contains the pertinent .dat files for that case
#-----

sub read_mmmp
{
    &mk_log ("${this_mach}: ".time." sub read_mmmp reached") if ($BUG);
    my($i) = 0 ;
    # Get the data from MMMP.CTL-----
    $thiswd = getcwd;
    #print"Read_mmmp: current dir: $this_mach $thiswd\n";
    open( MMMPCTL, "MMMPCTL") or die "Cannot open $MMMPCTL ($!)";

    LOOP1: while ( <MMMPCTL> )
    {
        next LOOP1 if ( /\s*$/ or /\s*#/ );      # skip blank lines, comments
        if( /\s*reps\s+(\d+)/ )                  # nr of reps to run
        {
            &mk_log ("${this_mach}: ".time." reps: $1") if ($VERBOSE) ;
            $NREPS = $1 ;
        }
        elsif( /\s*save\s+/ )                    # files to save in C_case
        {
            @sav_files = split; # the entire save line is put into @sav_files
            shift @sav_files;    # to remove the save
            #chop( $sav_files = $' ) ; #the $' grabs everything after the match
            &mk_log ("${this_mach}: ".time." SAVE: @sav_files") if ($VERBOSE) ;
        }
        elsif( /\s*com/i )
        {
            @compf = split ;      #the entire compress line is put into @compf
            shift @compf ;        # rm 'compress'
            &mk_log ("${this_mach}: ".time." COMPRESS: @compf") if ($VERBOSE) ;
        }
        elsif( /\s*tar\s+/ )
        {
            @tar_files = split ;
            shift @tar_files ;      # rm 'tar'
            $tar_dir = shift @tar_files ; # dir for tar file
            &mk_log ("${this_mach}: ".time." TAR: $tar_dir @tar_files") if
($VERBOSE) ;
        }
        elsif( /\s*individual\s+/ )              # Individual Rep Commands
        {
            LOOP8: while( <MMMPCTL> )
            {
                last LOOP8 if /\s*$/ ;          # blank line term
                next LOOP8 if /\s*#/ ;          # skip comments
                chop ;
                &mk_log ("${this_mach}: ".time." Command: $_") if ($VERBOSE) ;
                push ( @RepCommands, $_ );
            }
        }
        elsif( /\s*post/ )                      # post processor commands
        {
            LOOP2: while( <MMMPCTL> )
            {

```

```

last LOOP2 if /\s*$/ ;      # blank line term
next LOOP2 if /\s*#/ ;      # skip comments
chop ;
s/\$NUM_REPS/$num_runs/eg ;
s/\$ChecksOn/ /eg ;      # for now ...
if( />/ )
{
    s/$/ 2>> $OUTLS/ ; # stderr
}
else
{
    s/$/ >> $OUTLS 2>&1/ ; # stdout and stderr
}
&mk_log ("$this_mach: ".time." PP: $_") if ($VERBOSE) ;
push( @postpr, $_ ) ;
}
}
elseif( /\s*case\s+(\S+)/ )
{
    $cases[$i] = $1 ;
    $i++;
    $case = $1 ;
    &mk_log ("$this_mach: ".time." Case: $case") if ($VERBOSE);
    # if( not $nopr and $dup_case{$case}++ ) # chk dup case name
    # {
    #     print "\nDuplicate case name: $case\n" ;
    #     $C_stat{$case} = '(INP_ERR)' ;
    #     next ;
    # }
}
elseif ( /\s*THUNDER_Location\s+(\S+)/ ) { $THUNDERlocation = $1; }
elseif ( /\s*MMMP_Directory\s+(\S+)/ ) { $mmmplocation = $1; }
elseif ( /\s*Run_Directory\s+(\S+)/ ) { $runlocation = $1; }
elseif ( /\s*Out_Directory\s+(\S+)/ ) { $outlocation = $1; }
elseif ( /\s*Cases_Directory\s+(\S+)/ ) { $caseslocation = $1; }
elseif ( /\s*Library_Directory\s+(\S+)/ ) { $liblocation = $1; }
elseif ( /\s*Max_Run_Time\s+(\S+)/ ) { $max_run_time = $1; }
elseif ( /\s*Max_Failures\s+(\S+)/ ) { $max_failures = $1; }
elseif ( /\s*MaxOverage\s+(\S+)/ ) { $max_overage = $1; }
elseif ( /\s*Status_Directory\s+(\S+)/ ) { $statuslocation = $1; }
elseif ( /\s*c++_Location\s+(\S+)/ ) { $c++location = $1; }

}#end of LOOP1
unless( @cases )
{
    print "\nNo cases given\n" unless @cases ;
    $VALCK = 1 ;
}
$num_runs = ($#cases + 1) * $NREPS ;
close( MMMPCTL );

die "bin location: ($THUNDERlocation) is not good " if !( -d
$THUNDERlocation);
die "MMMP_dir: ($mmmplocation) is not good " if !( -d $mmmplocation);

# Make sure that we can write to the mmmplocation/..
$save_pwd = getcwd; # save present location
chdir $mmmplocation;
chdir "...";
$newpwd = getcwd;
die "data location/...: ($newpwd) is not good " if !( -w $newpwd);
chdir($save_pwd);

if ($VERBOSE)

```

```

{
    &mk_log("$this_mach: ".time." MMMP control file read and checked");

    # If Verbose, print execdata into xmon. This cannot be put into g.pl
    # because it would be before mk_log is defined.
    &mk_log("$this_mach: ".time." Directory containing code: ($MMMPDIR)");
    &mk_log("$this_mach: ".time." Executable name          : ($EXECNAME)");
}
}

#-----
#
#          spawn_a_run
#
# Spawn a single THUNDER run and deal with the returned value
#-----

sub spawn_a_run
{
    # Make sure that the bin directory is aprt of the path
    $ENV{'PATH'} = "$THUNDERlocation:" . $ENV{'PATH'};

    &mk_log ("$this_mach: ".time." sub spawn_a_run reached") if ($BUG);
    &mk_log ("$this_mach: ".time." edir is $EDIR\n") if ( $BUG );

    &mk_run_dir; #create the files and directories for this job

    #we should be in $dir_label from &mk_run_dir lets check if using -b option
    $test_wdir = getcwd;
    &mk_log ("$this_mach: ".time." Spawn: Directory is $test_wdir") if ( $BUG );

    $commandnum = 0;
    foreach $repcommand (@RepCommands)
    {
        print "Spawn: repcommand is $repcommand\n";
        $commandnum++;
        $outfile = 'cmd'."$commandnum" . "_" . "$run_label"; #shows up in
$dir_label
        # $exe_return = system " $repcommand > $outfile ";
        $exe_return = system " $repcommand > $outfile 2>&1 "; #STDOUT and STDERR
        if ($exe_return != 0)
        {
            &mk_log("$this_mach: ".time." Problem with $repcommand. Look in
$outfile");
            return $exe_return;
        }
    }

    &process_job;

    #-----
    # We now need to decrement the number of jobs on this machine
    #-----

    $JOBS{$this_mach}--;

    &mk_log("$this_mach: ".time." Spawn: $0 complete for $dir_label") if($BUG);
    $exe_return = 0;
    return (0);
}

sub write_csv
{
    LOOP1: while (defined($line = <RUNFILE> ))

```

```

{
    if ($line =~ /^"/)
    {
        $i = 0;
        print PUTFILE $line;
        $i++;
        LOOP2: while (defined($line = <RUNFILE> ))
        {
            $temp_line = join ', ' , split /\s+/ , $line ;
            print " $i";

            if ($i == 2)
            {
                print PUTFILE "$temp_line\n";
            }
            elsif (($i > 2)  && ($line =~ /\s*$/))
            {
                next LOOP1;
            }
            elsif ( $i > 3 )
            {
                print PUTFILE "$temp_line\n";
            }
            $i++;
        }#end of LOOP2
    }#end of if
}#end of LOOP1
}#end of write_csv

#-----
#
#                               writ_data_base
#
#       Initialize the database that will be used throughout mmmp
#-----

sub writ_data_base
{
    &mk_log ("$this_mach: ".time." sub writ_data_base reached") if ($BUG);
    dbmopen(%JOBS,"$MMMPDB",0644);
    $JOBS{"runs_to_do"} = $num_runs;
    $JOBS{"runs_succeeded"} = 0;
    $JOBS{"runs_failed"} = 0;
    $JOBS{"runs_launched"} = 0;
    $JOBS{"max_run_time"} = $max_run_time;
    $JOBS{"max_failures"} = $max_failures;
    $JOBS{"runs_to_do"} = $num_runs ; #num_runs is from &read_mmmp
    foreach $machine (@machine_name)
    {
        $JOBS{$machine} = 0;
    }
    dbmclose(%JOBS);
}

__END__

#####
#
#
# The following contains additional information that will hopefully help in
# deciphering the mmmp script and provide a clearer vision into its operation
#

```

```
#####
#
```

An excerpt from JMP showing the Plackett-Burman design matrix for 11 variables

```
1      1      1      1      1      1      1      1      1      1      1
-1     1     -1     1     1     1     -1     -1     -1     1     -1
-1    -1     1    -1     1     1     1     -1     -1     -1     1
```

An excerpt from DESIGN.CTL follows:

The first column contains the experimental design point (Case_one) and the remaining columns are flags that will be used by the cpp preprocessor to set the variables that have been included in the master data files.

```
Case_one ACS_HIGH_ AWACS_HIGH_ BLUEMSG_HIGH_ C3DEG_HIGH_
INTEGRATION_HIGH_
JSTARS_HIGH_ MAINSTAY_HIGH_ MEQ_HIGH_ PERCEPT_HIGH_ REDMSG_HIGH_ SREC_HIGH_
```

```
Case two ACS_LOW_ AWACS_HIGH_ BLUEMSG_LOW_ C3DEG_HIGH_ INTEGRATION_HIGH_
JSTARS_HIGH_ MAINSTAY_LOW_ MEQ_LOW_ PERCEPT_LOW_ REDMSG_HIGH_ SREC_LOW_
```

```
Case three ACS_LOW_ AWACS_LOW_ BLUEMSG_HIGH_ C3DEG_LOW_
INTEGRATION_HIGH_ JSTARS_HIGH_ MAINSTAY_HIGH_ MEQ_LOW_ PERCEPT_LOW_
REDMSG_LOW_ SREC_HIGH_
```

An excerpt from master_detect.dat showing the format for the cpp preprocessor commands:

```
#
#ifdef ACS_LOW_
#   define _BLUE_NO_AEW_ 0.75
#   define _BLUE_BLUE_AEW_ 0.313
#   define _BLUE_RED_AEW_ 0.188
#   define _BLUE_BOTH_AEW_ 0.75
#   define _RED_NO_AEW_ 0.75
#   define _RED_BLUE_AEW_ 0.188
#   define _RED_RED_AEW_ 0.313
#   define _RED_BOTH_AEW_ 0.75
#else
#   ifdef ACS_BASE_
#       define _BLUE_NO_AEW_ 1.00
#       define _BLUE_BLUE_AEW_ 1.25
#       define _BLUE_RED_AEW_ 0.75
#       define _BLUE_BOTH_AEW_ 1.00
#       define _RED_NO_AEW_ 1.00
#       define _RED_BLUE_AEW_ 0.75
#       define _RED_RED_AEW_ 1.25
#       define _RED_BOTH_AEW_ 1.00
#   else
#       ifdef ACS_HIGH_
#           define _BLUE_NO_AEW_ 1.25
#           define _BLUE_BLUE_AEW_ 1.56
#           define _BLUE_RED_AEW_ 0.94
#           define _BLUE_BOTH_AEW_ 1.25
#           define _RED_NO_AEW_ 1.25
#           define _RED_BLUE_AEW_ 0.94
#           define _RED_RED_AEW_ 1.56
#           define _RED_BOTH_AEW_ 1.25
#       endif
#   endif
#endif
#
```

An excerpt from detect.dat showing how variables (begin and end with _) can be put where you want the cpp directives to make changes depending upon the value of the flag that is passed by mk_CASES_dir:

```
DETECT.PROBS.205
MULT.FACTORS...BLUE...RED
  NO.AEW      BLUE_NO_AEW_ RED_NO_AEW_
  BLUE.AEW    BLUE_BLUE_AEW_ RED_BLUE_AEW_
  RED.AEW     BLUE_RED_AEW_  RED_RED_AEW_
  BOTH.AEW    BLUE_BOTH_AEW_ RED_BOTH_AEW_
```

BLUE.KILLER.AC

```
**make sure you are doing the monitor system call in the correct order.
# make it so that instead of just copying control.dat from the library to the
# run directory mmmmp will modify the report title to reflect the case/rep combo
# This title prints out in ttgraph.rpt
#-----
#this does not appear to work -- will have to work on this later
#      chomp $line2;
#      if ($line2 =~ /^\\s*REPORT.TITLE\\s+(S+)/)
#      {
#          $line2 =~ s/$1/"Case: $run_case Rep: $run_rep"/;
```

```
#####
##
```

The next series of scripts were used to extract MOE information from THUNDER's ttgraph.rpt. There is a master script and then individual scripts for each MOE.

```
#####
##
```

Get MOE

```
#####
##
```

```
#!/usr/local/bin/perl
```

```
#-----
#
#      get_moe
#
#      This script will make a list of all subdirectories within the
#      current directory. It will then launch other scripts in that
#      subdirectory. These other scripts search through any ttgraph* file
#      in this subdirectory and gathers the required information to be
#      to a file withing that directory. This is accomplished with the
#      -execute option. All of the MOE files will be stored in a directory
#      that either already exists or will be created by this script.
#      This script will also go through and remove these created files if
#      you so desire by using the -cleanup option.
#
#      To launch this script from the parent directory the following files
#      must be in the parent directory:
#
#      get_moe      get_airsup
```

```

# get_FLOT
# get_redloss
# get_blueloss
#
# As you add different MOEs you will need to make the following
# modifications to this script:
#
# 1. Add the system call for that script.
# 2. Add an unlink to the cleanup subroutine.
#
# Realize that this is an inefficient way to gather the MOEs because
# instead of one pass through the data it makes as many passes through
# the data as the number of MOEs. But it was easy to write and still
# works pretty fast. Feel free to modify and condense.
#-----
#-----
#                               Main
#-----
#-----
use Cwd;
#-----
# Global Variables
#-----
$MOE_dir = "MOE_dir";
my ($runlocation) = "";
$runlocation = getcwd;
#print "The current directory is: $runlocation\n";
if (!(defined($ARGV[0])))
{
    die "\nYou need to enter an arguement: $_\n\nFor help type: $0 -help\n\n" ;
}

while( defined($ARGV[0]) && $ARGV[0] =~ /^-/ )    # Get args
{
    $_ = shift ;
    if( /-e./ ) { $EXECUTE++; next; }
    if( /-c./ ) { $CLEANUP++; next ; }
    if( /-h./ ) { $HELP++; next ; }
    die "\nBad arg: $_\n\nFor help type '$0 -help'.\n" ;
}

#-----
# Make a list of subdirectories.
#-----
opendir( DATLOC, "$runlocation") or die "Cannot opendir runlocation/..($!)";
#get everything in directory except the hidden files
@List_All = grep !/^\.\/?$/ , readdir(DATLOC);
# Now eliminate everything that isn't a directory
foreach $listing (@List_All)
{
    if ( -d $listing)
    {
        push @List_Dirs, $listing;
    }
}
print "@List_Dirs\n";
closedir( DATLOC );

#-----
# Make a directory to store all of the MOE files
#-----
unless ( -d $MOE_dir )
{

```

```

    mkdir $MOE_dir, 0777 or die "cannot make dir ($MOE_dir)";
}

#-----
# None of these come back, but put the exit 0 there.
#-----
&cleanup, exit 0 if $CLEANUP;
&execute, exit 0 if $EXECUTE;
&helpopt, exit 0 if $HELP;

exit(0);

#-----
#
#                               SUBROUTINES
#-----
#
#                               EXECUTE
#-----
# Sent here by main, post processes the ttgraph reports.
#-----
sub execute
{
    for($i = 0; $i <= $#List_Dirs; $i++)
    {
        chdir "$List_Dirs[$i]";
        $newpwd = getcwd;
        print"$newpwd\n";
        #print"$List_Dirs[$i]\n";
        system"./get_FLOT $i";
        system"./get_blue_loss $i";
        system"./get_red_loss $i";
        system"./get_tahait $i";
        system"./get_airsup $i";
        chdir"$runlocation";
    }
}

#-----
#
#                               CLEANUP
#-----
# Cleans up the subdirectory, removing:
#   MOE files
#-----
sub cleanup
{
    #for($i = 0; $i <= $#List_Dirs; $i++)
    {
        print"sub cleanup reached\n";
        chdir "$MOE_dir";
        unlink <flot_file*>           if (-e <flot_file*>);
        unlink <blue_loss_file*>       if (-e <blue_loss_file*>);
        unlink <red_loss_file*>        if (-e <red_loss_file*>);
        unlink <tahait_file*>          if (-e <tahait_file*>);
        unlink <air_sup_file*>         if (-e <air_sup_file*>);

        chdir"$runlocation";
    }
}

#-----
#
#                               HELP
#-----

```



```

my ($runlocation) = "";
$end_war = 31.00;
$reduce_level = 0.2;

#-----
#   Get the script arguments
#-----
while( defined($ARGV[0]) )    # Get args
{
    $j = shift ;
}

#-----
#       Get the ttgraph.rpt files from the following directory
#-----
$runlocation = getcwd;
$air_sup_file = "air_sup_file".$j";

#-----
#       Make a list of all ttgraph.rpt files in this directory
#-----
opendir( DATLOC, "$runlocation")
    or die "Cannot opendir runlocation/..($!)";
@List_Files = grep { /ttgraph.rpt_*/ } readdir(DATLOC);
#print "@List_Files\n";
closedir( DATLOC );
# can't really figure out how to get these to print in the file.
print OUTFILE2 "-----\n";
print OUTFILE2 "    Design Point          Time (days)  \n";

#-----
# Go through the files one at a time
#-----
foreach $run_file (@List_Files)
{
    open (INFILE2, "$runlocation/$run_file")
        or die "can't open $run_file to read : $!";
    open (OUTFILE2, ">>../$MOE_dir/$air_sup_file")
        or die "cannot open $air_sup_file : $!";
    &search_air_sup;
    close (OUTFILE2);
    close (INFILE2);
}#end of foreach

#-----
#               sub search_air_sup
#-----
sub search_air_sup
{
    #print"Sub search_air_sup reached\n";
    my ($templ) = 0;
    # all files should have format filename = ttgraph.rpt_Case_8_Rep_1
    # strip off ttgraph.rpt_
    $run_file =~ s/ttgraph.rpt_//;
    LOOP1: while ($line2 = <INFILE2> )
    {
        if ($line2 =~ /"Red Sorties \\/Cycle/)
        {
            #print"$line2\n";
            # skip four lines one is blank (hence only 3 <INFILE2>)
            $line2 = <INFILE2>;
            $line2 = <INFILE2>;
            $line2 = <INFILE2>;
            $line2 = <INFILE2>;
        }
    }
}

```



```

#-----
#   Get the script arguments
#-----
while( defined($ARGV[0]) ) # Get args
{
    $j = shift ;
}

#-----
#   Get the ttgraph.rpt files from the following directory #-----
#-----
#   Get the files of interest from the following directory
$runlocation = getcwd;
$blue_loss_file = "blue_loss_file"."$j";

#-----
#   Make a list of all ttgraph.rpt files in this directory
#-----
opendir( DATLOC, "$runlocation") or die "Cannot opendir runlocation/..($!)";
@List_Files = grep { /ttgraph.rpt_*/ } readdir(DATLOC);
#print "@List_Files\n";
closedir( DATLOC );
# can't really figure out how to get these to print in the file.
print OUTFILE2 "-----\n";
print OUTFILE2 "    Design Point          Time (days) \n";

#-----
#   Go through the files one at a time
#-----
foreach $run_file (@List_Files)
{
    open (INFILE2, "$runlocation/$run_file")
        or die "can't open $run_file to read : $!";
    open (OUTFILE2, ">>../$MOE_dir/$blue_loss_file")
        or die "cannot open $blue_loss_file : $!";
    &search_blue_loss;
    close (OUTFILE2);
    close (INFILE2);
}#end of foreach

#-----
#   sub search_blue_loss
#-----
sub search_blue_loss
{
    #print"Sub search_blue_loss reached\n";
    my ($temp1) = 0;
    # all files should have format filename = ttgraph.rpt_Case_8_Rep_1
    # strip off ttgraph.rpt_
    $run_file =~ s/ttgraph.rpt_//;
    LOOP1: while ($line2 = <INFILE2> )
    {
        if ($line2 =~ /"Cum. Blue A-A Losses"/)
        {
            # skip three lines one is blank (hence only 2 <INFILE2>)
            $line2 = <INFILE2>;
            $line2 = <INFILE2>;
            LOOP2: while (defined($line2 = <INFILE2> ))
            {
                @blueline = split /\s+/, $line2 ;
                if ( $blueline[1] >= $end_war )
                {
                    print OUTFILE2 " $run_file $blueline[1] $blueline[2] $blueline[3]\n";
                }
            }
        }
    }
}

```

```

#-----
# This ttgraph.rpt has two sections with this
# information. One for each blue
# and red. This will skip the second table.
# After all, they are mirror images of each other.
#-----
last LOOP1;
    }
  }#end of LOOP2
}#end of if
}#end of LOOP1
}#end of write_csv

__END__

#####
##
##                                get FLOT
#####

#!/usr/local/bin/perl
use Cwd;
print"\tCalculating FLOT MOE\n";
#-----
#                                get_FLOT
#
# This script searches through a ttgraph.rpt until it finds the line
# "Cum. FLOT Movement (km)". It then starts looking for the point where
# column 2 (FLOT) makes its first increase (-36.0 to -34.0 is an
# an increase). This line is then printed out to a file.
#-----
#-----
# Variables
#-----
$MOE_dir = "MOE_dir";
my($j) = 0;
my ($runlocation) = "";
#-----
# Get the script arguments
#-----
while( defined($ARGV[0]) ) # Get args
{
  $j = shift ;
}
#-----
# Get the ttgraph.rpt files from the following directory #-----
$runlocation = getcwd;
#print " runlocation is $runlocation\n";
#print " MOE_dir is : $MOE_dir\n\n";
$flot_file = "flot_file".$j;
#-----
# Make a list of all ttgraph.rpt files in this directory
#-----
opendir( DATLOC, "$runlocation") or die "Cannot opendir runlocation/..($!)";
@List_Files = grep { /ttgraph.rpt_*/ } readdir(DATLOC);
#print "@List_Files\n";
closedir( DATLOC );
# can't really figure out how to get these to print in the file.

```

```

print OUTFILE "-----\n";
print OUTFILE "      Design Point          Time (days) \n";

#-----
# Go through the files one at a time
#-----
    foreach $run_file (@List_Files)
    {
#print "$run_file\n";
        open (INFILE, "$runlocation/$run_file") or warn "can't open $run_file to
read in $runlocation: $!";

        open (OUTFILE, ">>../$MOE_dir/$flot_file") or die "cannot open
$flot_file : $!";

            &search_flot;

        close (OUTFILE);
        close (INFILE);
    }#end of DIRLOOP
#-----
#               sub search_flot
#-----
sub search_flot
{
    my ($stemp1) = 0;
    # all files should have format filename = ttgraph.rpt_Case_8_Rep_1
    # strip off ttgraph.rpt_
    $run_file =~ s/ttgraph.rpt_//;

    LOOP1: while ($line = <INFILE> )
    {
        #print"$line";
        if ($line =~ /^"Cum. FLOT/")
        {
            #print "$run_file\n";
            #print OUTFILE " $run_file          \n";

            LOOP2: while (defined($line = <INFILE> ))
            {
                # skip three lines one is blank (hence only 2 <INFILE>)
                $line = <INFILE>;
                $line = <INFILE>;

#print"$line";

                @old_flot = split /\s+/, ($line = <INFILE>) ;
                # now find where the FLOT changes direction
                while (defined($line = <INFILE> ))
                {
                    @current_flot = split /\s+/, $line;

#print"$line";

                    if ( $current_flot[2] > $old_flot[2])
                    {
                        #print" $current_flot[1] ,  $old_flot[1]\n";
                        print OUTFILE "      $run_file          $current_flot[1]\n";

#-----
# This report has two sections where there is
# cumulative FLOT movement. One for each red
# and blue. This will skip the second table.
# After all, they are mirror images of each other.

```

```

#-----
last LOOP1;
}
@old_flot = @current_flot;
#print OUTFILE " $run_file never neutralized FLOT\n";
}
}#end of LOOP2
}#end of if
}#end of LOOP1
}#end of write_csv

__END__

```

```

#####
##
get_redloss
#####
##
#!/usr/local/bin/perl

#-----
#
# get_redloss
#
# This script searches through a ttgraph.rpt until it finds the line
# "Cum. Red A-A Losses". It then starts looking for the row designated
# by the variable $end_war. This line is then printed out to a file.
#-----
use Cwd;
print"\tCalculating Red Losses MOE\n";
#-----
# Variables
#-----
$MOE_dir = "MOE_dir";
my($j) = 0;
my ($runlocation) = "";
#-----
# Get the script arguments
#-----
while( defined($ARGV[0]) ) # Get args
{
    $j = shift ;
}
$end_war = 31.00;

#-----
# Get the ttgraph.rpt files from the following directory #-----
$runlocation = getcwd;
$red_loss_file = "red_loss_file".$j";

#-----
# Make a list of all ttgraph.rpt files in this directory
#-----
opendir( DATLOC, "$runlocation") or die "Cannot opendir runlocation/..($!)";
@List_Files = grep { /ttgraph.rpt_*/ } readdir(DATLOC);
#print "@List_Files\n";
closedir( DATLOC );
# can't really figure out how to get these to print in the file.
print OUTFILE2 "-----\n";
print OUTFILE2 "    Design Point          Time (days) \n";

```

```

#-----
# Go through the files one at a time
#-----
foreach $run_file (@List_Files)
{
    open (INFILE2, "$runlocation/$run_file")
        or die "can't open $run_file to read : $!";
    open (OUTFILE2, ">>../$MOE_dir/$red_loss_file")
        or die "cannot open $red_loss_file : $!";
    &search_red_loss;
    close (OUTFILE2);
    close (INFILE2);
}#end of foreach

#-----
#                               sub search_red_loss
#-----
sub search_red_loss
{
    #print "Sub search_red_loss reached\n";
    my ($temp1) = 0;
    # all files should have format filename = ttgraph.rpt_Case_8_Rep_1
    # strip off ttgraph.rpt_
    $run_file =~ s/ttgraph.rpt_//;
    LOOP1: while ($line2 = <INFILE2> )
    {
        if ($line2 =~ /"Cum. Red A-A Losses"/)
        {
            # skip three lines one is blank (hence only 2 <INFILE2>)
            $line2 = <INFILE2>;
            $line2 = <INFILE2>;
            LOOP2: while (defined($line2 = <INFILE2> ))
            {
                @redline = split /\s+/ , $line2 ;
                if ( $redline[1] >= $end_war )
                {
                    print OUTFILE2 " $run_file $redline[1]  $redline[2]
$redline[3]\n";

                    #-----
                    # This ttgraph.rpt has two sections with this
                    # information. One for each blue
                    # and red. This will skip the second table.
                    # After all, they are mirror images of each other.
                    #-----
                    last LOOP1;
                }
            }
        }
    }#end of LOOP2
}#end of if
}#end of LOOP1
}#end of

__END__

#####
##
                        get TAHAIT
#####
##

#!/usr/local/bin/perl

#-----

```

```

#                                     get_TAHAIT
#
#   This script searches through a ttgraph.rpt until it finds the line
#   "Cum. Red TANK ...".
#-----
use Cwd;
print"\tCalculating TAHAIT MOE\n";
#-----
#       Variables
#-----
$MOE_dir = "MOE_dir";
my($j) = 0;
my ($runlocation) = "";
$end_war = 31.00;          #could have this sent in by the user would require
                           #another shift in get script arguments section

#-----
#   Get the script arguments
#-----
while( defined($ARGV[0]) ) # Get args
{
    $j = shift ;
}

#-----
#       Get the ttgraph.rpt files from the following directory #-----
$runlocation = getcwd;
#print " runlocation is $runlocation\n";
#print "   MOE_dir is : $MOE_dir\n\n";
$tahait_file = "tahait_file".$j;

#-----
#       Make a list of all ttgraph.rpt files in this directory
#-----
opendir( DATLOC, "$runlocation") or die "Cannot opendir runlocation/..($!)";
@List_Files = grep { /ttgraph.rpt_*/ } readdir(DATLOC);
#print "@List_Files\n";
closedir( DATLOC );
# can't really figure out how to get these to print in the file.
print OUTFILE "-----\n";
print OUTFILE "   Design Point          Time (days) \n";

#-----
#   Go through the files one at a time
#-----
foreach $run_file (@List_Files)
{
    #print "$run_file\n";
    open (INFILE, "$runlocation/$run_file") or warn "can't open $run_file to
read in $runlocation: $!";

    open (OUTFILE, ">>../$MOE_dir/$tahait_file") or die "cannot open
$tahait_file : $!";

    &search_tahait;

    close (OUTFILE);
    close (INFILE);
}#end of DIRLOOP

#-----
#               sub search_tahait
#-----
sub search_tahait

```

```

{
my ($templ) = 0;
# all files should have format filename = ttgraph.rpt_Case_8_Rep_1
# strip off ttgraph.rpt_
$run_file =~ s/ttgraph.rpt_//;

LOOP1: while ($line = <INFILE> )
{
    #print"$line";
    if ($line =~ /^"Cum. Red TANK/)
    {
        #print "$run_file\n";
        #print OUTFILE " $run_file      \n";

        LOOP2: while (defined($line = <INFILE> ))
        {
            # skip three lines one is blank (hence only 2 <INFILE>)
            $line = <INFILE>;
            $line = <INFILE>;

#print"$line";

            @day1_tahait = split /\s+/ , ($line = <INFILE>) ;
            # now find the TAHAIT value for $end_war time
            while (defined($line = <INFILE> ))
            {
                @current_tahait = split /\s+/ , $line;

#print"$line";

                if ( $current_tahait[1] >= $end_war)
                {
                    #print" $current_tahait[1] , $day1_tahait[1]\n";
                    $red_tahait = $day1_tahait[2] - $current_tahait[2];
                    $blue_tahait = $day1_tahait[3] - $current_tahait[3];
                    # format the print output with a sprintf --someday
                    print OUTFILE "   $run_file   $red_tahait   $blue_tahait\n";

                    last LOOP1;
                }#end of if
            }#end of while
        }#end of LOOP2
    }#end of if
}#end of LOOP1
}#end of write_csv

__END__

```


Appendix D. Validation Results

Table 43. Plackett-Burman Validation Design Flag Matrix

Case_1	ACS_HIGH_	AWACS_HIGH_	BLUEMSG_HIGH_	C3DEG_HIGH_	INTEGRATION_HIGH_	JSTARS_HIGH_	MAINSTAY_HIGH_	MEQ_HIGH_	PERCEPT_HIGH_	REDMSG_HIGH_	SREC_HIGH_
Case_2	ACS_LOW_	AWACS_HIGH_	BLUEMSG_LOW_	C3DEG_HIGH_	INTEGRATION_HIGH_	JSTARS_HIGH_	MAINSTAY_LOW_	MEQ_LOW_	PERCEPT_LOW_	REDMSG_HIGH_	SREC_LOW_
Case_3	ACS_LOW_	AWACS_LOW_	BLUEMSG_HIGH_	C3DEG_LOW_	INTEGRATION_HIGH_	JSTARS_HIGH_	MAINSTAY_HIGH_	MEQ_LOW_	PERCEPT_LOW_	REDMSG_LOW_	SREC_HIGH_
Case_4	ACS_HIGH_	AWACS_LOW_	BLUEMSG_LOW_	C3DEG_HIGH_	INTEGRATION_LOW_	JSTARS_HIGH_	MAINSTAY_HIGH_	MEQ_HIGH_	PERCEPT_LOW_	REDMSG_LOW_	SREC_LOW_
Case_5	ACS_LOW_	AWACS_HIGH_	BLUEMSG_LOW_	C3DEG_LOW_	INTEGRATION_HIGH_	JSTARS_LOW_	MAINSTAY_HIGH_	MEQ_HIGH_	PERCEPT_HIGH_	REDMSG_LOW_	SREC_LOW_
Case_6	ACS_LOW_	AWACS_LOW_	BLUEMSG_HIGH_	C3DEG_LOW_	INTEGRATION_LOW_	JSTARS_HIGH_	MAINSTAY_LOW_	MEQ_HIGH_	PERCEPT_HIGH_	REDMSG_HIGH_	SREC_LOW_
Case_7	ACS_LOW_	AWACS_LOW_	BLUEMSG_LOW_	C3DEG_HIGH_	INTEGRATION_LOW_	JSTARS_LOW_	MAINSTAY_HIGH_	MEQ_LOW_	PERCEPT_HIGH_	REDMSG_HIGH_	SREC_HIGH_
Case_8	ACS_HIGH_	AWACS_LOW_	BLUEMSG_LOW_	C3DEG_LOW_	INTEGRATION_HIGH_	JSTARS_LOW_	MAINSTAY_LOW_	MEQ_HIGH_	PERCEPT_LOW_	REDMSG_HIGH_	SREC_HIGH_
Case_9	ACS_HIGH_	AWACS_HIGH_	BLUEMSG_LOW_	C3DEG_LOW_	INTEGRATION_LOW_	JSTARS_HIGH_	MAINSTAY_LOW_	MEQ_LOW_	PERCEPT_HIGH_	REDMSG_LOW_	SREC_HIGH_
Case_10	ACS_HIGH_	AWACS_HIGH_	BLUEMSG_HIGH_	C3DEG_LOW_	INTEGRATION_LOW_	JSTARS_LOW_	MAINSTAY_HIGH_	MEQ_LOW_	PERCEPT_LOW_	REDMSG_HIGH_	SREC_LOW_
Case_11	ACS_LOW_	AWACS_HIGH_	BLUEMSG_HIGH_	C3DEG_HIGH_	INTEGRATION_LOW_	JSTARS_LOW_	MAINSTAY_LOW_	MEQ_HIGH_	PERCEPT_LOW_	REDMSG_LOW_	SREC_HIGH_
Case_12	ACS_HIGH_	AWACS_LOW_	BLUEMSG_HIGH_	C3DEG_HIGH_	INTEGRATION_HIGH_	JSTARS_LOW_	MAINSTAY_LOW_	MEQ_LOW_	PERCEPT_HIGH_	REDMSG_LOW_	SREC_LOW_

Plackett-Burman Screening Data vs. Metamodel Comparison MOE#1 Days to Achieve Air Superiority				
Run	THUNDER	Metamodel	Diff Sq	MAPE Calc
1	26.82	27.80675	0.967	0.04
2	29.27	29.11425	0.023	0.01
3	28.84	28.94425	0.011	0.00
4	27.74	27.20675	0.286	0.02
5	29.73	30.58175	0.720	0.03
6	29.90	30.20425	0.093	0.01
7	29.00	29.48675	0.237	0.02
8	27.41	27.41425	0.000	0.00
9	26.97	27.21675	0.061	0.01
10	27.45	28.15925	0.497	0.03
11	28.44	29.30675	0.744	0.03
12	28.80	29.26425	0.216	0.02
				MSPR = 0.321 RMSPR = 0.242 MAPE = 1.680

Plackett-Burman Screening Data vs. Metamodel Comparison MOE#1 Days to Halt FLOT				
Run	THUNDER	Metamodel	Diff Sq	MAPE Calc
1	10.57	10.61	0.002	0.00
2	10.66	10.60	0.003	0.01
3	11.03	10.81	0.049	0.02
4	10.59	10.73	0.018	0.01
5	10.76	10.64	0.015	0.01
6	10.72	10.77	0.002	0.00
7	10.68	10.60	0.006	0.01
8	10.94	10.81	0.017	0.01
9	10.64	10.56	0.006	0.01
10	10.59	10.66	0.004	0.01
11	10.83	10.75	0.007	0.01
12	10.75	10.64	0.012	0.01
				MSPR = 0.012 RMSPR = 0.047 MAPE = 0.166

Plackett-Burman Screening Data vs. Metamodel Comparison MOE#3 TAHAIT				
Run	THUNDER	Metamodel	Diff Sq	MAPE Calc
1	60816.58	60849.69	1096.54	0.0005
2	60291.21	60642.31	123269.81	0.0058
3	61304.15	61564.80	67940.51	0.0043
4	60390.88	60642.31	63216.04	0.0042
5	60475.27	60599.96	15547.20	0.0021
6	60509.88	60020.76	239234.27	0.0081
7	61349.88	61014.02	112804.76	0.0055
8	61659.73	61729.13	4815.78	0.0011
9	60998.85	60756.13	58914.07	0.0040
10	61448.94	61221.50	51727.95	0.0037
11	62176.33	61635.56	292432.63	0.0087
12	60955.90	60506.39	202057.26	0.0074
				MSPR = 102754.733 RMSPR = 136.968 MAPE = .0841

Plackett-Burman Screening Data vs. Metamodel Comparison MOE#4 Exchange Ratio				
Run	THUNDER	Metamodel	Diff Sq	MAPE Calc
1	1.14	1.08	0.004	0.06
2	1.12	1.16	0.002	0.04
3	1.12	1.13	0.000	0.01
4	1.15	1.12	0.001	0.03
5	1.10	1.15	0.002	0.04
6	1.08	1.14	0.004	0.06
7	1.10	1.12	0.001	0.02
8	1.19	1.11	0.007	0.07
9	1.11	1.06	0.002	0.04
10	1.15	1.12	0.001	0.02
11	1.12	1.14	0.000	0.02
12	1.03	1.12	0.007	0.08
				MSPR = 0.0026 RMSPR = 0.0219 MAPE = 0.7452

Plackett-Burman Validation Data vs. Metamodel Comparison MOE#1 Days to Achieve Air Supremacy				
Run	THUNDER	Metamodel	Diff Sq	MAPE Calc
1	27.19	28.32	1.28	0.04
2	29.23	28.97	0.07	0.01
3	28.55	28.84	0.09	0.01
4	27.73	28.09	0.13	0.01
5	29.20	29.59	0.15	0.01
6	29.33	29.47	0.02	0.00
7	28.67	29.16	0.24	0.02
8	27.90	28.08	0.03	0.01
9	27.50	28.10	0.36	0.02
10	28.23	28.49	0.07	0.01
11	28.23	29.03	0.63	0.03
12	28.73	28.93	0.04	0.01
MSPR = 0.258				
RMSPR = 0.217				
MAPE = 1.511				

Plackett-Burman Validation Data vs. Metamodel Comparison MOE#2 Days to Halt FLOT				
Run	THUNDER	Metamodel	Diff Sq	MAPE Calc
1	10.67	10.65	0.000	0.001
2	10.80	10.68	0.016	0.012
3	10.67	10.72	0.003	0.005
4	10.58	10.71	0.017	0.012
5	10.68	10.65	0.001	0.003
6	10.66	10.69	0.001	0.003
7	10.68	10.65	0.001	0.003
8	10.59	10.72	0.018	0.013
9	10.58	10.64	0.004	0.006
10	10.73	10.68	0.003	0.005
11	10.78	10.70	0.007	0.008
12	10.58	10.65	0.005	0.007
MSPR = 0.006				
RMSPR = 0.0337				
MAPE = 0.116				

Plackett-Burman Validation Data vs. Metamodel Comparison MOE#3 TAHAIT					
Run	THUNDER	Metamodel	Diff Sq	MAPE Calc	
1	60613.93939	60871.60	66388.87	0.0043	
2	60746.18182	60868.24	14898.42	0.0020	
3	61013.10345	61229.15	46678.17	0.0035	
4	60516.875	60868.24	123458.00	0.0058	
5	60599.76667	60771.82	29600.80	0.0028	
6	59989.64516	60526.28	287977.24	0.0089	
7	61265.03226	61013.42	63310.29	0.0041	
8	61558.62069	61370.97	35212.07	0.0030	
9	60479.3125	60856.01	141897.54	0.0062	
10	61144	61098.18	2099.31	0.0007	
11	61564.21875	61339.78	50371.33	0.0036	
12	60565.875	60740.63	30538.18	0.0029	
					MSPR = 74369.18 RMSPR = 116.524 MAPE = 0.073

Plackett-Burman Validation Data vs. Metamodel Comparison MOE#4 Exchange Ratio					
Run	THUNDER	Metamodel	Diff Sq	MAPE Calc	
1	1.12	1.10	0.000	0.019	
2	1.13	1.14	0.000	0.011	
3	1.13	1.13	0.000	0.001	
4	1.12	1.12	0.000	0.004	
5	1.12	1.14	0.000	0.014	
6	1.12	1.13	0.000	0.010	
7	1.11	1.13	0.000	0.017	
8	1.15	1.11	0.001	0.029	
9	1.12	1.10	0.001	0.025	
10	1.14	1.12	0.000	0.015	
11	1.11	1.13	0.001	0.024	
12	1.15	1.12	0.001	0.031	
					MSPR = 0.0005 RMSPR = 0.009 MAPE = 0.307

Table 44. Plackett-Burman Validation Variable Design Levels

ACS	AWACS	BLUEMSG	C3DEG	INT	JSTARS	MINSTY	MEQ	PRCPT	REDMSG	SREC
1/2	1/2	1/2	1/2	1/2	1/3	0	1/2	1/2	1/2	1/2
- 1/2	1/2	- 1/2	1/2	1/2	1/3	- 1/2	- 1/2	- 1/2	1/2	- 1/2
- 1/2	- 1/2	1/2	- 1/2	1/2	1/3	0	- 1/2	- 1/2	- 1/2	1/2
1/2	- 1/2	- 1/2	1/2	- 1/2	1/3	0	1/2	- 1/2	- 1/2	- 1/2
- 1/2	1/2	- 1/2	- 1/2	1/2	- 2/3	0	1/2	1/2	- 1/2	- 1/2
- 1/2	- 1/2	1/2	- 1/2	- 1/2	1/3	- 1/2	1/2	1/2	1/2	- 1/2
- 1/2	- 1/2	- 1/2	1/2	- 1/2	- 2/3	0	- 1/2	1/2	1/2	1/2
1/2	- 1/2	- 1/2	- 1/2	1/2	- 2/3	- 1/2	1/2	- 1/2	1/2	1/2
1/2	1/2	- 1/2	- 1/2	- 1/2	1/3	- 1/2	- 1/2	1/2	- 1/2	1/2
1/2	1/2	1/2	- 1/2	- 1/2	- 2/3	0	- 1/2	- 1/2	1/2	- 1/2
- 1/2	1/2	1/2	1/2	- 1/2	- 2/3	- 1/2	1/2	- 1/2	- 1/2	1/2
1/2	- 1/2	1/2	1/2	1/2	- 2/3	- 1/2	- 1/2	1/2	- 1/2	- 1/2

Validation Design Input Variable Levels

Table 45. C2 Aircraft Design Levels

	LOW LEVEL	MID LEVEL	HIGH LEVEL
AWACS	9	12	15
JSTARS	4	6	7
MAINSTAY	1	2	2

Table 46. Integration Degrade Levels

MULTIPLIER	LOW LEVEL	BASELINE	HIGH LEVEL
INTEGRATION.DEGRADE	0.63	0.75	0.88

Table 47. Message Capacity Design Levels – *typeC3.dat*

Message Capacity Design Levels			
	Low	Baseline	High
Blue Unit C3 Fac	132	150	169
Blue CMD C3 Fac	263	300	338
Blue Log Fac C3 Fac	88	100	113
Red Unit C3 Fac	132	150	169
Red CMD C3 Fac	263	300	338
Red Log Fac C3 Fac	88	100	113

Table 48. Unit Strength C3 Degrade Design Levels

C3 Degrade Design Levels			
Delay	Low	Baseline	High
1.00	0	0	0
1.70	11.25	10.00	8.75
2.90	16.88	15.00	13.13
4.25	22.50	20.00	17.50
5.50	28.13	25.00	21.88

Table 49. ACS Multiplier Design Levels

ACS MULTIPLIER	LOW LEVEL		MID LEVEL		HIGH LEVEL	
	BLUE	RED	BLUE	RED	BLUE	RED
NO.AEW	0.88	0.88	1.00	1.00	1.13	1.13
BLUE.AEW	0.63	0.38	1.25	0.75	1.41	0.84
RED.AEW	0.38	0.63	0.75	1.25	0.84	1.41
BOTH.AEW	0.88	0.88	1.00	1.00	1.13	1.13

Table 50. Design Levels for Mean Error Quantities

	Design Levels		
	Low	Mid	High
Force Ratio	0.65	0.75	0.85
Unit Strength	85	75	65

Msg Processing	85	75	65
-----------------------	----	----	----

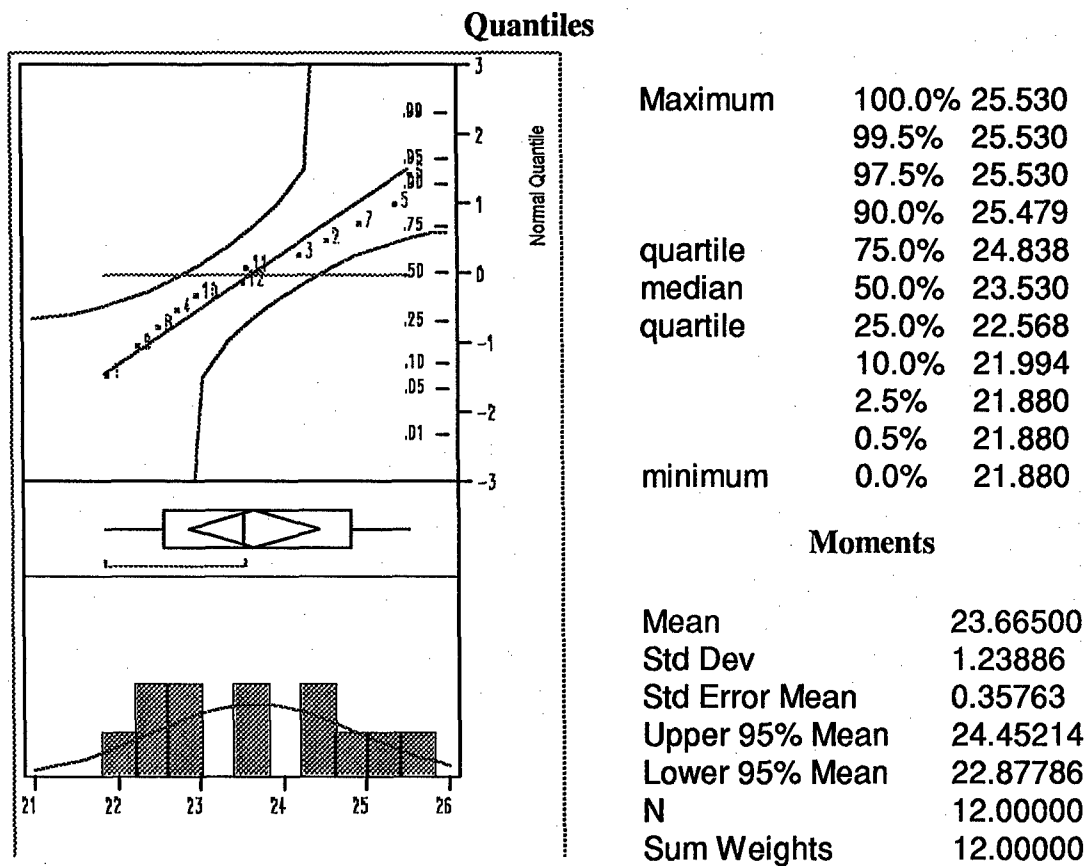
Table 51. Perception Degrade Design Levels

	LOW LEVEL	BASELINE	HIGH LEVEL
Perception Degrade	35 %	25 %	15 %

Appendix E. Data Normality

This appendix contains three different representations of the data for the different MOEs: Normal Quantile Plot, Outlier Box plot and Histogram of the data with a normal curve superimposed. To test for normality, JMP® uses the Shapiro_Wilk test when $n \leq 2000$. If the p-value reported is less than some α then the conclusion is that the distribution is not normal.

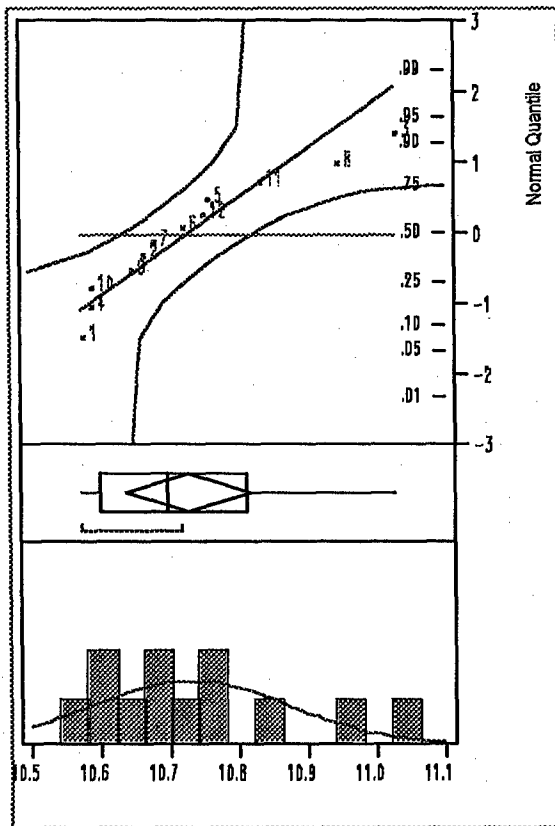
MOE #1: Days to Achieve Air Superiority – Plackett-Burman



Test for Normality Shapiro-Wilk W Test

W	Prob<W
0.951249	0.6074

MOE #2: Days to Halt FLOT – Plackett-Burman



Quantiles

maximum	100.0%	11.029
	99.5%	11.029
	97.5%	11.029
	90.0%	11.003
quartile	75.0%	10.814
median	50.0%	10.698
quartile	25.0%	10.600
	10.0%	10.577
	2.5%	10.574
	0.5%	10.574
minimum	0.0%	10.574

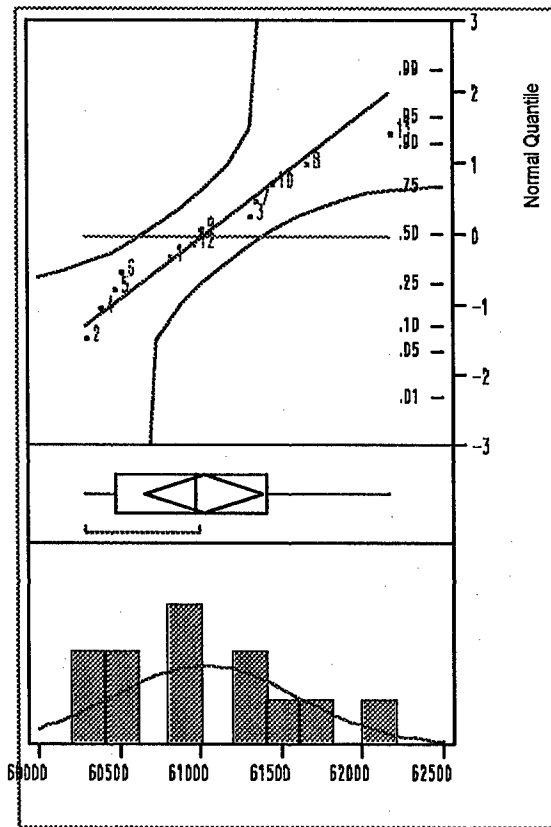
Moments

Mean	10.72976
Std Dev	0.14386
Std Error Mean	0.04153
Upper 95% Mean	10.82117
Lower 95% Mean	10.63836
N	12.00000
Sum Weights	12.00000

Test for Normality Shapiro-Wilk W Test

W	Prob<W
0.905491	0.1781

MOE #3: RedTAHAIT – Plackett-Burman



Quantiles

maximum	100.0%	62176
	99.5%	62176
	97.5%	62176
	90.0%	62021
	75.0%	61424
quartile	50.0%	60977
quartile	25.0%	60484
	10.0%	60321
	2.5%	60291
	0.5%	60291
	0.0%	60291
minimum		

Moments

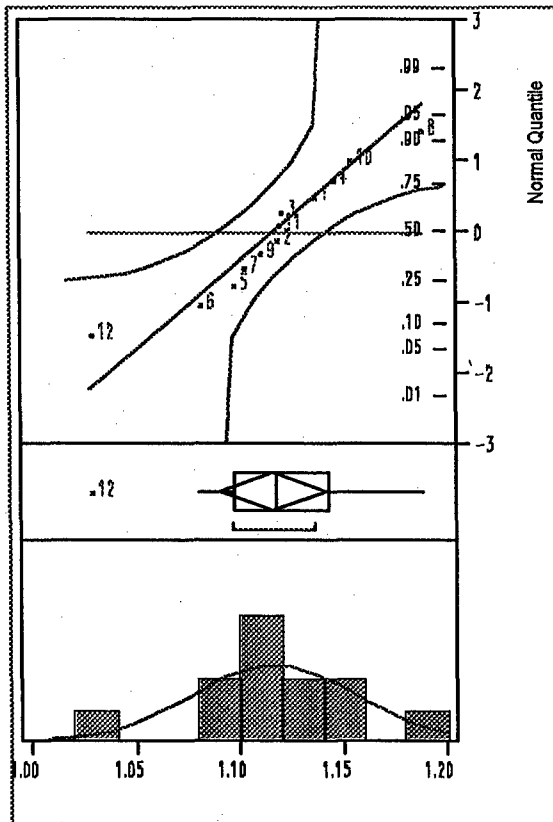
Mean	61031.47
Std Dev	575.89
Std Error Mean	166.25
Upper 95% Mean	61397.37
Lower 95% Mean	60665.56
N	12.00
Sum Weights	12.00

Test for Normality

Shapiro-Wilk W Test

W	Prob<W
0.952434	0.6236

MOE #4: Exchange Ratio – Plackett-Burman



Quantiles

maximum	100.0%	1.1889
	99.5%	1.1889
quartile	97.5%	1.1889
	90.0%	1.1784
quartile	75.0%	1.1440
	50.0%	1.1189
quartile	25.0%	1.0988
	10.0%	1.0445
minimum	2.5%	1.0287
	0.5%	1.0287
minimum	0.0%	1.0287

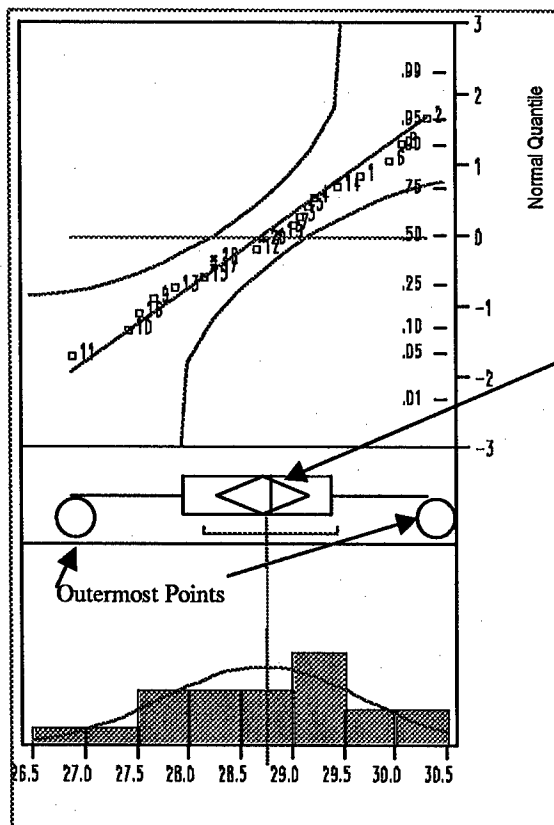
Moments

Mean	1.11715
Std Dev	0.03992
Std Error Mean	0.01152
Upper 95% Mean	1.14251
Lower 95% Mean	1.09178
N	12.00000
Sum Weights	12.00000

Test for Normality Shapiro-Wilk W Test

W	Prob<W
0.958403	0.7065

MOE #1: Days to Achieve Air Superiority – Resolution V



Quantiles

maximum	100.0%	30.320
	99.5%	30.320
quartile	97.5%	30.320
	90.0%	30.078
quartile	75.0%	29.403
	50.0%	28.815
quartile	25.0%	27.960
	10.0%	27.441
minimum	2.5%	26.900
	0.5%	26.900
minimum	0.0%	26.900

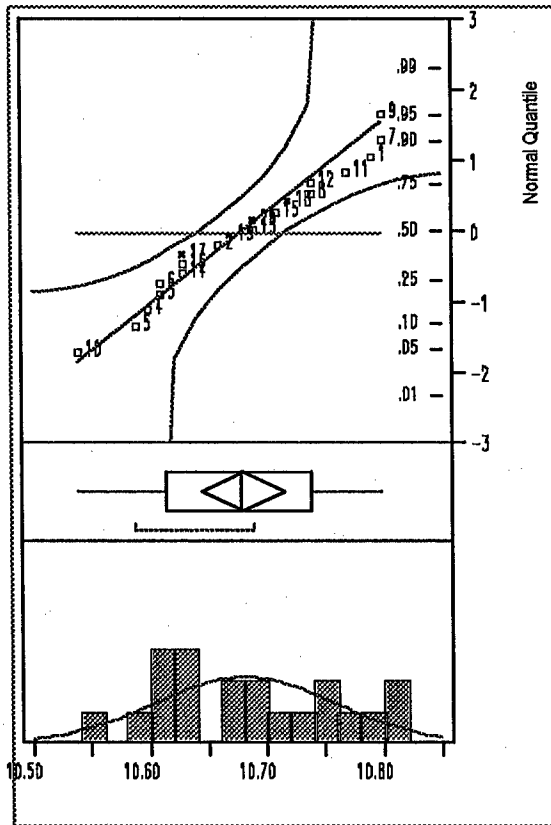
Moments

Mean	28.72550
Std Dev	0.94778
Std Error Mean	0.21193
Upper 95% Mean	29.16907
Lower 95% Mean	28.28193
N	20.00000
Sum Weights	20.00000

Test for Normality Shapiro-Wilk W Test

W	Prob<W
0.978728	0.9003

MOE #2: Days to Halt FLOT – Resolution V



Quantiles

maximum	100.0%	10.800
	99.5%	10.800
	97.5%	10.800
	90.0%	10.799
quartile	75.0%	10.740
median	50.0%	10.680
quartile	25.0%	10.615
	10.0%	10.591
	2.5%	10.540
	0.5%	10.540
minimum	0.0%	10.540

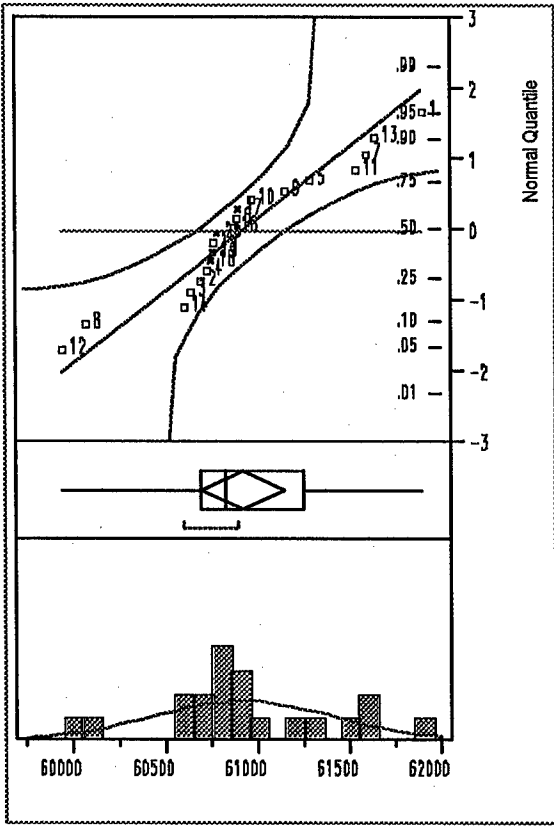
Moments

Mean	10.68100
Std Dev	0.07636
Std Error Mean	0.01707
Upper 95% Mean	10.71674
Lower 95% Mean	10.64526
N	20.00000
Sum Weights	20.00000

Shapiro-Wilk W Test Test for Normality

W	Prob<W
0.955880	0.4738

MOE #3: Red TAHAIT – Resolution V



Quantiles

maximum	100.0%	61901
	99.5%	61901
	97.5%	61901
	90.0%	61639
quartile	75.0%	61253
median	50.0%	60838
quartile	25.0%	60703
	10.0%	60130
	2.5%	59951
	0.5%	59951
minimum	0.0%	59951

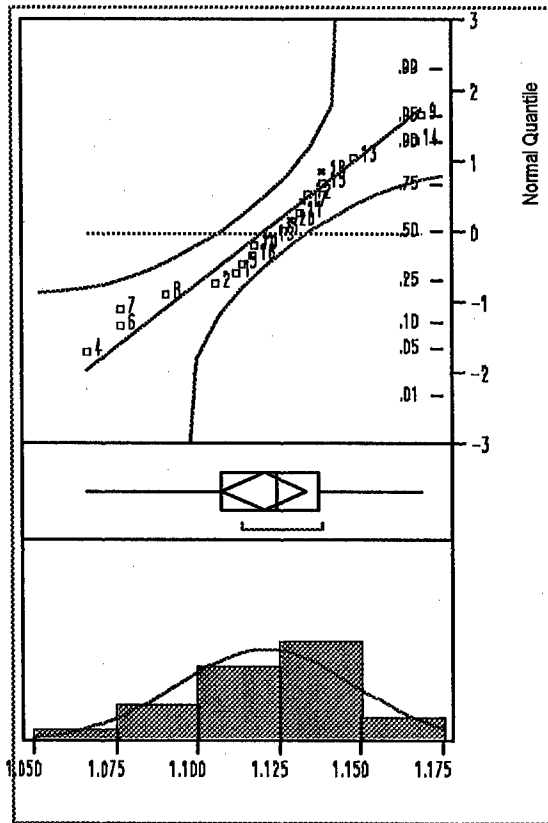
Moments

Mean	60926.92
Std Dev	487.98
Std Error Mean	109.12
Upper 95% Mean	61155.30
Lower 95% Mean	60698.54
N	20.00
Sum Weights	20.00

Test for Normality Shapiro-Wilk W Test

W	Prob<W
0.938341	0.2343

MOE #4: Exchange Ratio – Resolution V



Quantiles

maximum	100.0%	1.1690
	99.5%	1.1690
	97.5%	1.1690
	90.0%	1.1651
quartile	75.0%	1.1378
median	50.0%	1.1250
quartile	25.0%	1.1075
	10.0%	1.0770
	2.5%	1.0670
	0.5%	1.0670
minimum	0.0%	1.0670

Moments

Mean	1.12095
Std Dev	0.02753
Std Error Mean	0.00616
Upper 95% Mean	1.13384
Lower 95% Mean	1.10806
N	20.00000
Sum Weights	20.00000

Test for Normality Shapiro-Wilk W Test

W	Prob<W
---	--------

0.955993 0.4758

Appendix F. Regression Results

MOE #1: Days Needed to Achieve Air Superiority (AIRSUP)

Stepwise Linear Regression

Stepwise Regression Control

Prob to Enter 0.150

Prob to Leave 0.100

Current Estimates

SSE	DFE	MSE	RSquare	RSquare Adj	Cp	AIC
0.72502	11	0.06591	0.9575	0.9266	3.846994	-48.3458

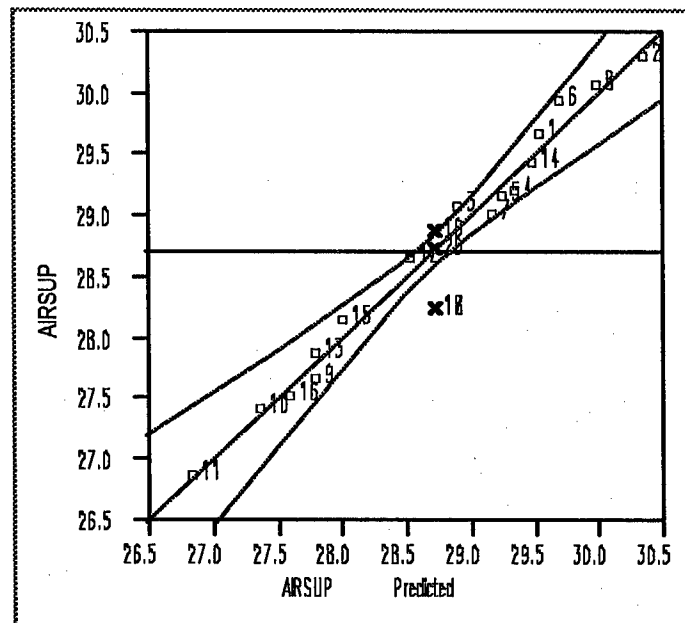
Lock	Entered	Parameter	Estimate	nDF	SS	"F Ratio"	"Prob>F"
X	X	Intercept	28.7255	1	0	0.000	1.0000
-	X	ACS	-0.8025	3	11.42702	57.790	0.0000
-	X	BLUEMSG	0.14375	2	0.69665	5.285	0.0246
-	X	ACS*BLUEMSG	0.15125	1	0.366025	5.553	0.0380
-	X	JSTAR	-0.18125	1	0.525625	7.975	0.0165
-	-	ACS*JSTAR	?	1	0.018225	0.258	0.6226
-	-	BLUEMSG*JSTAR	?	1	0.0121	0.170	0.6890
-	X	PRCPT	0.3175	2	2.4965	18.938	0.0003
-	-	ACS*PRCPT	?	1	0.0016	0.022	0.8847
-	-	BLUEMSG*PRCPT	?	1	0.015625	0.220	0.6489
-	-	JSTAR*PRCPT	?	1	0.009025	0.126	0.7299
-	X	SREC	-0.3125	3	3.203	16.199	0.0002
-	X	ACS*SREC	-0.2175	1	0.7569	11.484	0.0060
-	-	BLUEMSG*SREC	?	1	0.000225	0.003	0.9567
-	-	JSTAR*SREC	?	1	0.172225	3.116	0.1080
-	X	PRCPT*SREC	-0.235	1	0.8836	13.406	0.0037

Step History

Step	Parameter	Action	"Sig Prob"	Seq SS	RSquare	Cp	p
1	ACS	Entered	0.0001	10.3041	0.6037	38.542	2
2	PRCPT*SREC	Entered	0.0027	4.059	0.8416	11.808	5
3	ACS*SREC	Entered	0.0351	0.7569	0.8859	7.7042	6
4	JSTAR	Entered	0.0472	0.525625	0.9167	5.4652	7
5	ACS*BLUEMSG	Entered	0.0246	0.69665	0.9575	3.847	9
6	JSTAR*SREC	Entered	0.1080	0.172225	0.9676	4.4581	10
7	JSTAR*SREC	Removed	0.1080	0.172225	0.9575	3.847	9

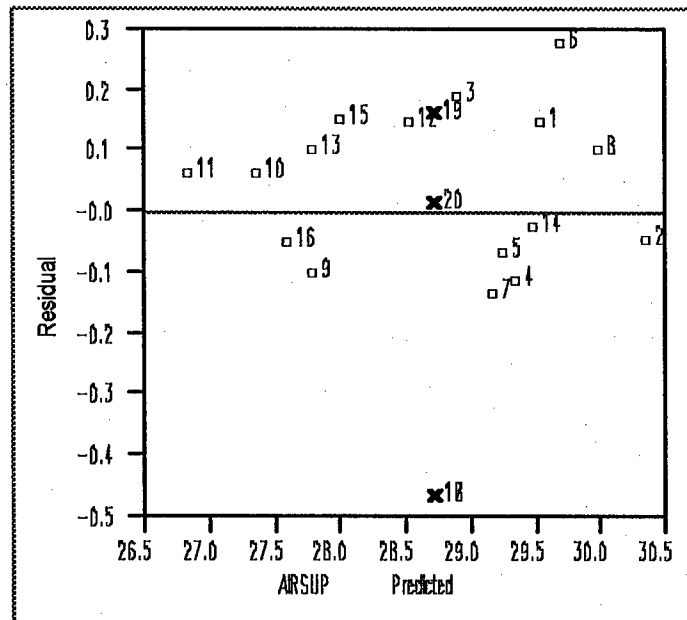
Standard Least Squares Regression

Whole-Model Test



Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	8	16.342275	2.04278	30.9931
Error	11	0.725020	0.06591	Prob>F
C Total	19	17.067295		<.0001



MOE #2: Days Needed to Halt FLOT (FLOT)

Stepwise Linear Regression

Stepwise Regression Control

Prob to Enter 0.250
Prob to Leave 0.100

Current Estimates

		SSE	DFE	MSE	RSquare	RSquare Adj	Cp	AIC
		0.0049988	6	0.000833	0.9549	0.8571	12.61113	-137.886

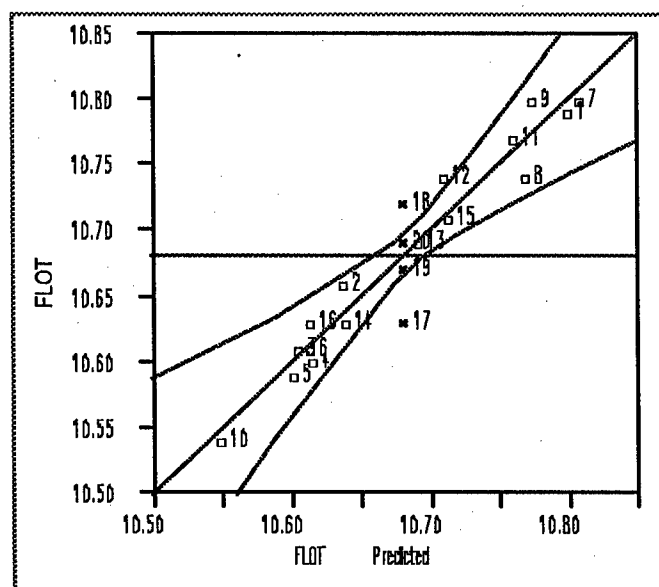
Lock	Entered	Parameter	Estimate	nDF	SS	"F Ratio"	"Prob>F"
X	X	Intercept	10.681	1	0	0.000	1.0000
-	X	ACS	0.006875	4	0.021875	6.564	0.0222
-	X	BLUEMSG	-0.006875	5	0.022581	5.421	0.0314
-	X	ACS*BLUEMSG	-0.016875	1	0.004556	5.469	0.0580
-	X	JSTAR	0.018125	3	0.020719	8.290	0.0148
-	-	ACS*JSTAR	?	1	0.000506	0.563	0.4867
-	X	BLUEMSG*JSTAR	0.026875	1	0.011556	13.871	0.0098
-	X	PRCPT	-0.038125	5	0.068281	16.392	0.0019
-	X	ACS*PRCPT	-0.015625	1	0.003906	4.689	0.0735
-	X	BLUEMSG*PRCPT	0.015625	1	0.003906	4.689	0.0735
-	X	JSTAR*PRCPT	0.015625	1	0.003906	4.689	0.0735
-	X	SREC	-0.003125	4	0.047925	14.381	0.0031
-	X	ACS*SREC	-0.028125	1	0.012656	15.191	0.0080
-	X	BLUEMSG*SREC	0.010625	1	0.001806	2.168	0.1913
-	-	JSTAR*SREC	?	1	0.000156	0.161	0.7045
-	X	PRCPT*SREC	-0.045625	1	0.033306	39.977	0.0007

Step History

Step	Parameter	Action	"Sig Prob"	Seq SS	RSquare	Cp	p
1	PRCPT*SREC	Entered	0.0081	0.056719	0.5120	37.869	4
2	ACS*SREC	Entered	0.1359	0.013413	0.6331	29.497	6
3	BLUEMSG*JSTAR	Entered	0.0902	0.017569	0.7917	19.29	9
4	ACS*BLUEMSG	Entered	0.1479	0.004556	0.8328	17.087	10
5	JSTAR*PRCPT	Entered	0.1554	0.003906	0.8680	15.484	11
6	ACS*PRCPT	Entered	0.1260	0.003906	0.9033	13.881	12
7	BLUEMSG*PRCPT	Entered	0.0850	0.003906	0.9386	12.277	13
8	BLUEMSG*SREC	Entered	0.1913	0.001806	0.9549	12.611	14

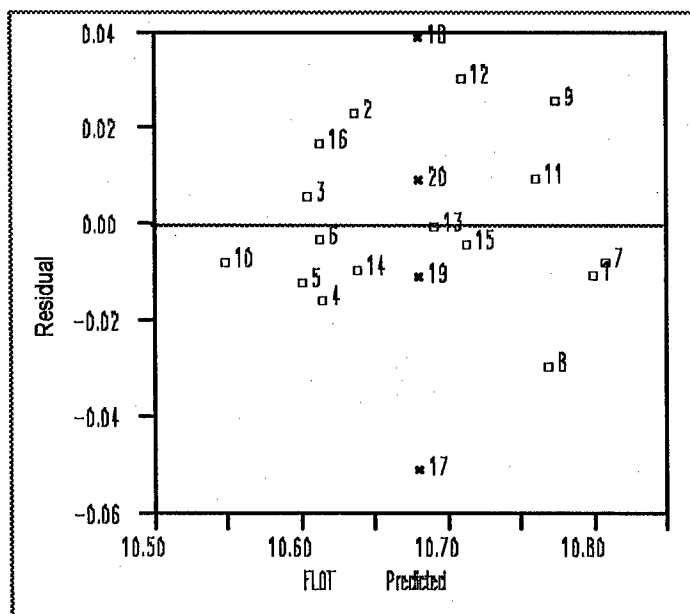
Standard Least Squares Regression

Whole-Model Test



Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	9	0.10230625	0.011367	13.4148
Error	10	0.00847375	0.000847	Prob>F
C Total	19	0.11078000		0.0002



MOE #3: Red TAHAIT Destroyed

Stepwise Linear Regression

Current Estimates

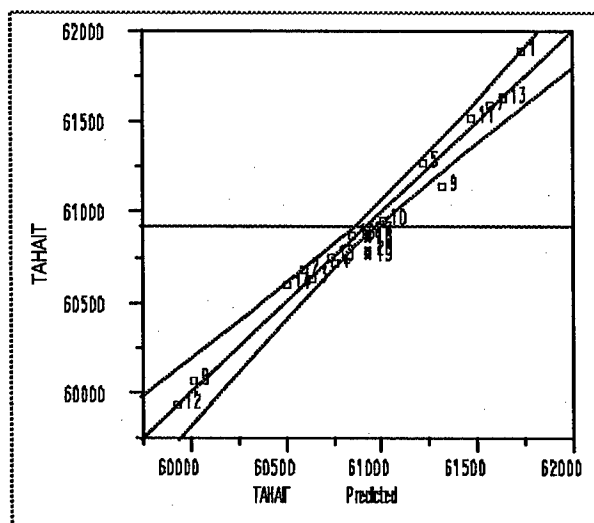
SSE		DFE	MSE	RSquare	RSquare Adj	Cp	AIC
141462.72		12	11788.56	0.9687	0.9505	2.220371	193.2812
Lock	Entered	Parameter	Estimate	nDF	SS	"F Ratio"	"Prob>F"
X	X	Intercept	60926.9205	1	0	0.000	1.0000
-	-	ACS	?	1	5716.494	0.463	0.5102
-	X	BLUEMSG	11.091875	2	36987.05	1.569	0.2482
-	-	ACS*BLUEMSG	?	2	15761.04	0.627	0.5540
-	X	JSTAR	-185.87938	3	759956	21.489	0.0000
-	-	ACS*JSTAR	?	2	20342.37	0.840	0.4601
-	X	BLUEMSG*JSTAR	46.783125	1	35018.57	2.971	0.1104
-	X	PRCPT	-357.55562	2	2076005	88.052	0.0000
-	-	ACS*PRCPT	?	2	12994.72	0.506	0.6177
-	-	BLUEMSG*PRCPT	?	1	210.1775	0.016	0.9005
-	-	JSTAR*PRCPT	?	1	423.0221	0.033	0.8592
-	X	SREC	310.746875	3	1747606	49.415	0.0000
-	-	ACS*SREC	?	2	6653.619	0.247	0.7859
-	-	BLUEMSG*SREC	?	1	11259.86	0.951	0.3504
-	X	JSTAR*SREC	103.718125	1	172119.2	14.601	0.0024
-	X	PRCPT*SREC	-43.638125	1	30468.58	2.585	0.1339

Step History

Step	Parameter	Action	"Sig Prob"	Seq SS	RSquare	Cp	p
1	PRCPT*SREC	Entered	0.0000	3621023	0.8003	27.724	4
2	JSTAR*SREC	Entered	0.0000	724937.5	0.9606	-0.153	6
3	BLUEMSG*JSTAR	Entered	0.2482	36987.05	0.9687	2.22048	

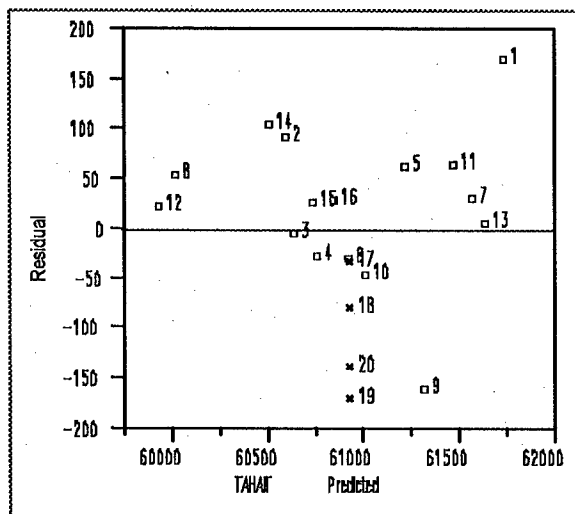
Standard Least Squares Regression

Whole-Model Test



Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	5	4350510.4	870102	81.8690
Error	14	148791.8	10628	Prob>F
C Total	19	4499302.1		<.0001



MOE #4: Exchange Ratio

Stepwise Linear Regression

Current Estimates

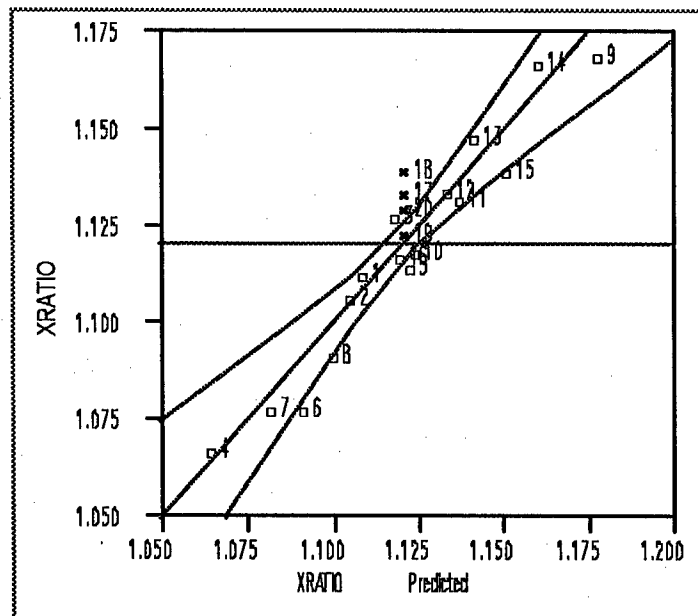
		SSE	DFE	MSE	RSquare	RSquare Adj	Cp	AIC
		0.0010025	12	0.000084	0.9304	0.8898	2.255806	-182.02
Lock	Entered	Parameter	Estimate	nDF	SS	"F Ratio"	"Prob>F"	
X	X	Intercept	1.12095	1		0 0.000	1.0000	
—	X	ACS	0.0220625	2	0.008103	48.497	0.0000	
—	X	BLUEMSG	-0.0021875	3	0.000887	3.538	0.0483	
—	X	ACS*BLUEMSG	0.0044375	1	0.000315	3.771	0.0760	
—	X	JSTAR	-0.0079375	1	0.001008	12.066	0.0046	
—	—	ACS*JSTAR	?	1	0.000068	0.801	0.3899	
—	—	BLUEMSG*JSTAR	?	1	0.000086	1.026	0.3328	
—	X	PRCPT	-0.0088125	2	0.001738	10.400	0.0024	
—	—	ACS*PRCPT	?	1	0.000086	1.026	0.3328	
—	X	BLUEMSG*PRCPT	0.0055625	1	0.000495	5.926	0.0315	
—	—	JSTAR*PRCPT	?	1	0.000005	0.056	0.8176	
—	X	SREC	-0.0124375	1	0.002475	29.626	0.0001	
—	—	ACS*SREC	?	1	0.000008	0.084	0.7778	
—	—	BLUEMSG*SREC	?	1	0.000014	0.156	0.7000	
—	—	JSTAR*SREC	?	1	5.625e-7	0.006	0.9388	
—	—	PRCPT*SREC	?	1	0.000095	1.152	0.3060	

Step History

Step	Parameter	Action	"Sig Prob"	Seq SS	RSquare	Cp	p
1	ACS	Entered	0.0002	0.007788	0.5407	25.278	2
2	SREC	Entered	0.0054	0.002475	0.7126	11.833	3
3	PRCPT	Entered	0.0186	0.001243	0.7988	6.0793	4
4	JSTAR	Entered	0.0127	0.001008	0.8688	1.7888	5
5	BLUEMSG*PRCPT	Entered	0.0961	0.000572	0.9085	2.2218	7
6	ACS*BLUEMSG	Entered	0.0760	0.000315	0.9304	2.2558	8

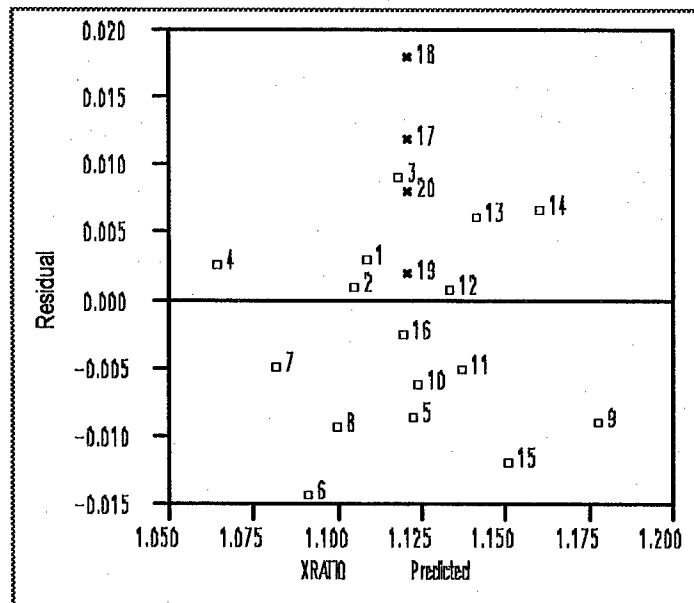
Standard Least Squares Regression

Whole-Model Test



Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	5	0.01300881	0.002602	26.1270
Error	14	0.00139414	0.000100	Prob>F
C Total	19	0.01440295		<.0001



Bibliography

1. Air and Space Power Validation Group (ASPVG). ASPVG Checklist of Objectives for Model Evaluation: Major Regional Conflicts. HQ USAF/XOM, Pentagon ADM VA, 20330. March 22, 1995.
2. Air Force Studies and Analysis Agency, THUNDER Analyst's Manual Version 6.4. Arlington VA: CACI Products Company. 1995.
3. Bailey, W. A. Jr., and T. D. Clark, Jr. Taxi Management and Route Control: A Systems Study and Simulation Experiment. In *Proceedings of the 1992 Winter Simulation Conference*. Edited by. J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson. New York: IEEE Press, 1992.
4. Bailey, Glen R. Class Notes, OPER 683, Response Surface Methodology. School of Engineering, Air Force Institute of Technology, Wright Patterson AFB, OH, July 1997.
5. Banks, Jerry, John S. Carson II, and Barry L. Nelson. Discrete-Event System Simulation. UpperSaddle River: Prentice Hall, 1996.
6. Box, George E. P. and Draper, Norman R. Empirical Model-Building and Response Surfaces. New York: John Wiley & Sons, 1987.
7. Department of Defense. Conduct of the Persian War – Final Report to Congress. PB 92-163674. Washington: GPO, April 1992.
8. Donohue, Joan M. Experimental Designs for Simulations. In *Proceedings of the 1994 Winter Simulation Conference*. Edited by. J.D. Tews, S. Manivannan, D.A. Sadowski, and A.F. Seila. New York: IEEE Press, 1994.
9. Donohue, Joan M. The Use of Variance Reduction Techniques in the Estimation of Simulation Metamodels. In *Proceedings of the 1995 Winter Simulation Conference*. Edited by C. Alexopoulos et al. New York: IEEE Press 1995.
10. Dorman, Clark, Sheldon, Bob and Warne, Dave. Using Genetic Algorithms (GAs) and Neural Networks (NNs) to Advance Campaign-Level Model Fidelity, Final Report for Phase I SBIR, Contract Number C F19628-96-C-0079, 31 Jan 97. S3I Report S3I-97-BS-01.

11. Farmer, Michael Ryan. The Effects of Changing Force Structure on Thunder Output MS Thesis. Air Force Institute of Technology, Wright Patterson Air Force Base, March 1996 (AFIT/GOA/ENS/96M-01).
12. Forsythe, Steven L. Optimization of the Air Apportionment in a TAC THUNDER Scenario Using Response Surface Methodology MS Thesis. Air Force Institute of Technology, Wright-Patterson Air Force Base, March 1994 (AFIT/GST/ENS/94M-02).
13. Gilly, Daniel. UNIX in a Nutshell System V Edition. Cambridge: O'Reilly & Associates, Inc., 1992.
14. Grier, James B. Linking Procurement Dollars to an Alternative Force Structure's Combat Capability Using Response Surface Methodology MS Thesis. Air Force Institute of Technology, Wright Patterson Air Force Base, March 1997 (AFIT/GOR/ENS/94M-18).
15. Gordon, S. C., J. J. Ausink, and R. J. Berdine. Using Experimental Design Techniques for Spacecraft Control Simulation. Simulation 62:303-309 1994.
16. Hallion, Richard P. Storm Over Iraq: Air Power and the Gulf War. Washington: Smithsonian Institution Press, 1992.
17. Hillestad, Richard J, and Louis Moore. The Theater-Level Campaign Model A Research Prototype for a New Generation of Combat Analysis Model RAND. Santa Monica CA, 1996.
18. Hobbs, Walt. TRCON, UNIX, Computer Software, RAND Corporation, Santa Monica, Ca, 1995
19. Hutcherson, Norman B., Command and Control Warfare Putting Another Tool in the War-Fighter's Data Base , Research Report NO. AU-ARI-941, Air University Press, Maxwell AFB, Alabama 36112-6610
20. JMP® Statistics and Graphics Guide, Version 3.1, Cary, NC. SAS Institute Inc., 1995
21. Jonini Baron Antoine Henri De. The Art of War, new edition of 1862 edition. London: Greenhill Books, 1992.
22. Kent, Glenn A. A Framework for Defense Planning. RAND Corporation, Santa Monica CA, 1996.
23. Khuri, A. I. And Cornell, J. A. Response Surfaces: Designs and Analysis. Marcel Dekker Inc., ASQC Quality Press, 1987.

24. Law, Averill M. and Kelton, David W. Simulation Modeling and Analysis (Second Edition). New York: McGraw-Hill, 1991.
25. Myers, Raymond H. and Montgomery Douglas C. Response Surface Methodology: Process and Product Optimization Using Designed Experiments. New York:, John Wiley & Sons, 1995.
26. Myers, Raymond H. , Khuri, Andre I. And Vining, Geoffrey. Response Surface Alternatives to the Taguchi Robust Parameter Design Approach. The American Statistician, May 1992, Vol. 46, No. 2.
27. Neter, John, Michael H. Kunter, Christopher J. Nachtsheim, and William Wasserman. Applied Linear Statistical Models (Fourth Edition). Chicago: Invin, 1996
28. Office of Aerospace Studies. Analysis of Alternatives (AOA) Guidance and Procedures. Draft OASP 97-1. Kirtland AFB: HQ AFMC/OAS, 1 July 1997.
29. Project Wedgetail. "Examples of Existing and Planned AEW&C Systems From Around the World." Excerpt from the website, WWWeb, <http://www1.tpgi.com.au/users/RAAF/AEWCAIR.HTM>.
30. SAB-TR-96-02ES. Vision of Aerospace Command and Control for the 21st Century. Executive Summary. SAF/PAS 96-1117. 26 November 1996. WWWeb, <http://web.fie.com/htdoc/fed/afr/sab/any/text/any/sabvispd.htm> . 21 January 1998.
31. Siegner, J. J. Analysis of Alternatives: Multivariate Considerations. MS Thesis. Air Force Institute of Technology, Wright Patterson Air Force Base, March 1998 (AFIT/GOA/ENS/98M-18).
32. Skordos, Panayotis A. Parallel Simulation of Subsonic Fluid Dynamics on a Cluster of Workstations. Massachusetts Institute of Technology Artificial Intelligence Laboratory, A.I. Memo No. 1485, Nov 1994.
33. Universal Joint Task List CJCSM 3500.04A Dynamics Research Corporation, Boston, MA. 13 Sep 96 http://www.drc.com/www/bus_area/systems/programs/tasks.htmlx. 16 December 1997.
34. UNIX Manual Pages, cpp (1), Sun Release 5.5, California, February 1995.
35. USAF Fact Sheet 96-13, E-3 Sentry (AWACS), Air Combat Command Public Affairs Office, 115 Thompson St, Ste 211 Langley AFB, VA 23665-1987. WWWeb, http://www.af.mil/news/factsheets/E_3_Sentry_AWACS_.html. 21 December 1997.

36. USAF Fact Sheet 91-03 Special Edition, Airpower in Operation Desert Storm, Air Combat Command Public Affairs Office, 115 Thompson St, Ste 211 Langley AFB, VA 23665-1987. WWWeb, <http://www.laafb.af.mil/SMC/CZ/homepag2/news/afnews11.htm>. 21 December 1997.
37. USAF Fact Sheet, E-8C Joint Stars, Air Combat Command Public Affairs Office, 115 Thompson St, Ste 211 Langley AFB, VA 23665-1987. WWWeb, http://www.af.mil/news/factsheets/E_8C_Joint_Stars.html. 21 December 1997.
38. Wackerly, Dennis D., Mendenhall, William, Scheaffer, Richard, L. Mathematical Statistics with Applications(Fifth Edition). Wadsworth Publishing Company, 1996
39. Webb, Timothy S. Analysis of THUNDER Combat Simulation Model MS Thesis. Air Force Institute of Technology, Wright-Patterson Air Force Base, March 1994 (AFIT/GOR/ENS/94M-18).

Vita

Major David A. Davies was born on 13 July 1962 in McGregor, Iowa. He graduated from Canoga Park High School in 1980 and entered undergraduate studies at the United States Air Force Academy (USAFA), Colorado. In May 1984, he received his Bachelor of Science Degree in Astronautical Engineering from USAFA. His first assignment was to Undergraduate Pilot Training (UPT) at Vance AFB, Oklahoma. Upon graduation from UPT he served a tour as a T-37 Instructor Pilot. Following this assignment he flew RC-135s at Offutt AFB, Nebraska serving as both an instructor and evaluator pilot. Following this he was assigned flying duties as the Chief of Flight Safety for Offutt AFB's 55th Wing. While at Offutt AFB, he earned a Master of Science degree in Aeronautical Science from Embry-Riddle University.

Major Davies was selected to attend the AFIT Graduate Program in Operations Analysis in 1996. Upon graduation from AFIT in March 1998, he was assigned to Air Combat Command as an operations analyst in the Studies and Analysis Squadron.

Permanent Address: 22222 Schoolcraft St
Canoga Park, CA 91303

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 1998		3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Sensitivity Analysis of the THUNDER Combat Simulation Model to Command and Control Inputs Accomplished in a Parallel Environment				5. FUNDING NUMBERS
6. AUTHOR(S) Major David A. Davies				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFIT/ENS 2950 P Street Wright - Patterson AFB, OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GOA/ENS/98M-02
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) ASC/SM, Bldg 28 2145 Monahan Way WPAFB, OH 45433-7017				10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFSAA/SAAB 1570 Air Force Pentagon Washington DC 20330-1570
11. SUPPLEMENTARY NOTES Advisor: Lt Col J.O. Miller, (937) 255-6565 ext. 4333, jmiller @afit.af.mil				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 words) This research had two objectives. The first was to develop a methodology to demonstrate the parallel processing capability provided by Air Force's Aeronautical System's Command (ASC) Major Shared Resource Center (MSRC) and apply that methodology to the SIMAF Proof of Concept project. Secondly, AFSAA/SAAB requested a sensitivity analysis of THUNDER to the modeled command and control (C2) inputs. The power of parallelization can not be overemphasized. The data collection phase of this thesis was accomplished at the MSRC using a script developed to automate the processing of an experimental design, providing the analyst with a launch and leave capability. On average it took 45 minutes to process a single replication of THUNDER. For this thesis we made 1,560 runs in slightly less than 3 days. To accomplish the same number of runs on a single CPU machine would have taken slightly more than 3 months. For our sensitivity analysis we used a Plackett-Burman Resolution III screening design to identify which of 11 input variables had a statistical impact upon THUNDER. The decision to investigate only the significant variables reduced the number of input variables from 11 to 5. This reduced the number of design points necessary to obtain the same Resolution V information from 128 to 16 and eliminated the need for 3,360 THUNDER runs. A significant savings! Using response surface methodology (RSM) techniques, we were then able to generate a response surface depicting the relationships between the input parameters and the output measures.				
14. SUBJECT TERMS Sensitivity, Command and Control Systems, Statistical Analysis, Theater Level Operations, Computerized Simulation, Computers, Input, Output,				15. NUMBER OF PAGES 220
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	