

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

**VALIDATION OF THE
TIME STRIKE OPTIMIZATION MODEL
THROUGH SIMULATION**

DTIC QUALITY INSPECTED 3

by

John J. Kosina

September, 1997

Thesis Advisor:
Second Reader:

Alan R. Washburn
Arnold H. Buss

Approved for public release; distribution is unlimited.

19980206 032

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 instruction, 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
September 1997

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
VALIDATION OF THE TIME STRIKE OPTIMIZATION MODEL THROUGH SIMULATION

5. FUNDING NUMBERS

6. AUTHOR(S)
Kosina, John J.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

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.

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

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

The TIME STRIKE optimization model was developed in 1995 for use by the cognizant US Air Force agencies to investigate requirements for conventional munitions and the feasibility of operational plans based on their availability and current budgets. The problem addressed here is: Is the output of TIME STRIKE accurate when compared to a simulation? This thesis develops a computer simulation, called SimStrike, which models all the same things TIME STRIKE does, using the same data, however with randomness used where TIME STRIKE uses expectations. It was found that TIME STRIKE and SimStrike produce similar results.

14. SUBJECT TERMS
Time Strike, Munitions, Optimization, Simulation, Linear Programming, USAF

15. NUMBER OF PAGES
84

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

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

Approved for public release; distribution is unlimited

**VALIDATION OF THE TIME STRIKE OPTIMIZATION MODEL
THROUGH SIMULATION**

John J. Kosina
Lieutenant, United States Navy
B.S.M.E., Wright State University, 1988

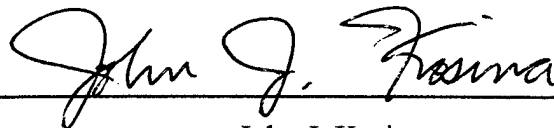
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

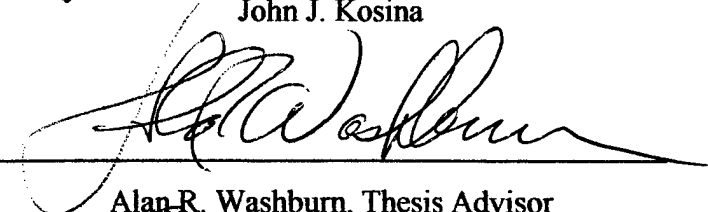
**NAVAL POSTGRADUATE SCHOOL
September 1997**

Author:



John J. Kosina

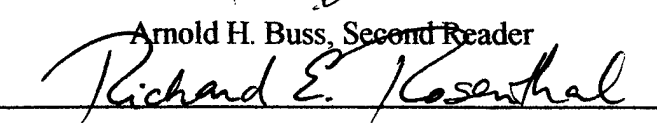
Approved by:



Alan R. Washburn, Thesis Advisor



Arnold H. Buss, Second Reader



Richard E. Rosenthal, Chairman
Department of Operations Research

ABSTRACT

The TIME STRIKE optimization model was developed in 1995 for use by the cognizant US Air Force agencies to investigate requirements for conventional munitions and the feasibility of operational plans based on their availability and current budgets. The problem addressed here is: Is the output of TIME STRIKE accurate when compared to a simulation? This thesis develops a computer simulation, called SimStrike, which models all the same things TIME STRIKE does, using the same data, however with randomness used where TIME STRIKE uses expectations. It was found that TIME STRIKE and SimStrike produce similar results.

TABLE OF CONTENTS

I. INTRODUCTION	1
A. OPTIMIZING VS. EVALUATIVE MODELS	1
B. THE TIME STRIKE OPTIMIZATION MODEL	1
C. AIM OF THIS THESIS	2
II. SIMSTRIKE DESCRIPTION	5
A. COMPARISON OF TIME STRIKE AND SIMSTRIKE	5
1. Constraints on Sorties	6
2. Distribution of Sorties	7
3. Expected Kills Per Sortie (EKS)	8
4. Attrition Per Sortie	8
5. Kill Goals	8
6. Time Periods and Planning Cycles	9
7. Battle-Damage Assessment and Target Regeneration	10
8. Calculation of the Objective Function Value	19
B. SUMMARY OF COMPARISONS OF TIME STRIKE AND SIMSTRIKE	19
III. RESULTS	21
A. RESULTS	21
B. SOURCES OF DIFFERENCE	23
1. Period Transitions and Restrikes	23
2. Target Rich Environment	24
3. Proportioning of Sorties Over Time and Restrikes	26
C. CONCLUSIONS	28
IV. FUTURE WORK	29
A. WEATHER	29
B. PROBABILITY DISTRIBUTION FOR RANDOM ROUNDING	29
APPENDIX A. PROGRAM LISTING AND DIRECTIONS FOR USE	31
APPENDIX B. SAMPLE OUTPUT	57
LIST OF REFERENCES	69
INITIAL DISTRIBUTION LIST	71

LIST OF FIGURES AND TABLES

Figure 1. The four stages associated with sorties in each day of a SimStrike period	13
Figure 2. SimStrike Objective Function Value Histogram for 1000 Replications	22
Figure 3. Comparison of TIME STRIKE and SimStrike for a Target Rich Environment	25
Table 1. Comparison Summary of TIME STRIKE and SimStrike	20
Table 2. Sensitivity Analysis	24
Table 3. Target Rich Environment Data for Figure 3	26
Table 4. Example of Carry-Over Restrikes	27

ACKNOWLEDGEMENTS

I wish to acknowledge the help provided by the following people, without whose assistance this thesis would not have been possible. The first is my Thesis Advisor, Professor Alan Washburn, who provided the guiding hand needed to get me back on track when it was apparent sometimes that I was heading down the wrong avenues in the development of the simulation. The second is Maj. Kirk Yost, USAF, the author of TIME STRIKE, who always showed patience in answering the numerous questions I seemed to have about his model.

I. INTRODUCTION

A. OPTIMIZING VS. EVALUATIVE MODELS

Models used in the field of Operations Research can be roughly divided into those that find an optimal decision for some problem (optimizing models), and those that merely evaluate a given decision (evaluative models). Since their ambitions are higher, optimization models generally are forced to make more abstractions and approximations in the data they use to achieve their results. These approximations may leave out much of the detail otherwise involved in the decision process being modeled, so one is led to ask the question: Are the results produced by a certain optimization model being studied accurate enough for the approximations used?

This question leads to the topic of verification of the optimization model. Verification can be accomplished through construction of an evaluative model that includes many more of the details left out of the optimization model. These details consist of refinement of the approximations used in the optimization model to more closely model the situation originally desired in the optimization model. The evaluative model can be used to test the optimal solution, results obtained and compared, and conclusions drawn.

This thesis attempts to verify one particular optimizing model, the TIME STRIKE munitions optimization model, by constructing a parallel evaluative model called SimStrike.

B. THE TIME STRIKE OPTIMIZATION MODEL

The linear program (LP) being evaluated by this thesis is the TIME STRIKE munitions optimization model (henceforth simply referred to as TIME STRIKE). TIME STRIKE was introduced in 1995 for use by various US Air Force agencies to develop requirements for conventional munitions, to refine operational plans based on the availability of different mixes of munitions, and to assess the effects of procuring different types and quantities of munitions [Ref. 1:p. i]. It creates sortie allocations across time for a given strike scenario against enemy targets based on things such as the type of aircraft, weapons types and loadouts, flight profiles possible, attrition and sortie rates, the length of time periods, target regeneration and battle damage assessment

(BDA), weather, budget, etc. Depending on the desires of the user, TIME STRIKE has available five objective functions [Ref. 1:p. 23], but the most useful of these, for the purposes of this thesis, is the one maximizing the weighted sum of target-value-destroyed (TVD) and time-scripted goals because it gives us a quantitative idea of the "reward" gained for the number of targets killed.

Since TIME STRIKE is a large LP, it is forced to make some approximations in order to permit it to stay linear. The major approximations made are permitting non-integer values for the number of sorties flown and expected kills per sortie (EKS), and using expected values in the place of random quantities. These approximations are necessary since TIME STRIKE must remain linear, but they carry with them the potential for error. Consider this simple example: Suppose there were 10 targets to be struck and there were 6 sorties available, each with an EKS of 1.8. TIME STRIKE would simply say you need $(10 / 1.8 =) 5.6$ sorties out of the 6 available to accomplish the mission, assigning the remaining 0.4 sorties to some other target. On the other hand, suppose instead that we force kills and sorties flown to be integer valued and random, as they are in real life. Using the notation "[no. kills by sortie 1, no. kills by sortie 2,...,no. kills by sortie 6]" to represent how many kills each sortie makes against the 10 targets, on any particular replication we may get the following results: [2, 2, 2, 2, 2, 0], which is a case where the first 5 sorties kill all 10 targets, exactly, so the 6th sortie need do nothing; [1, 2, 2, 2, 2, 2], which is a case where all 10 targets are killed with the number of kills achieved by the first 5 sorties plus 1 of the 2 kills achieved by the 6th sortie, but now there is 1 kill left over at the end, which could be applied to another target if there were more; [1, 2, 1, 2, 2, 2], which is another case in which all 10 targets are killed, exactly, but this time needing the total kills achieved over all 6 sorties in order to do it, instead of just the first 5 sorties as in the first case examined; or, if the sorties are particularly unlucky, we could get something like [1, 1, 1, 2, 1, 1], a case in which not all of the 10 targets could be killed using 6 sorties with the EKS given! But the interesting thing, and also the cause of concern, is that TIME STRIKE does not have to face any of these problems which may occur in real life.

C. AIM OF THIS THESIS

This thesis develops an independent Monte Carlo simulation, called SimStrike, to test the TIME STRIKE solution. SimStrike essentially re-flies the sorties output by

TIME STRIKE, but applying probabilities where TIME STRIKE uses expectations, and randomly rounding the non-integer numbers of sorties output by TIME STRIKE to make them integer. It accounts for all the same things TIME STRIKE does, i.e., weather, BDA, kill goals, etc., and generates the mean TVD for a large number of replications along with the standard deviation. The objective is to determine whether the output of SimStrike equals the output of TIME STRIKE, on the average. The actual SimStrike program listing and directions for use can be found in Appendix A, with a sample output provided in Appendix B.

The remainder of this thesis is organized as follows: Chapter II provides a point by point comparison of the major features modeled in TIME STRIKE and SimStrike, along with a concise summary of these comparisons tabulated at the end of the chapter. Chapter III discusses the results of the thesis obtained and conclusions drawn, and also discusses some sources of difference between TIME STRIKE and SimStrike. Chapter IV discusses two areas of future work which might be considered for possible future revisions to SimStrike.

II. SIMSTRIKE DESCRIPTION

A. COMPARISON OF TIME STRIKE AND SIMSTRIKE

SimStrike is a simulation model developed to evaluate the TIME STRIKE optimization model. SimStrike re-flies the sorties output by TIME STRIKE, and, using the same attrition and expected kills per sortie (EKS) values for each engagement in each period, applies probabilities rather than expectations to achieve its results. To achieve accurate results, all the same things are modeled which affect the life of a sortie as in TIME STRIKE.

Much of the complication in TIME STRIKE arises from the need to maintain the linearity of the optimization model. As a result, some of the quantities which represent physical entities, such as sorties, aircraft, and targets, are allowed to assume non-integer (also continuous) values, rather than being restricted to integer values, as they should be in real life, to represent unit quantities, e.g., 1 sortie, 1 aircraft, and 1 target vice 0.5 sorties, 0.6 aircraft, and 1.3 targets.

For the reader not familiar with TIME STRIKE, some explanations of terms used is in order first:

A *period* is a fundamental unit of time in TIME STRIKE. In TIME STRIKE, the entire time over which the model is run is divided into periods or user-selectable lengths, with each of these periods consisting of the same number of fixed-length *planning cycles* [Ref. 1:p. 8]. Planning cycles are more generally referred to as *days*, the two terms being interchangeable in meaning. Many variables and data items used in TIME STRIKE are subscripted by period, but not by day. In this thesis, the terms *campaign* or *timed strike* are also used to represent an entire run of the TIME STRIKE model over time.

Once TIME STRIKE sorts through the user-provided data, *sorties* are assigned by TIME STRIKE to fly *engagements* against *targets* [Ref. 1:p. 1], an engagement being the term applied to an encounter between a single sortie and its assigned target(s). A sortie is an aircraft taking off, flying strikes against one or more targets, making kills against them or perhaps being killed itself, then returning to land if not attrited. Targets are, to put it simply, the enemy. *Target types* refer to a group of targets which are all the same, i.e., all anti-aircraft guns, all tanks, all transport vehicles, etc. *Target classes* refer to a group of target types which all have the same *kill goals*, i.e., the same amount expected to be dead or in repair at a certain point in time during the campaign.

Battle-damage assessment (BDA) is a term used to represent what is thought to be the results of a previous day's strikes. If BDA is perfect, every target killed is known to be either *dead forever*, i.e., no chance of being repaired or coming back to life, or undergoing *regeneration*, i.e., being repaired. If BDA is less than perfect, in addition to knowing about dead targets as a result of being killed by sorties, one or more of these dead targets may be thought to be still alive, in which case a *restrike* sortie (henceforth simply referred to as a *restrike*) is assigned to be flown against it the next day. When a dead target is misclassified as still being alive, the term *mis-BDA* is used to apply to this situation. The possibility that live targets might be thought dead is modeled in neither TIME STRIKE or SimStrike.

1. Constraints on Sorties

Before TIME STRIKE can assign sorties to strikes against targets, it must sort through the user-provided data to find valid combinations of aircraft, weapons, weapons loadouts, delivery tactics (or profiles), time periods, weather states, and target types [Ref. 1:p. 1]. The number of *available* sorties is a function of the sortie rate, the attrition rate, and the length of the time period [Ref. 1: p. 54]. The sorties are then further constrained by the assigned missions which must be performed each period.

Because TIME STRIKE is an LP, it assumes perfect information with regards to *restrikes*, so it knows from the start exactly how many sorties it must place in each day of each period in order to carry out the *restrikes*. Sorties are proportioned over each day of each period to exactly accomplish the *restrike* missions with no excess left over. Herein lies a deviation from reality. In real life, it is not physically possible to assign 1.2 sorties, for example, to a *restrike* mission against some *mis-BDA*'d target. How does one come up with 0.2 sorties as a mission planner? TIME STRIKE allows this, however.

There is a similar problem for targets. Since target regeneration and kills are being optimized in TIME STRIKE to achieve the maximum possible TVD, targets may be proportioned as non-integer quantities with non-integer EKS to allow a non-integer number of sorties to kill every last one of the targets. The question which arises here is: How can TIME STRIKE's advice be taken? Or another way of stating the question: Can we be assured of the accuracy of its results given this obvious deviation from reality?

SimStrike handles this aspect a little differently than TIME STRIKE. The total number of sorties an aircraft type actually flies in SimStrike is either the total flown by

the same aircraft in TIME STRIKE, which is the maximum possible it could fly in the period, or some lesser value which could be as a result of losing the maximum number of aircraft allowed for the aircraft type in the period, or achieving the kill goal for the target type in the period, or simply killing the remainder of the targets. However, since SimStrike works exclusively with integers for physical entities, i.e., sorties, targets, and aircraft, and therefore also with EKS, target kills are tallied as integers, or whole objects of whatever type the targets are, which is the case in reality. So when an aircraft kills a target, it does not kill a portion of it, it kills the whole target. Likewise, when sorties are flown, they are flown as whole entities, not fractions of sorties.

The restrictions placed on the flying of sorties in SimStrike are all the same major restrictions placed on sorties by TIME STRIKE.

2. Distribution of Sorties

The number of sorties flown by any aircraft against any target by TIME STRIKE in any period is uniformly distributed over the period, proportioning a certain number for each day of the period to account for all the actions which must be performed, i.e., restrikes, strikes against live targets, target regeneration, aircraft attrition, and weather aborts. The smallest time unit of resolution for sorties is by period. Also, aircraft do not really exist in TIME STRIKE, but rather aircraft *sorties*. "Aircraft" is merely a term used to represent a category for which losses are counted while the LP optimizes under the constraint that there is a cap on how many aircraft of each type can be lost, which is cumulative over time.

SimStrike handles this differently and comes closer to reality on the issue of sorties and aircraft. SimStrike treats aircraft as actually existing and important entities, separate from the number of sorties they actually fly. Each aircraft is given individual attention as it flies its sorties. SimStrike still distributes sorties uniformly over each period as TIME STRIKE does, but the main difference is that once the number of sorties is distributed uniformly over the period in the simulation, this value is used to produce how many sorties per day an aircraft could possibly fly, i.e., the maximum available for the day for the aircraft type. But then SimStrike rounds this non-integer valued quantity randomly to an integer value, e.g., 2.3 sorties have a 70 percent chance of being 2 and a 30 percent chance of being 3 sorties. Over a large number of replications of SimStrike, we expect this random rounding to average to the number TIME STRIKE flies.

3. Expected Kills Per Sortie (EKS)

EKS is input data to TIME STRIKE, subscripted by aircraft type to which the sortie belongs, weapon, target, weapons loadout, weapons delivery profile, and time period. EKS can be greater than 1 for a sortie with multiple weapons, and need not be an integer. To describe the problem which exists with this in TIME STRIKE, we go back to the simple example mentioned in Chapter I. If EKS is 1.8, and there are 10 targets with 6 sorties available to fly strikes against them, TIME STRIKE will allow $(10 / 1.8 =) 5.6$ sorties to fly and exactly kill the 10 targets. In reality, it is not possible to proportion physical entities such as targets and sorties in this manner.

EKS is also input data to SimStrike, but SimStrike rounds EKS values randomly, as it does sorties, to make them integer. If an aircraft is not attrited and does not experience an in-flight weather abort, it is expected to make kills against any remaining live targets based on its EKS in SimStrike. Since sorties, targets, and target kills are integer valued, an integer number of kills is applied to an integer number of targets.

4. Attrition Per Sortie

TIME STRIKE treats attrition as a non-integer valued expectation of aircraft losses over time and so proportions a certain number of these losses over each period (see Ref. 1 for details). The problem here is that aircraft, like targets, are physical entities in real life. Either the whole aircraft must be destroyed, or none of it.

SimStrike maintains this integrality of attrition. Before an aircraft sortie has a chance to do anything else in SimStrike, whether it be restrikes or regular strikes against live targets, it has a chance to be attrited based on this sortie's attrition value. If attrited, it does not get a chance to kill any targets, i.e., no "kamikaze" capability, and loses the remainder of its sorties for the day. SimStrike then proceeds to the next aircraft in the loop, or the next day if this was the last aircraft to fly. Other aircraft must pick up the lost aircraft's mission requirements, as would be the case in reality. If an aircraft is attrited, the total number of aircraft of this particular aircraft type is decremented by 1.

5. Kill Goals

The chosen objective function is based on time-scripted kill goals for each of the target classes present. These kill goals in TIME STRIKE are cumulative over time so

that in at least the last time period of a campaign this value is 1.0. A kill goal of 1.0 translates into all the targets originally present in the beginning (start of period 1) of this particular target type are expected to be dead or in repair (a target must be struck and killed before it has a chance to be in repair) by the end of the period of the campaign in which the kill goal became 1.0. Aircraft in any period can kill up to the kill goal, but not over the goal, with anything killed which was less than the goal being charged as a penalty against any TVD reward gained so far.

This aspect of TIME STRIKE is modeled exactly the same in SimStrike, with the only difference being that the actual calculation in SimStrike of the proportion killed, found from the number of targets dead or in repair divided by the total number present at the very start, has a numerator and denominator which are both integer. This presents the possibility of some instances arising where SimStrike cannot exactly meet kill goals in some periods. For example, if the kill goal for one period is 0.5, and for the next period it is 0.75, and say there are 3 targets, the most kills SimStrike can hope to achieve in the first period is 1 out of the 3 targets, which is approximately 0.33 proportion killed and under the goal of 0.5. If it killed 2, this would create a proportion of approximately 0.67, which is over the goal and so not allowed. Therefore, to make the 2nd kill, SimStrike would force the sorties to wait until the next period when the kill goal is 0.75, and the 3rd kill would have to wait until a period in which the goal rises to 1.0.

6. Time Periods and Planning Cycles

The number of periods and days per period are input data to TIME STRIKE. TIME STRIKE places the required number of sorties into each of these days, or planning cycles, with the expectation of accomplishing everything planned for that day, i.e., restrikes and regular strikes against live targets.

The problem is that in reality, a sortie may not be able to accomplish the mission it was assigned to perform. Whether because of nature or pure bad luck, the sortie might experience an in-flight weather abort, not kill as many targets as thought, or perhaps more, or suffer attrition. The randomness of nature is not modeled in TIME STRIKE when it comes to sorties. Everything planned may not get accomplished in a day, with the result that tomorrow's sorties may have to pick up some of today's mission objectives.

The number of periods and days per period are also input to SimStrike. The file of solution variables generated by TIME STRIKE after each run of the model, from which SimStrike will directly read data for its corresponding run, generates a line of data for each engagement run by each aircraft type flying sorties against a particular target type. So, for example, if aircraft type 2 flies sorties against target type 5 once in each of five periods, there will be a total of five lines of data for this scenario, one for each engagement run in each period. The file is organized by period and then by aircraft type in each period. This file is in spreadsheet format and is called "TSVTST.CSV", and is placed in the TIME STRIKE "RESULTS" directory after each run of the LP.

Each line of data in this output file will contain the total number of sorties flown over the *whole period* for the engagement run. Once the number of sorties per day is determined by SimStrike, it then simply steps through the days in the period. In each day, each *surviving* aircraft gets to fly sorties at its sortie rate against the targets of the target type, first against restrikes, and then until the maximum possible number of sorties is flown, the maximum allowed amount of aircraft are lost, all the targets are dead, or the kill goal is reached. SimStrike essentially uses nested loops here to accomplish this with test conditions for premature exit. This way SimStrike can model reality as close as possible by exiting a loop before normal completion when necessary because the conditions for exit exist. There is also a varying condition for the start of the aircraft loop for each day, based on how many aircraft of the aircraft type have been lost, which is continuously updated.

The important aspect modeled by SimStrike here is that the randomness of nature does exist and must be accounted for. While SimStrike reflects TIME STRIKE's sorties, the result may not be the same. For example, the same number of sorties required to accomplish restrikes on one day, may not be the number SimStrike uses due to possibly having to account for mission objectives which were not accomplished by the sorties on a previous day, or fewer sorties may be needed for the restrikes simply because they were luckier and conducted more kills per sortie. In summary, the randomness of reality and changing mission requirements is modeled in SimStrike.

7. Battle-Damage Assessment and Target Regeneration

As part of the target data input to TIME STRIKE for each target type, there are probability expectations associated with correct BDA and repair proportions for killed

targets, the repair proportions having to do with target regeneration. These are dealt with as proportions by TIME STRIKE when determining how many targets will be restrikes and how many will be in repair in each day of each period, separate from the other category which is possible, namely targets which are killed and become dead forever. All targets are in one of these three states throughout a run of TIME STRIKE: a restrike (mis-BDA'd), in repair, or dead forever. For a more detailed explanation of the precise determining equations associated with placing the targets into these categories, the interested reader is referred to Appendix A of Ref. 1.

The problem with this classification method used by TIME STRIKE is that the same proportion of targets struck and killed on any previous day of a period will get into one of the three categories mentioned above on a current or future day *every time*. One therefore knows with certainty that for a certain number killed today, what the proportion of targets are that must be restruck tomorrow, or what future regeneration will be. Again, the randomness of reality is not present here in TIME STRIKE.

The same data for BDA and target repair/regeneration are used as probabilities within SimStrike and are applied across essentially four stages, or phases, which must be accomplished, or at least checked to see if the conditions are right for accomplishment, each day of the campaign. These four stages are: (1) Regeneration, executed once each day, which represents the probability of targets killed in previous days of regenerating on the current day; (2) Restrikes, executed once each day, but potentially by more than one aircraft and possibly many sorties, depending upon how many restrikes there are, until all the restrikes are complete or all the day's are sorties flown against them; (3) Regular strikes against actual live targets based upon each sorties' EKS value; and (4) Redistribution of kills made by the regular strike phase based on BDA. When actions against targets are kept track of in this manner, the same categories possible for targets in TIME STRIKE are also modeled in SimStrike. The major difference, however, is that the same proportion of killed targets will not be placed into the same category each time. Instead, the randomness of reality is accounted for in SimStrike, and it is only *possible*, based on probability, for a killed target to evolve in a certain way throughout any particular simulation run, which will not be the same way every time SimStrike is run.

For purposes of the discussion of these stages in the subsections which follow, we will refer often to Figure 1 below, which is an example of progressing through these stages with 40 targets of a particular target type, with the starting condition as shown for each of the categories, and the probabilities of following any path as indicated on the

arrows. The notation "Pc {2}" means, for example, that with probability Pc, every target in the category at the tail of the arrow will move independently to the category at the head of the arrow, and in this case 2 targets met the conditions for movement on the arrow; when just a number in brackets is present, with no associated probability listed next to it, e.g., "{3}", this means that all targets in the category at the tail move to the category at the head of the arrow.

a. Target Categories

The six target categories shown in Figure 1, which are also what the variable names are called in SimStrike, are defined below:

(1) **LiveN Targets.** LiveN targets are the number of live targets that are not restrikeable. These represent all the actual live targets, recognizable as alive by the aircraft flying sorties against them. At the start of a replication of a SimStrike run, this variable is initialized to the total number of targets of the target types present as read in from the data input file.

(2) **DeadN Targets.** DeadN targets are the number of dead targets that are not restrikeable. These are targets which are dead forever, i.e., will not regenerate or be subject to BDA. Once a target gets into this category, it stays there throughout the remainder of the SimStrike replication run.

(3) **RegenN Targets.** RegenN targets are the number of repairable targets that have previously been killed but placed into repair, and are not restrikeable.

... Start Day

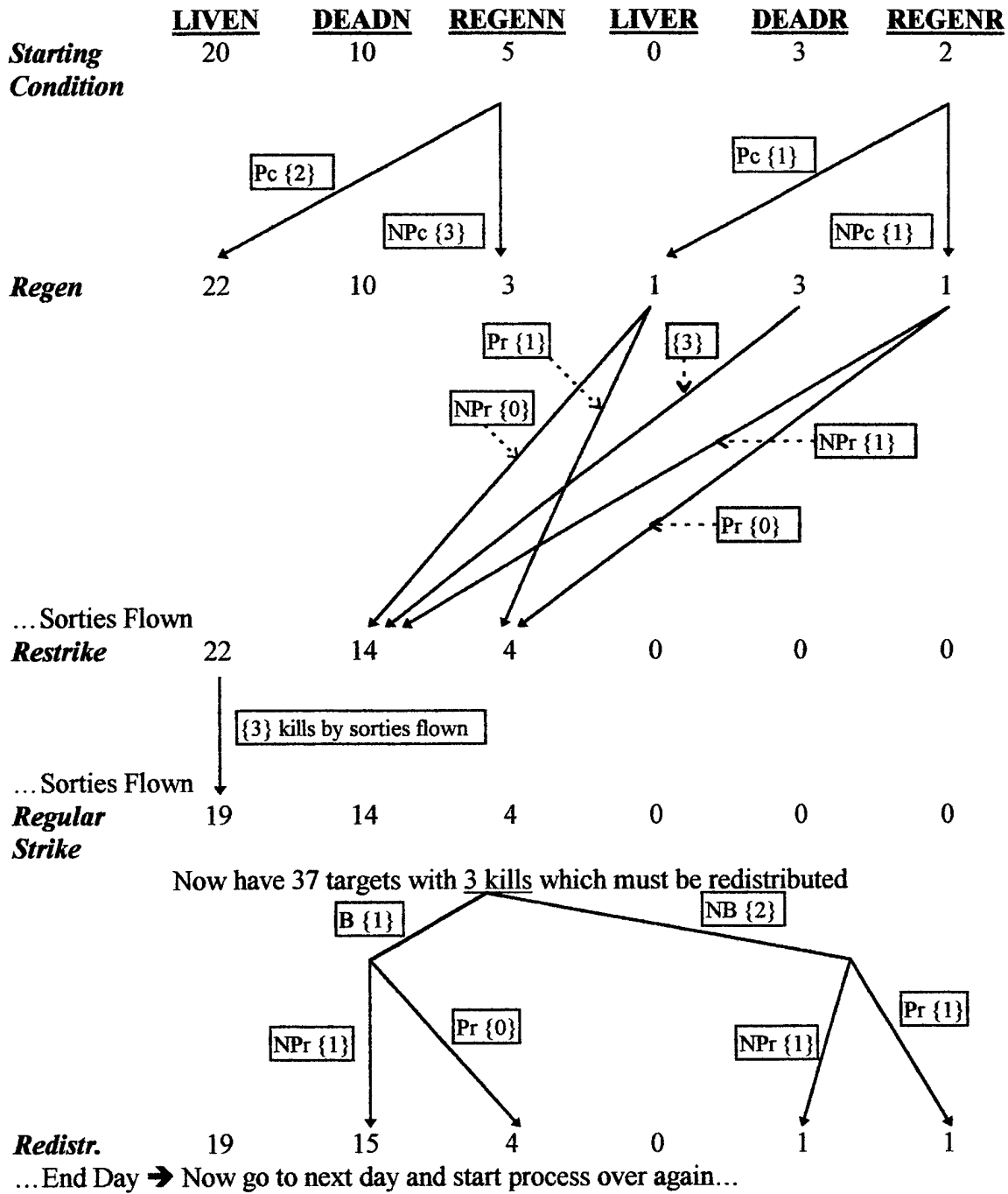


Figure 1 - The four stages associated with sorties in each day of a SimStrike period.

(4) **LiveR Targets.** LiveR targets are the number of live targets that are restrikeable. This category represents the number of targets which have just come out of repair in the planning cycle but may be restruck and returned to repair (become a RegenN target), or become dead forever (a DeadN target).

(5) **DeadR Targets.** DeadR targets are the number of dead targets that are restrikeable. These are the targets that were actually killed in a previous planning cycle but were not repairable, i.e., dead forever, but were mis-BDA'd and are believed to be still alive. The only possibility for these targets are to become dead forever (DeadN) when restruck.

(6) **RegenR Targets.** RegenR targets are the number of repairable targets that are restrikeable. These targets may regenerate and become a LiveR target, or remain as a RegenR target through the regeneration phase, and then any remaining RegenR targets may become RegenN or DeadN targets during the restrike phase.

b. Comparison of SimStrike Target Categories with TIME STRIKE

TIME STRIKE does not explicitly use the target categories as shown in Figure 1 and described above, but allows for each of the same six possibilities for the state of a target at any given time through the BDA equations it uses to calculate the proportion of targets existing in each of the states over time (see Appendix A of Ref. 1). These six categories in SimStrike, however, along with the associated probabilities shown on the arrows in Figure 1, and defined below, model the same aspects of the BDA equations used in TIME STRIKE, but probabilistically.

(1) **Probability a target regenerates in the next planning cycle, P_c .** This probability is associated only with the RegenN and RegenR categories of targets in SimStrike and is calculated from, $P_c = 1 - e^{-1 / \text{Repair Time}}$, where "Repair Time"

is input data for both TIME STRIKE and SimStrike representing the amount of time in days required for repair of a killed target of a particular target type. The probability NPC is merely the quantity $(1 - Pc)$.

During the regeneration phase at the start of each day, this is the probability used to determine how targets in the RegenN and RegenR categories will be distributed as shown by the arrows in Figure 1.

(2) **Probability a target is repairable after a strike, Pr.** This probability is used in two of the four phases which sorties experience each day, the restrike and redistribution of kills phases. The most obvious application is within the redistribution of kills phase where, after the BDA of the targets just killed has been determined, for each of the groups of good and mis-BDA'd targets, this probability is used to determine which ones will go into a regeneration, or repair category, and which will go into a dead category.

The other application, which is not so obvious but still modeled within TIME STRIKE, is in the restrike phase. If the category chosen from which the restrikes will come, which is randomly decided in SimStrike, is from the DeadR category, we do not have to worry about this probability, or any probability for that matter because these targets will, with probability 1.0, become dead forever, or DeadN, when they are restruck. However, if the category chosen from which restrikes will come is either LiveR or RegenR, the probability Pr applies.

Pr is set equal to the repair proportion read in for each target type, which is data to both TIME STRIKE and SimStrike. When restrikes are chosen from the LiveR or RegenR category, the restruck targets are distributed as shown in Figure 1 for the probability Pr. NPr is merely the quantity $(1 - NPr)$.

(3) **Probability of correct BDA for a target, B.** This probability only exists in the decision process within the redistribution of kills phase. It is simply used to decide which of the targets just killed in the regular strike phase will have correct BDA, and which will be mis-BDA'd ($NB = 1 - B$), i.e., thought to be still alive, and must therefore be restruck the next day, even though they were actually killed. Of these two groups, it is further decided whether they will also become dead or in repair as shown in Figure 1.

c. *Description of the Four Stages associated with Sorties*

“Stages”, and “Phases”, are identical in Figure 1. These stages are passed through in the same order every day of a SimStrike run: (1) Regeneration, (2) Restrike, (3) Regular Strike, (4) Redistribution of Kills.

It is important to remember that any numbers in “{•}” in Figure 1 represent the number of targets moving along that arrow as a result of execution of that stage. The arrows point to where the targets are moving, and Figure 1 is only an *example* of what may happen in one day for a case where there are 40 targets to start with. Also, throughout a SimStrike run, it is always true that $(LiveN + DeadN + RegenN + LiveR + DeadR + RegenR = \text{Total number of targets starting the SimStrike run})$. In the example in Figure 1, this total would always be 40.

The following is a description of what happens in each of these stages:

(1) **Regeneration Stage.** This stage in SimStrike is within the days loop but outside of the aircraft loop so it is only executed once per day. This stage is executed independent of aircraft and their sorties, only operating on any targets in the RegenN or RegenR categories. As shown in Figure 1, this stage marks the beginning of a new day and represents the first thing which must be done, or decided by SimStrike for a new day, the initial conditions of which are the results of the *previous* day's actions. This is meant to model the same lag in detection of regenerated targets by one planning cycle that TIME STRIKE does, preventing it from acting on new information until the next day [Ref. 1:p. 11]. The number of targets in the RegenN category that move to the

LiveN category is determined from a Binomial(RegenN,Pc) probability (see Ref. 2 or any probability and statistics textbook for a discussion of Binomial probabilities), with the remainder left in the RegenN category. Likewise, the number in the RegenR category which will move to the LiveR category is determined from a Binomial(RegenR,Pc) probability, with the remainder left in the RegenR category.

(2) **Restrike Stage.** This and the remainder of the stages are within the aircraft loop because although they are only executed once per day, some or all of the aircraft of the aircraft type involved in the strike may fly sorties against the target type. Aircraft flying sorties against restrikes are still subject to attrition and in-flight weather aborts just as they are for regular strikes against live targets. As with regular strikes, if an aircraft is not attrited and does not experience an in-flight weather abort, it is expected to make "kills" against these restrikes. All restrikes for the target type must be done before any aircraft of any type is allowed to fly sorties against actual live targets of the target type, just as is done in TIME STRIKE.

Restrikes are accumulated from the Regular Strike phase of the previous day, or potentially carried over from more than one day ago. The latter case would mean all the aircraft fly their sorties against restrikes but cannot quite finish them all off and so do not proceed to the regular strike stage, in which case restrikes may accumulate for more than one day.

When sorties are making restrikes, there are three categories from which the accounting for the "kills" against them can come: LiveR, DeadR, or RegenR. SimStrike assumes each restrike is against a target randomly chosen from the total of all these categories. Binomial(*category*,Pr) probabilities apply to the distribution of kills for the LiveR and RegenR categories, with the targets in the DeadR category always going to the DeadN category when they are struck, as shown in Figure 1.

Aircraft will continue to fly sorties against the target type's restrikes until all are restructured (LiveR, DeadR, and RegenR all equal to zero), the maximum number of sorties possible are flown, or the maximum allowed amount of aircraft are lost due to attrition. The exit conditions of the restrike stage will then contribute to the initial conditions of the regular strike stage.

(3) **Regular Strike Stage.** This stage involves only those targets in the LiveN category. When restrikes are complete, and if there are aircraft sorties remaining to be flown, then, if these aircraft are not attrited and do not experience an in-flight weather abort, they are expected to make kills against actual live targets based on their EKS values. When the aircraft gets to this point within this stage, there are no longer any probabilities associated with the outcome of the kills it will make, as shown in Figure 1, with the exception of the random rounding of the sortie's EKS value to make it integer.

The number of kills this sortie will make is calculated and then program flow immediately passes to the redistribution stage to distribute these kills amongst the possible categories as shown in Figure 1.

(4) **Redistribution of Kills Stage.** The kills are distributed as they are made amongst the possible categories to which they can go based on the probabilities B, NB, Pr, and NPr as shown in Figure 1. Thus, the redistribution stage actually occurs successively right after the regular strike stage for each sortie, but a boolean condition prevents the restrike stage from being executed more than once per day, even though it is within the aircraft loop.

When the maximum number of sorties have been flown, the maximum number of aircraft has been lost, all the targets have been killed, or the kill goal has been achieved, the current day ends.

8. Calculation of the Objective Function Value

In TIME STRIKE, two general quantities are calculated to be used in the overall objective function value calculation: the TVD reward gained, and the penalties for not meeting the time-scripted kill goals in each period, if any [Ref. 1:p. 24]. TIME STRIKE takes credit in each period for the target value of each target for the current period, times the total number of each target currently dead or in repair. Penalties against TVD gained are calculated from a set of known kill goals, which are data to TIME STRIKE and are cumulative over time. The kill goals constrain the total targets dead or in repair through the end of a particular period to a proportion of the total targets originally present at the start of the campaign. TIME STRIKE is allowed to kill targets up to the goal, but not over the goal in each period. Any difference below the goal is taxed against TVD by summing the weighted proportional differences over all periods for all target types.

SimStrike calculates the same objective function as TIME STRIKE. The simulation routine within SimStrike is run once for each of TIME STRIKE's sortie variables obtained from the output data file "TSVTST.CSV". At the end of a run, just before program flow passes back to the main program within SimStrike, the TVD reward is collected by multiplying the target value applying to the current period for the current target type by the quantity (DeadN + RegenN + DeadR + RegenR), which represents all the targets of the target type dead or in repair at this point in time.

B. SUMMARY OF COMPARISONS OF TIME STRIKE AND SIMSTRIKE

Table 1 on the following page provides a summary of the comparisons made between TIME STRIKE and SimStrike in section A.

Category	TIME STRIKE	SimStrike
Constraints on Sorties	Fractional sorties and targets	Integer sorties and targets
Distribution of Sorties	Deterministically uniform over periods	Randomly uniform over periods
EKS	Fractional numbers of kills	Integer numbers of kills
Attrition Per Sortie	Fractional losses of "aircraft"	Integer losses of aircraft
Kill Goals	Cumulative proportion of original number of targets expected to be dead	Cumulative proportion of original number of targets expected to be dead
Time Periods and Planning Cycles	Periods and days; mis-BDA'd targets become restrikes the next day with all restrikes taken care of on the day they are assigned	Periods and days; mis-BDA'd targets become restrikes the next day with the possibility existing that not all restrikes will be taken care of on the day they are assigned, i.e., may carry over one more day
BDA and Target Regeneration	Treated as proportions	Treated as probabilities
Calculation of the Objective Function Value	TVD taxed by penalties for not meeting kill goals	TVD taxed by penalties for not meeting kill goals

Table 1 - Comparison Summary of TIME STRIKE and SimStrike

III. RESULTS

A. RESULTS

Two simplifications were made in SimStrike for the purposes of this thesis. The first was to make all aircraft arrive on the first day of the campaign. For this, the data files were matched to have the total amount of aircraft of each aircraft type simply be read in as being there the very first day of the first period. The second was to run TIME STRIKE and SimStrike over only one Major Regional Conflict (MRC). Therefore, the attribute of allowing aircraft swings between two MRCs was not modeled.

A model data set was provided by Maj. Kirk Yost, USAF, (TIME STRIKE's author) for an example Air Force simple strike scenario. The scenario the input data sets to TIME STRIKE represent are groups of known targets and their locations at various distances from the sortie origination point, a given set of aircraft, weapons, crew qualifications, weather forecast, kill goals, etc., meant to represent just one of many possible MRC's. The data set was unclassified, but representative of realistic data in use. For this data set, TIME STRIKE produced an answer of 4,140,022 "points". Using the same data set, after 1000 replications, SimStrike produced an (mean) answer of 4,010,561 points with a (sample) standard deviation of 245,326. The answers produced at each replication of SimStrike are plotted on a histogram in Figure 2 on the next page. Assuming TIME STRIKE is producing the true mean and SimStrike is producing the sample mean and standard deviation for a large sample (large number of replications), the Sample t-Test [Ref. 2:p. 322] with a null hypothesis of $H_0: \mu = \mu_0$ and corresponding alternative hypothesis $H_a: \mu \neq \mu_0$ produced a test statistic value of 16.7. This is enough to reject the hypothesis that TIME STRIKE and SimStrike produce the same results at virtually any level of confidence desired.

The question now is: What does this mean for TIME STRIKE? Does it lead one to conclude that the model is no good? The answer to this last question is emphatically NO. After all, the objective functions differ by only 1%. TIME STRIKE's problems are incurable, but not serious, and these results only lead to the need to study the way the LP models reality a little closer.

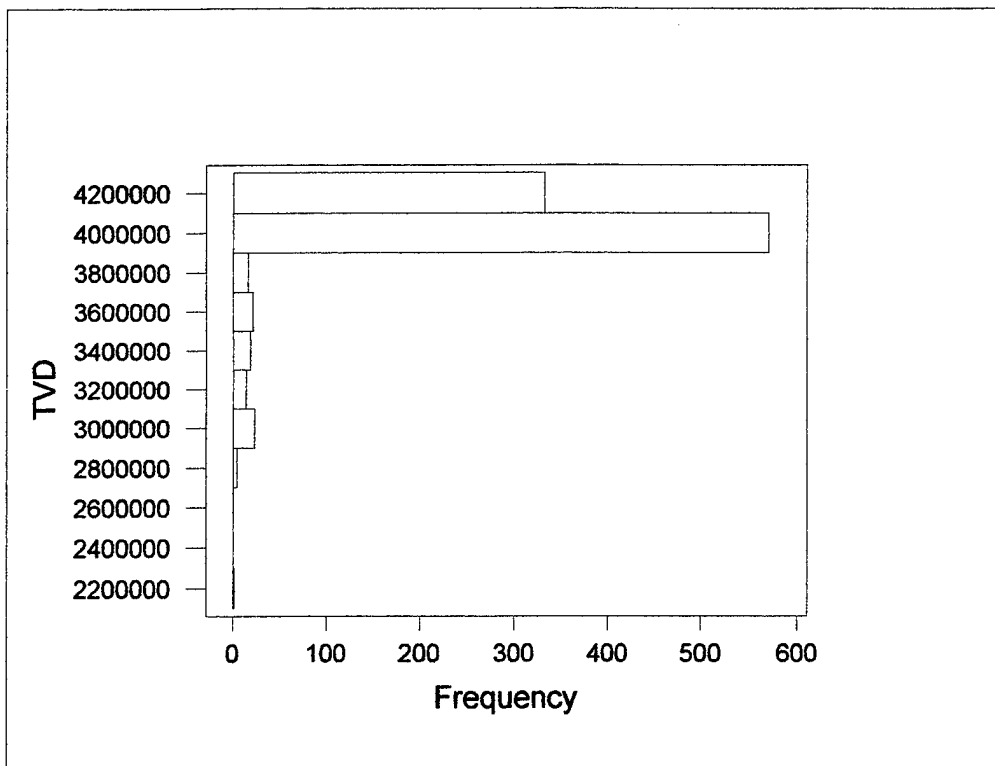


Figure 2 - SimStrike Objective Function Value Histogram for 1000 Replications

B. SOURCES OF DIFFERENCE

This section will discuss some explanations of why TIME STRIKE and SimStrike results differ.

1. Period Transitions and Restrikes

In TIME STRIKE, within the same period, any targets killed today which carry over to tomorrow as restrikes due to bad BDA, must be restruck before any further live targets of the same target type are allowed to be struck. A problem occurs when the restrikes carry over from the last day of one period to the first day of the next period, in which case the model does not allow sorties to be flown against them due to the problems of mathematical complexity at the time the model was developed. Instead, it discounts TVD by the number of restrikes which would have to be done, were they allowed to be, times the target value of this target type for the *new* period. The restrikes are then essentially discarded and regular sorties are flown against live targets of the target type. To reiterate, *this only occurs at period transitions within TIME STRIKE.*

In SimStrike, period transitions are transparent as far as restrikes are concerned. If restrikes carry over from the last day of one period to the first day of the next period, they are restruck first, as they would be in any other day, before sorties are flown against actual live targets of the target type. SimStrike essentially views all the periods as actually just a string of days, which happen to transition through periods, only using the current period number to determine which data gets manipulated with other variables, e.g., target values, kill goals, etc. SimStrike merely lets restrikes randomly play out over time. This accounts for only a small numerical difference between the results of SimStrike and TIME STRIKE as shown in Table 2 below, and only a slight difference from the result SimStrike produced as noted in Section A above.

Also shown in Table 2 is a sensitivity analysis for four other special cases performed to compare results between SimStrike and TIME STRIKE. As can be seen, the results all closely agree quantitatively, except for a target rich environment, Case 4, which is explained further in the next subsection below. The scaling factor for a target rich environment is the reduction from the base case, 1.0, which is no change in the original number of aircraft present, i.e., not target rich or target poor. For example, the scaling factor shown in Table 2 for Case 4 means the base case, 1.0, divided by 2, or half the number of aircraft originally present, to produce a scaling factor of $(1.0 / 2 =) 0.5$.

Case	TIME STRIKE Objective Function Value	SimStrike Objective Function Value	SimStrike Standard Deviation
(1) No restrike carry-over at period transitions	4,140,022	4,000,164	236,407
(2) Perfect BDA	4,235,479	4,020,925	201,105
(3) All target values set to 4.0	689,913	539,358	33,881
(4) Target rich environment for scaling factor of 0.5	4,115,600	3,774,709	278,424
(5) Target poor environment	4,162,430	4,125,762	277,408

Table 2 - Sensitivity Analysis

2. Target Rich Environment

Target rich environments were created by reducing the number of aircraft present to fly against the same number of targets in the original data set. The case shown in Table 2 is for half the original number of aircraft, or a scaling factor of 0.5. Several target rich environments were produced and results plotted in Figure 3 on the next page for TIME STRIKE and SimStrike. A scaling factor of 1.0 means the original number of aircraft present, or the base case. Any scaling factor less than 1.0 is a target rich environment, and anything above 1.0 would be target poor, which is not shown in Figure 3 because TIME STRIKE and SimStrike closely agree quantitatively for a scaling factor of 1.0 and above.

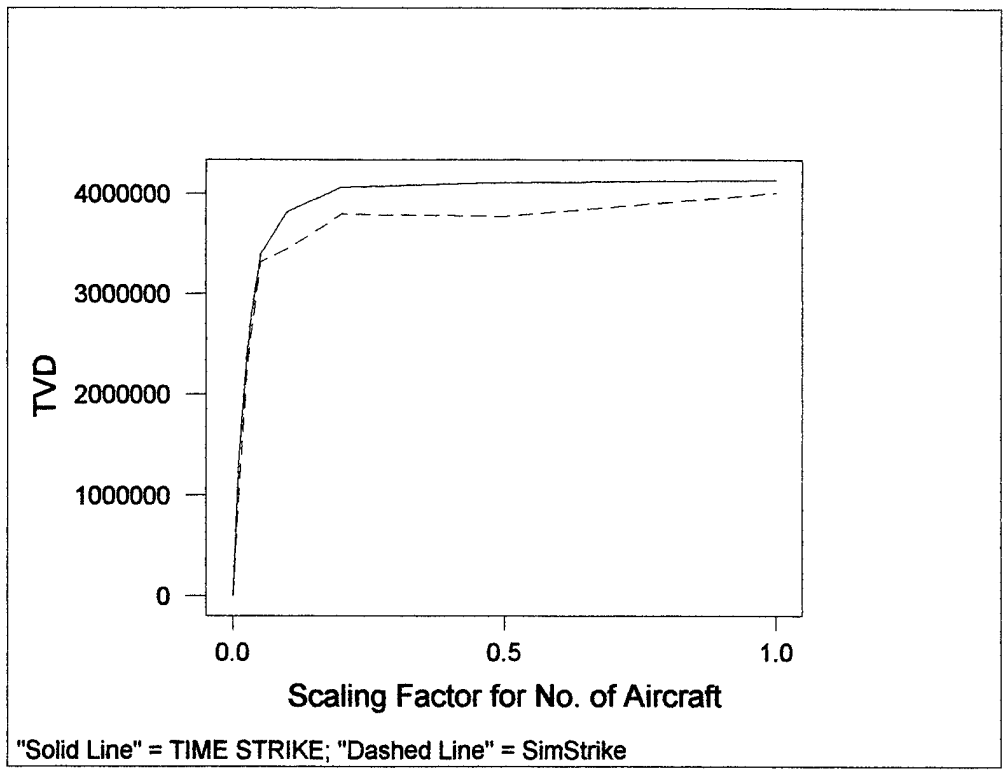


Figure 3 - Comparison of TIME STRIKE and SimStrike for a Target Rich Environment

Figure 3 shows a noticeable difference between results for TIME STRIKE and SimStrike for a target rich environment, but then rapidly approaching one another as a scaling factor of 1.0 is approached. The data used to plot Figure 3 is shown in Table 3 below. Although not shown in Figure 3, the data for the target poor environment case of a scaling factor of 2.0 is shown in Table 3 for comparison purposes.

Scaling Factor	TIME STRIKE Objective Function Value	SimStrike Objective Function Value	SimStrike Standard Deviation
0.000	0	0	0
0.010	1,338,622	1,019,691	100,006
0.025	2,362,772	2,217,805	89,532
0.033	2,762,169	2,612,735	75,176
0.050	3,390,982	3,319,477	74,465
0.100	3,823,302	3,449,200	58,398
0.200	4,066,308	3,798,582	105,531
0.500	4,115,600	3,774,709	278,424
1.000	4,140,022	4,010,561	245,326
2.000	4,162,430	4,125,762	277,408

Table 3 - Target Rich Environment Data for Figure 3

3. Proportioning of Sorties Over Time and Restrikes

When TIME STRIKE decides what sorties will be flown in a campaign, it looks at its constraints and data, and then places a certain number of sorties to be flown by the various aircraft against the various targets in every day of every period. A portion of the sorties which have been selected to fly in every day of the campaign are there to account for restrikes. The extra number within the total allotted due to the restrike requirement is to *exactly account for* these restrikes, i.e., all the restrikes will be taken care of in their

entirety each day before the aircraft go on to fly sorties against live targets, and no restrikes will carry over to the next day.

SimStrike does not recognize a certain number of the sorties to be flown on any particular day as being present solely for restrike purposes, as TIME STRIKE does. How many sorties will be needed to clear the restrikes is decided randomly, so it is possible that more or less sorties, if any, will be dedicated to restrikes than were in TIME STRIKE on any given day of the campaign.

The potential source of difference here lies in the possibility that when SimStrike is doing restrikes, if a small number of aircraft are flying sorties at low sortie rates and low EKS values against the restrikes, not all the restrikes may be cleared in a day. Therefore, there may be some restrikes which carry over to the next day, which could lead to more or less being done on some particular day in SimStrike than TIME STRIKE would expect for its circumstances. This situation arises because the new day's restrikes in SimStrike would now include carry over restrikes from yesterday which were not completed.

Experience with SimStrike, however, does show that most restrikes are done on the day they are encountered, and very few carry over to the next day. For example, if we look at the detailed statistical output for the first replication at the beginning of the base case output for SimStrike contained in Appendix B, the total number of restrikes which carry over from the last day of each period to the first day of the next period is shown in Table 2 below.

Period	Carry-over Restrikes
1	38
2	36
3	34
4	41
5	29

Table 4 - Example of Carry-Over Restrikes

The carry-over restrikes shown in Table 2 are the totals over all target types in each period, and when compared to the total targets of each type present and the total dead or in repair each period, the quantities in Table 2 represent well less than 1% of these totals.

C. CONCLUSIONS

The differing results between SimStrike and TIME STRIKE are cause to examine closer TIME STRIKE's modeling of reality. Although many estimates are being made, perhaps within the BDA equations in the optimization model, these hurdles are not insurmountable. The objective function values obtained by TIME STRIKE and SimStrike are similar, although the hypothesis they produce the same expected measure of effectiveness (MOE) is rejected. The sources of difference discussed in the previous section do not contribute significantly to the differing results, so the source of deviation must lie elsewhere.

SimStrike was developed to model all the same things TIME STRIKE does, but to use probabilities where TIME STRIKE uses expectations. As shown in Table 1 in Chapter II, these probabilities were applied at the appropriate places where reality comes into play, while the integrity of other factors such as the distribution of sorties over periods, kill goals, and the calculation of the objective function value, were maintained exactly the same as TIME STRIKE. Therefore, somewhere within the many expectations used by TIME STRIKE for the purposes of optimization lies one or more approximations which may be too abstract. Thus, it is a matter of identifying and refining one or more of these expectations which is in order.

IV. FUTURE WORK

A. WEATHER

TIME STRIKE currently assumes that weather is known before profiles are selected for given aircraft types [Ref. 1:p. 12]. Expectations for six weather states are read in as data and correlated with what weapons and tactics can be used for a given weather state against a certain target type, then the sorties are assigned. SimStrike uses the same expectations as data, then as a probability comparison to a uniform random number to see if there will be an aircraft sortie weather abort given the present weather state. TIME STRIKE and SimStrike are therefore both relying on the accuracy of the *same* forecast, and therefore both miss the true randomness of nature.

A better model might be one in which weather is decided randomly *each day*. Of course linear programs do not have random number generators, so this is not an option for TIME STRIKE. But one could take SimStrike a step further (it would actually be a giant step in this case) and randomly decide weather on each day, and then pick the weapons and tactics on a day-by-day basis for available aircraft to fly sorties against targets. However, this kind of simulation would not read data in from the solution variables produced by a run of TIME STRIKE, it instead would be generating its own sorties over time, so it would therefore be a re-work of SimStrike.

B. PROBABILITY DISTRIBUTION FOR RANDOM ROUNDING

Currently, SimStrike uses a uniform random number to determine whether a non-integer value gets rounded up or down. Using this consistently where rounding is required means that, over a large number of replications, these rounded values average out to the non-integer value from TIME STRIKE. This rounding arises for sorties flown, EKS, and sortie rate values in SimStrike, a uniform random number being used to ensure SimStrike models TIME STRIKE as closely as possible.

One might argue that this rounding may be better done using a Poisson distribution if the number of sorties flown as read from the file of solution variables produced by TIME STRIKE are viewed as events occurring over fixed time intervals. In this case, since the outcome is based on the total number of draws, or sortie occurrences,

to be made over time, the rounded values will not necessarily average out as closely to what their original non-integer values were as in the uniform random number case above.

APPENDIX A: PROGRAM LISTING AND DIRECTIONS FOR USE

SimStrike is coded in Turbo Pascal for Windows, but is compatible with Turbo Pascal for DOS as long as the "Uses WinCrt;" code line is commented out. SimStrike uses four input data files and produces one output file which can be opened by almost any spreadsheet application, however, our preference is Excel. Only one of the four data input files is "homemade" because it contains only selective data required by SimStrike which is contained in a much, much larger data input file to TIME STRIKE. The other three data input files are merely the same ones used by TIME STRIKE, but with the filename extension changed to make it useable by the Turbo Pascal compiler.

For the output, the user has several options as annotated in the comments within the main program part of SimStrike. The output can take many forms, very limited or detailed based on what the user comments out or keeps in, but is set up to provide the most detailed statistics as listed in this Appendix, with an example output contained in Appendix B. The very last three lines of any output, however, will be the number of replications ran, the mean objective function value, and the standard deviation if the number of replications is greater than or equal to 30.

Before running the program, there is a note in its comments at the very beginning of the program listing which must be heeded. Since the data segment created by the program is very large, the default values for the stack and heap sizes for the Turbo Pascal compiler are not of sufficient sizes. These should be manually set to 15,000 bytes each by the user before the program is compiled, or a stack overflow runtime error will result. Setting the stack and heap sizes too large, however, may prevent compilation if the machine does not have enough memory to support it. A typical 486 or Pentium PC with 16MB of RAM will do a quick job of running SimStrike with a compiler stack and heap size of 15,000 bytes.

SimStrike has been designed to be as robust as possible, but it is assumed that only a user somewhat knowledgeable of TIME STRIKE will actually be using it, so it is biased toward that end. Most terms used in SimStrike are those which would normally be known by those familiar with TIME STRIKE, and the detailed comments in the beginning and throughout the program listing are geared towards these users. Also, there is not complete data input error trapping, especially associated with some of the more larger data input files, because it is assumed only meaningful data will be used, and

would also cause an especially excessive additional amount of code to be added to an already very long program.

This Appendix will now describe in detail how to use SimStrike, the program listing for which can be found on the following pages. It may be helpful to refer to the program listing as these directions are read. To reinforce the directions, the data input file "sim2in.dat", which is used by SimStrike, but *not* by TIME STRIKE, is presented after the program listing. This input file contains specific data from one of the larger data files used by TIME STRIKE.

Conditions to be set prior to executing the directions for running SimStrike:

- Set the STACK and HEAP sizes of the Turbo Pascal compiler on the machine on which SimStrike is to be run to 15,000 bytes (minimum) each. This is needed so the data segment created by SimStrike is accepted by the compiler, otherwise there will be a stack overflow runtime error. Note: Do not go overboard with the STACK and HEAP sizes. Setting them too large will prevent the compiler from compiling if the machine the program is run on does not have enough memory to support it.
- Ensure the simulation program and the data files it will use all reside in the same directory. The output file will also be placed in this directory when it is created. Set the Turbo Pascal default directory to this directory if not already done so.
- Ensure the data input file "sim2in.dat" in SimStrike's directory contains the same requisite data (see the listing later in this Appendix for details) as "stand13.dat" in TIME STRIKE's "Data" directory.
- Ensure the target values in the data input file "tgtval22.txt" in TIME STRIKE's "Data" directory and in the data input file "tgtval22.dat" in SimStrike's directory are the same. The "tgtval22.dat" file used by SimStrike may be a copy of "tgtval22.txt" used by TIME STRIKE, with file extension changed, except that the "MRC1." preceding each target type number must be deleted before it is useable as an input file to SimStrike. The same requirements apply to the files "tgtdat22.txt" in TIME STRIKE's "Data" directory and "tgtdat22.dat" in SimStrike's directory, also ensuring all the target data is the same for both.
- Copy the file "tsvtst.csv" from TIME STRIKE's "Results" directory to the directory the simulation program is in. Change its file extension to ".dat", and then open it with Microsoft Wordpad (a common application on any PC with

any version of Windows installed) and remove the text lines and extra blank lines preceding each set of numerical data to which they apply. If there are any extra blank lines at the end of the file, remove these also. Note: Another application can be used to open and edit "tsvtst.dat" as long as the font is set to "Courier" or "Courier New". In either case, make sure to save it when done editing.

Remember that there is no data input error trapping, so one must pay attention to the above directions and make sure they are followed exactly so input errors or bad results do not occur. In most cases it is simply a matter of copying files from one directory to the next, then just changing the file extension (and removing the "MRC1." text in some cases).

Directions for running SimStrike:

- If the program will be run via the DOS version of Pascal, then the code line "Uses WinCrt;" must be commented out. This is the first line of program code after the initial comments.
- Scroll down in the program listing to the constant declarations (code line "Const"). Here you may set the values of five of the constants used in the program: the number of replications, the seed for the random number generator, and three replication output instances. Note: The random number generator does not have to be seeded. The "Randomize;" code line given in the main program part can be used instead, which seeds the random number generator with the system clock on the machine SimStrike is run on, and the "RandSeed" code line can be commented out. When using the Randomize function, however, the string of random numbers produced will not be repeated each time SimStrike is ran. When setting a replication output instance remember that these will be the instances out of all the replications which will be output in detailed statistical tabular format. If only one replication, or none, of these instances are desired to be included in the output, set the values in the "Const" declaration section to instances which are higher than the number of replications which are being ran.
- Now scroll down to the main program part of SimStrike. Here will be found several comment blocks, which, among other things, explain how to change the output into other formats depending on one's desire. SimStrike "comes from the factory", per se, with the most detailed and useful output form, outputting one statistical tabular replication instance, along with objective function values for each replication, and the final statistics, all in Microsoft

Excel spreadsheet format. The output file which is created is “sim2out.csv”, and can potentially be opened by other spreadsheet applications which accept this format.

At this point SimStrike is ready to be run. Be advised that a previous version of “sim2out.dat”, which may be open for viewing in a spreadsheet application, must be closed *before* actually running SimStrike because a runtime error will occur due to the inability of SimStrike to write to the output file if it is still open. Also, there are detailed comments at the beginning and throughout the program listing which provide significant help in interpreting sections of program code and aid in getting it set up for running.

The program listing starts on the next page, and the listing for “sim2in.dat” can be found on the pages immediately following the program listing.

Program Simulation2;

{
Programmer: John J. Kosina
Thesis Advisor: Prof. Alan R. Washburn
Operations Research Department
Naval Postgraduate School, Monterey, CA

Revision Date: 7/15/97

Purpose: Monte Carlo simulation used to conduct error analysis on the Air Force's Time Strike Optimization model using some modifications to certain variables and data (explained below) used in objective function 4 to maximize the weighted sum of TVD. GAMS model is run in parallel to compare outputs.

Notes: (1) "sim2in.dat" data input file must contain data in the form:

1st line:
TVDWGT, GOALWGT

2nd line:
number of periods, number of days in each period

3rd line:
number of aircraft types

Each of following lines up to the total number of AIRCRAFT TYPES, one line per aircraft type:
aircraft type, number of aircraft for aircraft type starting, max loss allowed for the aircraft type, sortie rate (sorties per aircraft per day) for the aircraft type

Next line:
number of target classes

Each of following lines up to the total number of TARGET CLASSES, one line per target class:
proportion of targets in target class to be killed to achieve the goal for each time period

Each of following lines up to the total number of TARGET CLASSES, one line per target class:
objective function penalty for not meeting the time-scripted goal for the target class (as given in the data block just prior to this one) by the end of the period for each time period

Each of following lines up to the total number of TARGET CLASSES, one line per target class:
targets which belong to the target class

(2) "tgtval22.dat" data input file is target values for each target type by period in which they are killed. These values may be randomly generated for all the target types as long as the same data values are used for both the GAMS model and simulation runs.

(3) "tgtat22.dat" data input file is target data by distance band (used to obtain total targets of each type), bda probability, repair time, percent of killed targets that regenerate. Target Elements are not used by the simulation. This is the same data as those used by the GAMS model except that target data by distance band must be integer (therefore will be the same for both GAMS model and simulation), and repair times must be integer (also same for both).

(4) "tsvtst.dat" data input file is conversion of GAMS model output file "tsvtst.csv". This file contains the solution data and variable values for the last run of the GAMS model.

When the GAMS model is run for a certain data set, the "tsvtst.csv" file from the GAMS model Results directory should be copied to the directory containing the simulation and renamed to "tsvtst.dat". After deleting the short text lines and any extra blank lines at the beginning of the long lines of numerical data for each period, data may be used as is. Even though they are real values separated by commas, which is not a proper data format for Turbo Pascal, this simulation has a Procedure which effectively

picks out the pertinent data and converts them to either integer or real values based on the use.

Pertinent data from this file which is stored and used by the simulation is: time period, aircraft type flying sorties, target type sorties are flown against, EKS per sortie, probability of no weather aborts (NABORT) for the particular sorties, attrition per sortie, and sorties flown.

All data is used "as is", with the exception of sorties flown, which are reported as real valued quantities by the GAMS model. For this simulation we are interested in integer values (since you can only fly a whole number of sorties in real life!), therefore, after the real number of sorties flown is uniformly distributed over the current period, another Procedure is used to round this either up or down, based on a U[0,1] random number, to an integer value. The integer number of sorties is then the quantity used by the simulation. When a kill is made by a sortie, the same is done to the EKS value for that particular sortie since you can only kill an integer number of targets in real life!

- (5) Output will be the objective function value after a large number of replications (see NUMBERREPS below in CONST declarations). Extra output statistics can also be obtained by removing comment brackets from those places noted in the Main Program part of the Simulation below. Output is sent to "sim2out.csv", which can be viewed in Excel spreadsheet format. There are code options to allow output to go to the computer screen also.

- (6) Data which should be the same as the GAMS model LP is:

time period
aircraft types
target types
EKS per sortie
probability of no weather aborts
attrition per sortie
number of aircraft starting for each type (integer)
max loss allowed for each aircraft type (integer)
sortie rate for each aircraft type
TVDWGT
GOALWGT
number of periods
number of planning cycles (days) in each period
kill goals for each target class
number of targets starting for each type (integer)
penalties for not achieving kill goals for each target class
target class-target correspondence
repair proportion/probability for each target type
probability of correct BDA for each target type

- (7) Probabilities used in the simulation are (GAMS model data or variable name correspondence, if any, is shown in parentheses next to each):

aircraft attrition (ATTR)
prob. of no weather abort (NABORT)
target BDA prob., i.e. prob. of correct BDA for a target (BDAPROB, B (as used in target regeneration and BDA equations))
prob. of incorrect BDA for a target (NB (= 1 - B as used in target regeneration and BDA equations))
prob. the target is repairable after a strike (REPROP, Pr (as used in target regeneration and BDA equations))
prob. a target is not repairable (NPr (= 1 - Pr as used in target regeneration and BDA equations))
prob. a target regenerates in the next planning cycle (Pc (as used in target regeneration and BDA equations))
prob. a target does not regenerate in the next planning cycle (NPc (= 1 - Pc as used in target regeneration and BDA equations))
conversion of real no. of sorties flown to integer (x)
conversion of real valued sortie rates to integer (SRTTRTPER)
conversion of real no. of targets killed from associated EKS to integer (EKS)
choice from which category restrikes will come

- (8) Output generated by this simulation to be compared with LP:

mean objective function value for weighted sum of target value

destroyed (TVD) over a large number of replications

Additional output generated by simulation:

standard deviation for mean objective function value
number of replications done

- (9) Data input files are "simlin.dat", "tgtval22.dat", "tgtat22.dat", and "tsvtst.dat". Output file is "sim2out.csv".
- (10) Note that the MAXLOSS values in the GAMS model input file "stand13.dat" must also be rounded to integer values so the GAMS model and simulation both use the same data. This is also true of the number of aircraft and targets of each type in their respective data input files.
- (11) There is no data input error trapping so data in input files must be entered correctly and also be meaningful for proper output to be generated. Basically, a user knowledgeable of the GAMS model and simulation is assumed here!
- (12) *** IMPORTANT! *** Set the STACK and HEAP sizes of the compiler on the machine this is run on to 15,000 bytes (minimum) each. This is needed so the data segment created by this program is accepted by the compiler, otherwise there will be a stack overflow runtime error. Note: Don't go overboard either! Setting the STACK and HEAP sizes too large will prevent the compiler from compiling if the machine this is run on does not have enough memory to support it.

}
Uses WinCrt;

```
Const
NUMBERREPS = 1000;    {set number of replications here}
SEEDFORRANDBANDS = 999; {seed for random numbers so they are the same each time if needed}
MAXNUMPERIODS = 7;    {max number of periods}
MAXNUMACFTTYPES = 10; {max number of aircraft types expected to be encountered}
MAXNUMTGTTYPES = 87;  {max number of target types}
MAXNUMTGTCLASSES = 10; {max number of target classes}
MAXNUMDISTBANDS = 7;  {max number of distance bands}
REPOUT1 = 1;          {replication output instance}
REPOUT2 = 1057;       {replication output instance}
REPOUT3 = 1092;       {replication output instance}
```

```
Type
PeriodIndexRange = 1..MAXNUMPERIODS;
AcftTypeIndexRange = 1..MAXNUMACFTTYPES;
TgtTypeIndexRange = 1..MAXNUMTGTTYPES;
TgtClassIndexRange = 1..MAXNUMTGTCLASSES;
AcftTypeArray = Array[AcftTypeIndexRange] Of Integer;
SortieRateArray = Array[AcftTypeIndexRange] Of Real;
TgtValueArray = Array[TgtTypeIndexRange, PeriodIndexRange] Of Real;
TgtKillArray = Array[TgtTypeIndexRange, PeriodIndexRange] Of Integer;
TgtRealTypeArray = Array[TgtTypeIndexRange] Of Real;
TgtIntTypeArray = Array[TgtTypeIndexRange] Of Integer;
KillGoalArray = Array[TgtClassIndexRange, PeriodIndexRange] Of Real;
TgtClassTgtArray = Array[TgtClassIndexRange, TgtTypeIndexRange] Of Integer;
TgtClassArray = Array[TgtClassIndexRange] Of Integer;
StringType = String[10];
```

```
Function UniformProb: Real;
{
A uniform [0,1] random number generator available for any procedure or
function which requires a U[0,1] probability.
```

```
Pre: None.
Post: U[0,1] probability.
```

```
}
Begin {Function UniformProb}
UniformProb := Random;
End; {Function UniformProb}
```

```
Function Binomial (Num: Integer;
Prob: Real): Integer;
```

```
{
Determines the number of possibilities from a certain number of samples, Num, which
meet the condition  $U[0,1] \leq \text{Prob}$ , where Prob is a particular probability from
```

```

any particular distribution desired which was calculated previously and passed to
this Function.

Pre:  Num = total number of samples (must be > 0); Prob = probability for a U[0,1]
      to be compared against.
Post: Binomial = number out of the total for which U[0,1] <= Prob is true.
}
Var
  I, Count: Integer;

Begin {Function Binomial}
  Count := 0;
  For I := 1 To Num Do
  Begin
    If (UniformProb <= Prob) Then
    Begin
      Count := Count + 1;
    End; {If}
  End; {For}
  Binomial := Count;
End; {Function Binomial}

Function GetInteger (Num: Real): Integer;
{
  Converts a Real number to an Integer number by rounding either up or down based
  on a U[0,1] random number.

  Pre:  Num = Real number
  Post: GetInteger = Integer number
}
Var
  Result: Integer;

Begin {Function GetInteger}
  If (UniformProb <= Frac(Num)) Then
  Begin
    Result := Trunc(Num) + 1;
  End
  Else
  Begin
    Result := Trunc(Num);
  End; {If}
  GetInteger := Result;
End; {Function GetInteger}

Function Raise (Base: Real;
               Exponent: Integer): Real;
{
  Raises the Base to the Exponent power.

  Pre:  Exponent > 0
  Post: Raise = Base to the Exponent power
}
Var
  Count: Integer;
  Product: Real;

Begin {Function Raise}
  Product := 1;
  For Count := 1 To Exponent Do
  Begin
    Product := Product * Base;
  End; {For}
  Raise := Product;
End; {Function Raise}

Procedure GetRealNumber (St: StringType;
                       Var x: Real);
{
  Converts a string to it's real valued number. Only handles positive numbers, and
  if the number will be < 1, there must be a leading zero before the decimal point,
  e.g. St = '0.0394' vice St = '.0394'.

  Pre:  St = a string of type StringType.
  Post: x = the real valued quantity imbedded in St.
}
Var
  I: Integer;

```

```

Num1, Num2: Real;
OnPeriod: Boolean;
St1, St2: StringType;

Begin {Procedure GetRealNumber}
  Num1 := 0.0;
  Num2 := 0.0;
  St1 := '';
  St2 := '';
  OnPeriod := True;
  For I := 1 To Length(St) Do
  Begin
    If ((St[I] <> '.') And (OnPeriod)) Then
    Begin
      St1 := St1 + St[I];
    End
    Else
    Begin
      If OnPeriod Then
      Begin
        OnPeriod := False;
      End
      Else
      Begin
        St2 := St2 + St[I];
      End; {If}
    End; {If}
  End; {For}
  For I := 1 To Length(St1) Do
  Begin
    Num1 := Num1 + (Ord(St1[I]) - 48) * Raise (10, (Length(St1) - I));
  End; {For}
  If (Length(St2) > 0) Then
  Begin
    For I := 1 To Length(St2) Do
    Begin
      Num2 := Num2 + (Ord(St2[I]) - 48) * Raise (10, (Length(St2) - I));
    End; {For}
    Num2 := Num2 / (Raise (10, Length(St2)));
  End; {If}
  x := Num1 + Num2;
End; {Procedure GetRealNumber}

Function GetTargetClass (Tgt: Integer;
  TgtClassTgt: TgtClassTgtArray;
  TgtsInClass: TgtClassArray;
  NumTgtClasses: Integer): Integer;
{
  For a given target type, determines which target class it belongs to.
  Pre: Target type in question, target class data arrays, and number of target classes.
  Post: GetTargetClass = Target class the target belongs to.
}
Var
  Class, Target, Temp: Integer;

Begin {Function GetTargetClass}
  For Class := 1 To NumTgtClasses Do
  Begin
    For Target := 1 To TgtsInClass[Class] Do
    Begin
      If (TgtClassTgt[Class,Target] = Tgt) Then
      Begin
        Temp := Class;
        Target := TgtsInClass[Class];
        Class := NumTgtClasses;
      End; {If}
    End; {For}
  End; {For}
  GetTargetClass := Temp;
End; {Function GetTargetClass}

Procedure RunSimulation (NumDays, NumAcft, MaxLoss: Integer;
  SortieRate: Real;
  Var AcftLost: Integer;
  TotTgts: Integer;
  TgtVal, Goal: Real;
  Eks, Nabort, Attr, x: Real;

```

```

Var TotSortiesFlown: Integer;
Pr, NPr, Pc, NPC, B, NB: Real;
Var LiveN, DeadN, RegenN: Integer;
Var LiveR, DeadR, RegenR, deltaTgtsKilled: Integer;
Var TempTvd: Real);

```

{ For the given mission data, runs the simulation and determines the Target Value Destroyed (TVD). This Procedure is run for one aircraft type, flying a certain profile, against a particular target type, with sorties distributed uniformly over one period. The simulation loops through the days in the period and the number of aircraft in the aircraft type until all sorties are flown, OR all the targets of the target type are killed, OR the max number of the aircraft type are lost, OR the kill-goal for the period has been achieved. It is important to note that this Procedure is run once for each line of data in "tsvtst.dat". It essentially re-flies the sorties the GAMS model flew and determines the outcome randomly for later comparison with the GAMS model results.

Pre: See the parameters passed to the Procedure above.

Post: For a particular engagement flown, updates aircraft of the aircraft type lost, targets of the target type killed, future restrikes and repairable targets, future regeneration of targets, and total TVD.

Variable definitions:

 *** For parameters passed to the Procedure, see Main Program variable definitions as the same names for them were used.

AcftKilled - stores a Boolean value for whether or not an aircraft is attrited.
 AcftWxAbort - stores a Boolean value for whether or not an aircraft experiences an in-flight weather abort.
 delta - used generically to determine an incremental change in some integer variable.
 deltaB - used to store the number of kills from the total amount made as a result of a regular strike based on the probability "B" which will go to the "non-restrikeable" category (become either DeadN or RegenN).
 deltaBNPr - used to store the number of kills from deltaB which will become DeadN based on the probability "NPr".
 deltaBPr - used to store the number of kills from deltaB which will become RegenN based on the probability "Pr".
 deltaNB - used to store the number of kills from the total amount made as a result of a regular strike based on the probability "NB" which will go to the "restrikeable" category (become either DeadR or RegenR).
 deltaNBPr - used to store the number of kills from deltaNB which will become DeadR based on the probability "NPr".
 deltaNBPr - used to store the number of kills from deltaNB which will become RegenR based on the probability "Pr".
 deltaPd - used to store the number of periods ahead of the current one that a target will be regenerated in.
 I - counter variable for loops.
 InRegularStrikePhase - boolean variable which indicates whether the regular strike phase has been entered or not in the current day so as to prevent the restrike phase from occurring more than once per day.
 K - counter variable for loops.
 MaxSortiesToday - the integer max number of sorties which may be flown today based on sorties per day as derived from the sorties flown for the period read in from "tsvtst.dat".
 NumTgtsKilled - integer number of targets killed by a successful sortie based on EKS.
 PropKilled - cumulative proportion of targets of the target type killed.
 RTypeTotal - total number of targets in all restrike categories which are currently subject to restrike.
 SortiesFlownToday - counter to keep track of the sorties flown on the particular day for comparison to see if the max sorties for the day have been flown.
 SortiesPerAcftPerDay - sortie rate for each aircraft of the aircraft type.
 SortiesPerDay - sorties per day, uniformly distributed over the period.
 SortiesToFly - the sorties which will actually be flown on a particular day.
 Start - the starting point for the For loop over the number of aircraft of the aircraft type.
 TempPropKilled - temporary proportion of the total number of targets killed; calculated in advance to compare to the kill-goal before kills are actually applied to the LiveN type targets.
 WhichRType - random variable to aid in deciding from which restrike category the kills held in the variable NumTgtsKilled will be applied to.

Notes for below:

 (1) This uniformly distributes the number of sorties to be flown over the period. The integer number of sorties actually flown on any day is found using

SortiesPerDay in the Function GetInteger. SortiesPerAcftPerDay is the sortie rate for each aircraft of the particular aircraft type.

- (2) While there are sorties left to fly in this day, and the max number of aircraft have not been lost, and the total amount of targets in this type have not been killed, and the kill-goal for the period has not been met, sorties may still be flown.
- (3) If an aircraft is attrited, it is lost, and therefore any sorties it had left to fly are also lost.
- (4) If an aircraft is not attrited and does not experience an in-flight weather abort, it is expected to make kills based on EKS. However, a sortie cannot kill more targets than the total amount of the target type present (common sense!), therefore this is also accounted for and the number of kills from the sortie adjusted if necessary. Also, a sortie is restricted by the kill-goal for this target type for the particular period, kill goals being cumulative across the periods. The sorties may kill up to the goal, under the goal, but not over the goal.
- (5) The target value credited is for every target of this target type which is dead or in repair over all periods up through the current one, up to the current kill-goal.
- (6) While there are mis-classified targets scheduled for restrike due to bad BDA in the previous planning cycle, and there are sorties left to fly in this day, and the max number of aircraft have not been lost, and the max number of sorties for the day have not been flown yet, these restrikes must be done first before other targets are engaged. Note, an aircraft doing restrikes is still subject to attrition.
- (7) The proportion killed are the dead targets plus targets in repair divided by the total number of targets of the target type starting. It is calculated this way because targets in repair were initially killed to get them into that category, and are recognized as dead until they reappear. This is the quantity which is compared to the kill-goal for the target type. Note: There is no chance for division by zero in the calculation of the proportion since the denominator remains unchanged throughout the entire program run. The quantity in the denominator will always contain the total number of targets of the target type which started the program run, the total amount killed being kept track of by other variables and compared to the original number present to determine the proportion killed. This is convenient calculation wise since kill-goals are cumulative over time. Also, this simulation routine is only run for valid sortie-target engagement combinations as determined by the GAMS model, so the quantity in the denominator will always be a quantity greater than zero.
- (8) Any targets under repair but restrikeable will either become dead forever or under repair and NON-restrikeable. Any targets which are live and restrikeable are targets which have just came out of repair but are immediately restruck, and then become either dead forever or under repair and NON-restrikeable.
- (9) This If..Then represents the case in which all restrikeable targets have been restruck, but the last restrike sortie flown resulted in overkill, i.e., the number of targets killed based on the sortie's EKS was greater than the number of targets left to restrike. Therefore, this overkill is applied to live targets as a regular strike, where it is assumed that live targets existed in the vicinity of the restrikes when they occurred, however, the overkill is applied without the risk of aircraft attrition or weather abort since these were already checked for when the sortie occurred.
- (10) Note that an aircraft which suffers an in-flight weather abort is still subject to attrition, thus explains the position of this conditional check in the overall If..Then structure.
- (11) Each surviving aircraft of the aircraft type gets a chance to fly the sorties per day, in accordance with their sortie rate, until all the days are done or sorties have been flown, or the kill goal is reached. The total sorties which may be flown by all aircraft for each day, however, is capped at a maximum derived from the sorties flown from "tsvtst.dat". When the limit has been reached, we skip to the next day.
- (12) Any dead and restrikeable targets, i.e., mis-BDA'd, will, with probability 1.0, become dead forever when restruck because their "true identity" becomes known upon restrike.
- (13) The current sortie on a restrike mission was not attrited and did not experience a weather abort, so the category of targets it will restrike will be decided randomly with the category having the most number of restrikeable targets having the higher probability of being picked.
- (14) A temporary proportion of the total targets killed is calculated in advance before the current value of NumTgtsKilled is applied to the LiveN type targets so it can be predicted whether or not applying all the kills will cause the kill-goal to be exceeded. If the kill-goal would be exceeded, NumTgtsKilled is successively decremented by 1 until either the predicted value of PropKilled does not exceed the kill-goal, or NumTgtsKilled equals 0.

}

Var

SortiesToFly, NumTgtsKilled, deltaB, deltaNB, deltaBPr, deltaBNPr: Integer;
I, K, deltaPd, delta, deltaNBPr, deltaNBNPr, RTypeTotal: Integer;
SortiesFlownToday, MaxSortiesToday, Start: Integer;
PropKilled, SortiesPerDay, SortiesPerAcftPerDay, WhichRType: Real;

```

TempPropKilled: Real;
AcftKilled, AcftWxAbort, InRegularStrikePhase: Boolean;

Begin {Procedure RunSimulation}
SortiesPerDay := x / NumDays; {** Note 1 **}
SortiesPerAcftPerDay := SortieRate; {** Note 1 **}
AcftKilled := False;
AcftWxAbort := False;
Start := AcftLost + 1;
For I := 1 To NumDays Do
Begin
{
**** Regeneration Phase ****
}
If (RegenN > 0) Then
Begin
delta := Binomial (RegenN, Pc);
LiveN := LiveN + delta;
RegenN := RegenN - delta;
End; {If}
If (RegenR > 0) Then
Begin
delta := Binomial (RegenR, Pc);
LiveR := LiveR + delta;
RegenR := RegenR - delta;
End; {If}
{ PropKilled := DeadN / TotTgts;} {** debug alternative to next line **}
PropKilled := (DeadN + RegenN + DeadR + RegenR) / TotTgts; {** Note 7 **}
MaxSortiesToday := GetInteger (SortiesPerDay);
SortiesFlownToday := 0;
InRegularStrikePhase := False;
For K := Start To NumAcft Do {** Note 11 **}
Begin
SortiesToFly := GetInteger (SortiesPerAcftPerDay);
{
**** Restrike Phase ****
}
NumTgtsKilled := 0;
While ((SortiesToFly > 0) And (AcftLost < MaxLoss) And ((LiveR > 0) Or (DeadR > 0) Or
(RegenR > 0)) And (SortiesFlownToday < MaxSortiesToday) And
(Not InRegularStrikePhase)) Do {** Note 6 **}
Begin
SortiesFlownToday := SortiesFlownToday + 1;
SortiesToFly := SortiesToFly - 1;
TotSortiesFlown := TotSortiesFlown + 1; {** Data purposes only **}
AcftKilled := (UniformProb <= Attr);
If AcftKilled Then
Begin
AcftLost := AcftLost + 1;
SortiesToFly := 0; {** Note 3 **}
End
Else
Begin
AcftWxAbort := (UniformProb > Nabort); {** Note 10 **}
If (Not AcftWxAbort) Then
Begin
NumTgtsKilled := GetInteger (Eks); {** Note 4 **}
While ((NumTgtsKilled > 0) And ((LiveR > 0) Or (DeadR > 0) Or (RegenR > 0))) Do
Begin {** Note 13 **}
RTypeTotal := LiveR + DeadR + RegenR;
WhichRType := RTypeTotal * Random;
If ((WhichRType <= LiveR) And (LiveR > 0)) Then
Begin
If (LiveR <= NumTgtsKilled) Then {** Note 8 **}
Begin
NumTgtsKilled := NumTgtsKilled - LiveR;
delta := Binomial (LiveR, Pr);
LiveR := LiveR - delta;
RegenN := RegenN + delta;
DeadN := DeadN + LiveR;
LiveR := 0;
End
Else
Begin
LiveR := LiveR - NumTgtsKilled;
delta := Binomial (NumTgtsKilled, Pr);
NumTgtsKilled := NumTgtsKilled - delta;
RegenN := RegenN + delta;

```

```

        DeadN := DeadN + NumTgtsKilled;
        NumTgtsKilled := 0;
    End; {If}
End
Else If ((WhichRType <= DeadR) And (DeadR > 0)) Then
Begin
    If (DeadR <= NumTgtsKilled) Then {** Note 12 **}
    Begin
        NumTgtsKilled := NumTgtsKilled - DeadR;
        DeadN := DeadN + DeadR;
        DeadR := 0;
    End
    Else
    Begin
        DeadR := DeadR - NumTgtsKilled;
        DeadN := DeadN + NumTgtsKilled;
        NumTgtsKilled := 0;
    End; {If}
End
Else
Begin
    If (RegenR > 0) Then {** Note 8 **}
    Begin
        If (RegenR <= NumTgtsKilled) Then
        Begin
            NumTgtsKilled := NumTgtsKilled - RegenR;
            delta := Binomial (RegenR, Pr);
            RegenR := RegenR - delta;
            RegenN := RegenN + delta;
            DeadN := DeadN + RegenR;
            RegenR := 0;
        End
        Else
        Begin
            RegenR := RegenR - NumTgtsKilled;
            delta := Binomial (NumTgtsKilled, Pr);
            NumTgtsKilled := NumTgtsKilled - delta;
            RegenN := RegenN + delta;
            DeadN := DeadN + NumTgtsKilled;
            NumTgtsKilled := 0;
        End; {If}
    End; {If}
End; {While}
    End; {If AcftWkAbort}
End; {If AcftKilled}
End; {Main While loop for Restrike Phase}
{
**** Regular Strike and Redistribution of Kills Phase ****
}
{
*****
The If..Then below allows any left over "kills" in the variable NumTgtsKilled which
were not needed to complete all the restrikes from the restrike phase above to be
applied to LiveN targets, if any. Remove the single comment brackets in column 1
ONLY if you want to use this code. ** Note ** This is not consistent with how Time
Strike allocates sorties across time accounting for restrikes, but can be used for
exploratory purposes by the user.
*****
}
    If ((LiveR = 0) And (DeadR = 0) And (RegenR = 0) And (NumTgtsKilled > 0) And
        (LiveN > 0)) Then {** Note 9 **}
    Begin
        TempPropKilled := (DeadN + RegenN + DeadR + RegenR + NumTgtsKilled) / TotTgts;
        While ((TempPropKilled > Goal) And (NumTgtsKilled > 0)) Do {** Note 14 **}
        Begin
            NumTgtsKilled := NumTgtsKilled - 1;
            TempPropKilled := (DeadN + RegenN + DeadR + RegenR + NumTgtsKilled) / TotTgts;
        End; {While}
    End;
    If (NumTgtsKilled > 0) Then
    Begin
        If (LiveN <= NumTgtsKilled) Then
        Begin
            deltaTgtsKilled := deltaTgtsKilled + LiveN;
            deltaB := Binomial (LiveN, B);
            deltaNB := LiveN - deltaB;
            LiveN := 0;
        End
    End

```

```

Else
Begin
  deltaTgtsKilled := deltaTgtsKilled + NumTgtsKilled;
  deltaB := Binomial (NumTgtsKilled, B);
  deltaNB := NumTgtsKilled - deltaB;
  LiveN := LiveN - NumTgtsKilled;
End; {If}
{
  If (deltaB > 0) Then
  Begin
    deltaBPr := Binomial (deltaB, Pr);
    deltaBNPr := deltaB - deltaBPr;
    RegenN := RegenN + deltaBPr;
    DeadN := DeadN + deltaBNPr;
  End; {If}
{
  If (deltaNB > 0) Then
  Begin
    deltaNBPr := Binomial (deltaNB, Pr);
    deltaNBNPr := deltaNB - deltaNBPr;
    RegenR := RegenR + deltaNBPr;
    DeadR := DeadR + deltaNBNPr;
  End; {If}
{
  End; {If}
{
  PropKilled := DeadN / TotTgts; {** debug alternative to next line **}
  PropKilled := (DeadN + RegenN + DeadR + RegenR) / TotTgts; {** Note 7 **}
  While ((SortiesToFly > 0) And (AcftLost < MaxLoss) And (LiveN > 0) And
    (DeadN < TotTgts) And (PropKilled < Goal) And
    (SortiesFlownToday < MaxSortiesToday)) Do {** Note 2 **}
  Begin
    InRegularStrikePhase := True;
    SortiesFlownToday := SortiesFlownToday + 1;
    SortiesToFly := SortiesToFly - 1;
    TotSortiesFlown := TotSortiesFlown + 1; {** Data purposes only **}
    AcftKilled := (UniformProb <= Attr);
    If AcftKilled Then
    Begin
      AcftLost := AcftLost + 1;
      SortiesToFly := 0; {** Note 3 **}
    End
  Else
  Begin
    AcftWxAbort := (UniformProb > Nabort); {** Note 10 **}
    If (Not AcftWxAbort) Then
    Begin
      NumTgtsKilled := GetInteger (Eks); {** Note 4 **}
      TempPropKilled := (DeadN + RegenN + DeadR + RegenR + NumTgtsKilled) / TotTgts;
      While ((TempPropKilled > Goal) And (NumTgtsKilled > 0)) Do {** Note 14 **}
      Begin
        NumTgtsKilled := NumTgtsKilled - 1;
        TempPropKilled := (DeadN + RegenN + DeadR + RegenR + NumTgtsKilled) / TotTgts;
      End; {While}
      If (NumTgtsKilled > 0) Then
      Begin
        If (LiveN <= NumTgtsKilled) Then
        Begin
          deltaTgtsKilled := deltaTgtsKilled + LiveN;
          deltaB := Binomial (LiveN, B);
          deltaNB := LiveN - deltaB;
          LiveN := 0;
        End
      Else
      Begin
        deltaTgtsKilled := deltaTgtsKilled + NumTgtsKilled;
        deltaB := Binomial (NumTgtsKilled, B);
        deltaNB := NumTgtsKilled - deltaB;
        LiveN := LiveN - NumTgtsKilled;
      End; {If}
      If (deltaB > 0) Then
      Begin
        deltaBPr := Binomial (deltaB, Pr);
        deltaBNPr := deltaB - deltaBPr;
        RegenN := RegenN + deltaBPr;
        DeadN := DeadN + deltaBNPr;
      End; {If}
      If (deltaNB > 0) Then
      Begin
        deltaNBPr := Binomial (deltaNB, Pr);
        deltaNBNPr := deltaNB - deltaNBNPr;

```



```

        RegenR := RegenR + deltaNBPr;
        DeadR := DeadR + deltaNBPr;
    End; {If}
End; {If}
{
    PropKilled := DeadN / TotTgts;} {** debug alternative to next line **}
    PropKilled := (DeadN + RegenN + DeadR + RegenR) / TotTgts; {** Note 7 **}
End; {If Not AcftWxAbort}
End; {If AcftKilled}
End; {While loop for Regular Strike and Redistribution of Kills Phase}
If (SortiesFlownToday >= MaxSortiesToday) Then {** Note 11 **}
Begin
    K := NumAcft;
End; {If}
End; {For loop for aircraft}
Start := AcftLost + 1;
End; {For loop for days}
TempTvd := TgtVal * (DeadN + RegenN + DeadR + RegenR); {** Note 5 **}
End; {Procedure RunSimulation}

Procedure GetPdiffValues (NumPeriods, NumTgtTypes, NumTgtClasses: Integer;
    TgtsDeadOrInRepair: TgtKillArray;
    TotTgts: TgtIntTypeArray;
    TgtClassTgt: TgtClassTgtArray;
    TgtsInClass: TgtClassArray;
    Goal: KillGoalArray;
    Var Pdiff: TgtValueArray);
{
    Calculates the pdiff values (proportion of kills below the goal for each target type
    in each period) to be used in the objective function calculation. Note that there
    are no targets present of some target types, so sorties are not flown against them.
    Therefore, division by zero must be accounted for below. Also, if there are targets
    of a target type present, but no kills were made, pdiff is merely set to the kill-goal
    for the target class, i.e., max penalty possible is charged for the case-in-point.

    Pre:  Number of periods, target types, and target classes present. Cumulative number
          of targets of each type dead or in repair each period by period, total targets of
          each type which were present/alive at the start of the program run. Target
          Class-Target correspondence data along with the number of target types in each
          class. Kill goals for each target class in each period.

    Post: Pdiff array = proportion of kills below the goal for each target type in
          each period
}
Var
    Pd, Tgt, Class, I, TotTgtsKilled: Integer;
    PropKilled: Real;

Begin {Procedure GetPdiffValues}
    For Pd := 1 To NumPeriods Do
        Begin
            For Tgt := 1 To NumTgtTypes Do
                Begin
                    Class := GetTargetClass (Tgt, TgtClassTgt, TgtsInClass, NumTgtClasses);
                {
                    *****
                    Total targets killed (actually dead or in repair) are calculated with a For loop here
                    because the cumulatives used in the proportion calculation below must be for only the
                    target type up through the current period of the outer loop.
                    *****
                }
                    TotTgtsKilled := 0;
                    For I := 1 To Pd Do
                        Begin
                            TotTgtsKilled := TotTgtsKilled + TgtsDeadOrInRepair[Tgt,I];
                        End; {For}
                    If (TotTgts[Tgt] <> 0) Then
                        Begin
                            PropKilled := TotTgtsKilled / TotTgts[Tgt];
                        End; {If}
                    If ((TotTgts[Tgt] <> 0) And (PropKilled <= Goal[Class,Pd])) Then
                        Begin
                            Pdiff[Tgt,Pd] := Goal[Class,Pd] - PropKilled;
                        End
                    Else
                        Begin
                            Pdiff[Tgt,Pd] := 0.0;
                        End; {If}
                    End; {For}
                End; {For}
            End; {For}
        End; {For}
    End; {For}
}

```

```

End; {Procedure GetPdiffValues}

Function CalculateObjFcnValue (NumPeriods, NumTgtTypes, NumTgtClasses: Integer;
    TgtClassTgt: TgtClassTgtArray;
    TgtsInClass: TgtClassArray;
    Tvd: Real;
    Ppen: KillGoalArray;
    Pdiff: TgtValueArray;
    TvdWgt, GoalWgt: Real): Real;
{
    Calculates objective function value for maximizing the weighted sum of TVD. The
    Procedure does not actually maximize the value, it calculates the objective function
    value based on TVD gained for a replication of the simulation for which sorties were
    already assigned by the GAMS model, but the results of the sorties were decided
    randomly by this simulation. After the random results of the sorties, kill-goal
    achievement was determined, and then the resultant values for cumulative TVD collected
    and the pdiff values are passed to this Procedure, then the calculation proceeds as
    in the objective function contained in the GAMS model.

    Pre:   Number of periods, target types, and target classes present. Target
           Class-Target correspondence data along with the number of target types in
           each class. The target value destroyed which has been credited for the current
           replication of the simulation. The objective function penalty for not meeting
           the kill-goal for each target class in each period. The objective function
           weights for TVD and goal achievement.
    Post:  CalculateObjFcnValue = Weighted sum of TVD
}
Var
    Pd, Tgt, Class: Integer;
    Penalty: Real;

Begin {Function CalculateObjFcnValue}
    Penalty := 0.0;
    For Pd := 1 To NumPeriods Do
    Begin
        For Tgt := 1 To NumTgtTypes Do
        Begin
            Class := GetTargetClass (Tgt, TgtClassTgt, TgtsInClass, NumTgtClasses);
            Penalty := Penalty + (Ppen[Class,Pd] * Pdiff[Tgt,Pd]);
        End; {For}
    End; {For}
    CalculateObjFcnValue := (TvdWgt * Tvd) + (GoalWgt * Penalty);
End; {Function CalculateObjFcnValue}
{
    *****
    ***** Main Program *****
    *****

    The Main Program is used to assign input and output files, open and close
    input and output files, input all the necessary data from the input files,
    and output the data to the output file. The Main Program is the only part
    of the simulation which reads and outputs data, the other Procedures and
    Functions are the ones which manipulate the data with regards to actually
    performing the simulation. This prevents having to instantiate too many
    array types when passing data from the Main Program to a Procedure or Function,
    which in turn would use up much too much memory since the array types defined
    are quite large!

    The only actual calculations done by the Main Program are for intermediate
    values which must be output for debug purposes or statistical data for the user.
    Comments are imbedded within the Main Program code below which indicates which
    lines should be kept or commented out based on user preference for the output
    generated by the debug lines of code, or to add or subtract output as the user
    sees fit.

    Main Variable definitions:
    -----
    Acft - counter variable for aircraft types.
    AcftLost - array of the cumulative number of aircraft lost for each aircraft type over
              the whole simulation run.
    AcftType - array of aircraft types actually flying sorties.
    Attr - losses per sortie (probability of attrition per sortie in the range [0,1]).
    B - probability of correct BDA for a target (B = BdaProb[Tgt]).
    BdaProb - array of the BDA probabilities for each target.
    BeforeTgtsDeadOrInRepair - the number of targets of the target type dead or in repair
                              just before the simulation routine is run for the next
                              engagement against the target type.
    Ch - a Char type variable used for reading data from "tsvtst.dat" character by

```

character to get the real valued quantities and disregard the commas. Used in conjunction with the variable 'St' as defined below.

Class - counter variable for target classes.

DataValRead - counter variable which keeps track of how many data values have been read in "tsvtst.dat" to determine which simulation variable to store the quantity in.

DeadN - array of the current number of dead targets which ARE NOT restrikeable for each target type (dead forever).

DeadR - array of the current number of dead targets which ARE restrikeable for each target type; the only possibility for these are to become dead forever (see DeadN above) when restruck.

deltaTgtsKilled - difference between total targets of the target type killed after the simulation routine is run and BeforeTgtsKilled, i.e., the number of targets of the target type killed during the last particular run of the simulation routine.

DistBandNum - counter variable for distance band numbers.

Eks - expected kills per sortie for an aircraft type in a particular period.

Goal - array of the proportion of targets in each target class to be killed to achieve the goals in each period.

GoalWgt - objective function weight for goal achievement.

I - counter variable for For loops.

LiveN - array of the current number of live targets which ARE NOT restrikeable for each target type.

LiveR - array of the current number of live targets which ARE restrikeable for each target type; these represent the number of targets which have just come out of repair in the planning cycle but may be restruck and returned to repair (see RegenN below), or become dead forever (see DeadN above).

MaxLoss - array of the max loss allowed of each aircraft type.

Nabort - probability there is no weather abort in flight for a particular aircraft type flying sorties against a particular target type.

NB - probability of incorrect BDA for a target ($NB = 1 - B$).

NPC - probability a target does not regenerate in the next planning cycle ($NPc = 1 - Pc$).

NPr - probability the target is not repairable after a strike ($NPr = 1 - Pr$).

NumAcft - array of the number of aircraft for each aircraft type starting.

NumAcftTypes - number of aircraft types which will fly sorties.

Number - temporary storage location for a real valued number.

NumDays - number of days (planning cycles) per period.

NumPeriods - number of periods the simulation is run over.

NumTgtClasses - number of target classes.

NumTgtsInBand - temporary storage location for the read in number of targets in a particular distance band.

NumTgtTypes - number of target types encountered in "tgtval22.dat".

Pc - probability a target regenerates in the next planning cycle.

Pd - counter variable for periods.

Pdiff - array of proportion of kills below the class goal for a particular target in a particular period.

Ppen - array of objective function penalties for not meeting time-scripted goals for a target class by the end of a particular period (as contained in the Goal array).

Pr - probability the target is repairable after a strike.

RegenN - array of the current number of repairable targets which ARE NOT restrikeable for each target type.

RegenR - array of the current number of repairable targets which ARE restrikeable for each target type; these may regenerate and become LiveR or remain RegenR during the regeneration phase, and then any RegenR may become RegenN or DeadN during the restrike phase.

RepProp - array of the percent of killed targets that regenerate for each target; this is actually treated as a probability of repair in the Simulation.

RepTime - array of the repair times for each target (repair times are in days).

SimInFile(1-4) - files to get input data from.

SimOutFile - file to send output data to.

SortieRate - array of sortie rates (sorties per aircraft per day) for each aircraft type.

St - string type variable used for storing data read from "tsvtst.dat" so the real valued quantities can be picked out of the lines of data since there are imbedded commas as a result of the GAMS model output.

StdDev - the standard deviation of the TVD obj fcn values found over the number of replications performed.

Sumy - sum of objective function values calculated for each replication of simulation.

SumX1 - holds the sum of the $x(i)^2$ terms for the standard deviation calculation.

SumX2 - holds the $(\text{sum of } x(i))^2$ term for the standard deviation calculation.

Temp1 - temporary storage location for a Real value when reading in data.

Temp2 - temporary storage location for a Real value when reading in data.

TempSumTgts - temporary storage location for summing number of targets as the number in particular distance bands is read in.

TempTvd - temporary storage location for TVD value returned by simulation; to be added later to the total.

Tgt - temporary storage for a target type number.

TgtClassTgt - array of target class-target correspondence.
TgtElts - temporary storage location for the data field for target elements in input file "tgt.dat"; not used for anything, just provides a way of skipping unwanted data fields to get to wanted ones.
TgtsDeadOrInRepair - array of cumulative number of targets dead or in repair for each target type BY PERIOD (not to be confused with the cumulative so far) over the whole simulation run; because of the nature of this variable, it may be possible that it has a negative quantity stored at the end of some periods (except the 1st period), however, the sum of these quantities over all periods, or just more than one period, will always be a positive value (or zero).
TgtsInClass - array of the number of targets in each class.
TgtVal - array of target values for all targets over all periods.
TotSortiesFlown - total number of sorties flown over all aircraft and all days for the last simulation run.
TotTgts - array of the total number of each type of target.
Tvd - sum of TVD values returned by simulation for each aircraft type in each period.
TvdWgt - objective function weight for TVD.
v - objective function value for weighted sum of TVD.
x - number of sorties flown for a PARTICULAR aircraft type against a particular target type.

Lines of program code used for debug purposes only (which may also be useful to some users as a normal part of the program) are annotated with ***** debug ***** in comment brackets. Any variables specifically declared and needed for their execution are defined below. These lines of code may be deleted or commented out without disturbing normal program execution, however, they must all either be deleted in their entirety or retained in their entirety because removal of only portions of ***** debug ***** code may disturb other portions retained if they relied on values obtained from the code that was removed.

Debug Variable definitions:

P - counter variable for For loops.
PropKilled - cumulative proportion of targets of the target type killed.
T - counter variable for For loops.

Notes for below:

-
- (1) Have to read the aircraft types as separate data and store them in an array, even though we know the number flying, because the types are not sequential, i.e., certain aircraft types may not be present in the GAMS model data for one reason or another.
 - (2) Only desire certain data from "tsvtst.dat" (see comments at beginning of program), so any undesired data, i.e., not useful to the simulation run, is just dumped off into a temporary storage location, and then just discarded when another data item gets put there. Only certain data within the first 12 quantities in "tsvtst.dat" is required, so rather than wasting time in reading to the end of each line of data, the first 12 items are read, the useful data extracted, and then the rest of the line skipped. The GAMS model will always output data to tsvtst the same way, so this method of reading data will always apply.
 - (3) This loop simply ensures that all the data meant to be output gets written to the output data file. This is required since sometimes long programs will terminate normally, but not all data will be written to the output file if it is sent to one (this problem does not exist if output only goes to the screen, but it's hard to analyze data that way). The loop, therefore, slightly delays program termination so there is time for all data to get written to the output file.
 - (4) The standard deviation will not be calculated or output unless at least 30 replications are performed. This is so it will have at least some meaning when it is calculated.
 - (5) This limits the number of times the obj fcn value is output for each replication to a maximum of 100 times, no matter how many replications are being done. This is done to prevent the output file from growing larger than typical spreadsheet software can handle.

)
Var
SimInFile1, SimInFile2, SimInFile3, SimInFile4, SimOutFile: Text;
Ch: Char;
NumPeriods, NumAcftTypes, NumTgtTypes, NumTgtClasses: Integer;
Tgt, Pd, DistBandNum, Acft, Class, I, TgtElts: Integer;
NumTgtsInBand, TempSumTgts, NumDays, DataValRead: Integer;
TotSortiesFlown, BeforeTgtsDeadOrInRepair, deltaTgtsKilled: Integer;
P, T: Integer; {*** debug ***}

```

Attr, Tvd, TempTvd, v, Sumv, TvdWgt, Temp1, Number: Real;
GoalWgt, Eks, Nabort, Temp2, x, Pr, NPr, Pc, NPC, B, NB: Real;
SumX1, SumX2, StdDev: Real;
PropKilled: Real; {*** debug ***}
TgtVal, Pdiff: TgtValueArray;
TgtsDeadOrInRepair: TgtKillArray;
BdaProb, RepProp: TgtRealTypeArray;
TotTgts, RepTime, LiveN, DeadN, RegenN, LiveR: TgtIntTypeArray;
DeadR, RegenR: TgtIntTypeArray;
AcftType, NumAcft, MaxLoss, AcftLost: AcftTypeArray;
SortieRate: SortieRateArray;
Goal, Ppen: KillGoalArray;
TgtClassTgt: TgtClassTgtArray;
TgtsInClass: TgtClassArray;
St: StringType;

Begin {Main Program}
{
*****
Use only one of the following two seed statements for seeding the random number generator.
Comment the other one out.
*****
}
RandSeed := SEEDFORRANDBANDS; {uses the Const declared seed so random nos. ARE repeated}
{ Randomize;} {takes a seed off the system clock so random nos. ARE NOT repeated}
Assign (SimInFile1, 'sim2in.dat');
Assign (SimInFile2, 'tgtval22.dat');
Assign (SimInFile3, 'tgtat22.dat');
Assign (SimInFile4, 'tsvtst.dat');
Reset (SimInFile1);
Reset (SimInFile2);
Reset (SimInFile3);
Reset (SimInFile4);
Assign (SimOutFile, 'sim2out.csv');
Rewrite (SimOutFile);
{
*****
Read in target value data from "tgtval22.dat".
*****
}
NumTgtTypes := 0;
While (Not (SeekEof(SimInFile2))) Do
Begin
  Read (SimInFile2, Tgt);
  NumTgtTypes := NumTgtTypes + 1;
  For Pd := 1 To MAXNUMPERIODS Do
  Begin
    Read (SimInFile2, TgtVal[Tgt,Pd]);
  End; {For}
  ReadLn (SimInFile2);
End; {While}
{
*****
Read in target data by distance band, total targets, bda probability, repair time,
percent of killed targets that regenerate from "tgtat22.dat".
*****
}
While (Not (SeekEof(SimInFile3))) Do
Begin
  Read (SimInFile3, Tgt);
  TempSumTgts := 0;
  For DistBandNum := 1 To MAXNUMDISTBANDS Do
  Begin
    Read (SimInFile3, NumTgtsInBand);
    TempSumTgts := TempSumTgts + NumTgtsInBand;
  End; {For}
  TotTgts[Tgt] := TempSumTgts;
  Read (SimInFile3, TgtElts);
  Read (SimInFile3, BdaProb[Tgt]);
  Read (SimInFile3, RepTime[Tgt]);
  ReadLn (SimInFile3, RepProp[Tgt]);
End; {While}
{
*****
Read in simulation specific data from "sim2in.dat".
*****
}
ReadLn (SimInFile1, TvdWgt, GoalWgt);

```

```

ReadLn (SimInFile1, NumPeriods, NumDays);
ReadLn (SimInFile1, NumAcftTypes);
For Acft := 1 To NumAcftTypes Do
Begin
  ReadLn (SimInFile1, AcftType[Acft], NumAcft[AcftType[Acft]], MaxLoss[AcftType[Acft]],
    SortieRate[AcftType[Acft]]); {** Note 1 **}
End; {For}
ReadLn (SimInFile1, NumTgtClasses);
For Class := 1 To NumTgtClasses Do
Begin
  Read (SimInFile1, Temp1);
  For Pd := 1 To NumPeriods Do
  Begin
    Read (SimInFile1, Goal[Class,Pd]);
  End; {For}
  ReadLn (SimInFile1);
End; {For}
For Class := 1 To NumTgtClasses Do
Begin
  Read (SimInFile1, Temp1);
  For Pd := 1 To NumPeriods Do
  Begin
    Read (SimInFile1, Ppen[Class,Pd]);
  End; {For}
  ReadLn (SimInFile1);
End; {For}
For Class := 1 To NumTgtClasses Do
Begin
  Read (SimInFile1, Temp1);
  Tgt := 0;
  While (Not (SeekEoln(SimInFile1))) Do
  Begin
    Tgt := Tgt + 1;
    Read (SimInFile1, TgtClassTgt[Class,Tgt]);
  End; {While}
  TgtsInClass[Class] := Tgt;
  ReadLn (SimInFile1);
End; {For}
}
*****
Start replication loop; begin reading data from "tsvtst.dat" and run simulation.
*****
}
Sumv := 0.0;
SumX1 := 0.0;
SumX2 := 0.0;
{
*****
If you do not want the objective function value for each replication output to the
output file in tabular format, comment out the next line. If you want it in tabular
format, don't comment it out, but then you must also choose the 2nd method of the two
presented further below of outputting the replication and obj fcn value (see imbedded
comments below).
*****
}
{ WriteLn (SimOutFile,"Replication",',','Obj Fcn Val');}
For I := 1 To NUMBERREPS Do
Begin
{
*****
Next If..Then is *** debug ***. These are the column headings for the major output
statistics output showing the results of a particular replication simulation run.
*****
}
  If ((I = REPOUT1) Or (I = REPOUT2) Or (I = REPOUT3)) Then
  Begin
    If (I <> 1) Then
    Begin
      WriteLn (SimOutFile);
    End;
    WriteLn (SimOutFile,"Replication",',','Period',',','Aircraft',',',
      'Target',',','Class',',','EKS',',','Nabort',',','Attr',',',
      'Sorties',',','TotSortiesFlown',',','SortieRate',',','MaxLoss',',',
      'AcftLost',',','Kill Goal',',','PropKilled',',','Pr',',',
      'Pc',',','B',',','LiveN',',','DeadN',',','RegenN',',',
      'Liver',',','DeadR',',','RegenR',',','TotTgts',',',
      'deltaTgtsKilled',',','TgtsDeadOrInRepair',',','TgtVal',',',
      'deltaTVD',',','TVD');
  End;
}

```

```

End; {If}
Reset (SimInFile4);
St := '';
Tvd := 0.0;
For Acft := 1 To NumAcftTypes Do
Begin
  AcftLost[AcftType[Acft]] := 0;
End; {For}
For Tgt := 1 To NumTgtTypes Do
Begin
  LiveN[Tgt] := TotTgts[Tgt];
  DeadN[Tgt] := 0;
  RegenN[Tgt] := 0;
  LiveR[Tgt] := 0;
  DeadR[Tgt] := 0;
  RegenR[Tgt] := 0;
  For Pd := 1 To NumPeriods Do
  Begin
    Pdiff[Tgt,Pd] := 1.0;
    TgtsDeadOrInRepair[Tgt,Pd] := 0;
  End; {For}
End; {For}
While (Not (SeekEof(SimInFile4))) Do
Begin
  For DataValRead := 1 To 12 Do {** Note 2 **}
  Begin
    Read (SimInFile4, Ch);
    While (Ch <> ',') Do
    Begin
      St := St + Ch;
      Read (SimInFile4, Ch);
    End; {While}
    GetRealNumber (St, Number);
    Case DataValRead Of {** Note 2 **}
      2: Pd := Trunc(Number);
      4: Acft := Trunc(Number);
      6: Tgt := Trunc(Number);
      9: Eks := Number;
      10: Nabort := Number;
      11: Attr := Number;
      12: x := Number
    Else
      Temp1 := Number; {** Note 2 **}
    End; {Case}
    St := '';
  End; {For}
  Class := GetTargetClass (Tgt, TgtClassTgt, TgtsInClass, NumTgtClasses);
  TempTvd := 0.0;
  BeforeTgtsDeadOrInRepair := DeadN[Tgt]; {** debug alternative to next line **}
  BeforeTgtsDeadOrInRepair := DeadN[Tgt] + RegenN[Tgt] + DeadR[Tgt] + RegenR[Tgt];
  deltaTgtsKilled := 0;
  TotSortiesFlown := 0; {** Data purposes only **}
  Pr := RepProp[Tgt];
  NPr := 1 - Pr;
  Pc := 1 - Exp(-(1 / RepTime[Tgt]));
  NPc := 1 - Pc;
{
  *****
  If you want to test the case of perfect BDA, i.e., no restrikes will occur, then remove
  comments from next line and comment out the line after next. If you choose perfect BDA
  for the B probability assignment here, you must also choose the same option for the NB
  probability immediately following the B probability assignment code lines.
  *****
}
{
  B := 1.0; {** Perfect BDA **}
  B := BdaProb[Tgt]; {** Imperfect BDA **}
}
{
  *****
  If you chose to test the case of perfect BDA, then remove comments from next line and
  comment out the line after next.
  *****
}
{
  NB := 0.0; {** Perfect BDA **}
  NB := 1 - B; {** Imperfect BDA **}
  RunSimulation (NumDays, NumAcft[Acft], MaxLoss[Acft], SortieRate[Acft],
  AcftLost[Acft], TotTgts[Tgt], TgtVal[Tgt,Pd], Goal[Class,Pd],
  Eks, Nabort, Attr, x, TotSortiesFlown, Pr, NPr, Pc, NPc, B, NB,
  LiveN[Tgt], DeadN[Tgt], RegenN[Tgt], LiveR[Tgt], DeadR[Tgt],

```

```

                RegenR[Tgt], deltaTgtsKilled, TempTvd);
Tvd := Tvd + TempTvd;
TgtsDeadOrInRepair[Tgt,Pd] := TgtsDeadOrInRepair[Tgt,Pd] + ((DeadN[Tgt] +
                RegenN[Tgt] + DeadR[Tgt] + RegenR[Tgt]) -
                BeforeTgtsDeadOrInRepair);
{
    PropKilled := DeadN[Tgt] / TotTgts[Tgt];} {** debug alternative to next line **}
PropKilled := (DeadN[Tgt] + RegenN[Tgt] + DeadR[Tgt] + RegenR[Tgt]) /
                TotTgts[Tgt]; {*** debug ***}
{
    *****
Next If..Then is *** debug ***. These are the major output statistics showing the
results of a particular replication simulation run.
    *****
}
    If ((I = REPOUT1) Or (I = REPOUT2) Or (I = REPOUT3)) Then
    Begin
        WriteLn (SimOutFile,I,',',Pd,',',Acft,',',Tgt,',',Class,',',Eks:9:4,',',
                Nabort:9:4,',',Attr:9:4,',',x:9:4,',',TotSortiesFlowm,',',
                SortieRate[Acft]:9:4,',',MaxLoss[Acft],',',AcftLost[Acft],',',
                Goal[Class,Pd]:9:4,',',PropKilled:9:4,',',Pr:9:4,',',Pc:9:4,',',
                B:9:4,',',LiveN[Tgt],',',DeadN[Tgt],',',RegenN[Tgt],',',
                LiveR[Tgt],',',DeadR[Tgt],',',RegenR[Tgt],',',TotTgts[Tgt],',',
                deltaTgtsKilled,',',TgtsDeadOrInRepair[Tgt,Pd],',',TgtVal[Tgt,Pd]:9:4,',',
                TempTVD:9:4,',',Tvd:9:4);
    End; {If}
    St := '';
    ReadLn (SimInFile4);
    End; {While (Not (SeekEof(SimInFile4)))}
{
    *****
Next If..Then is *** debug ***. This just outputs a blank line to separate blocks of
output data in the output file.
    *****
}
    If ((I = REPOUT1) Or (I = REPOUT2) Or (I = REPOUT3)) Then
    Begin
        WriteLn (SimOutFile);
    End; {If}
    GetPdiffValues (NumPeriods, NumTgtTypes, NumTgtClasses, TgtsDeadOrInRepair, TotTgts,
                TgtClassTgt, TgtsInClass, Goal, Pdiff);
{
    *****
Next If..Then is *** debug ***. This outputs the pdiff values for each target type by
period.
    *****
}
    If ((I = REPOUT1) Or (I = REPOUT2) Or (I = REPOUT3)) Then
    Begin
        WriteLn (SimOutFile,"Period",',',,"Target",',',,"Pdiff");
        For P := 1 To NumPeriods Do
            Begin
                For T := 1 To NumTgtTypes Do
                    Begin
                        WriteLn (SimOutFile,P,',',T,',',Pdiff[T,P]:9:4);
                    End;
                End;
            WriteLn (SimOutFile);
        End; {If}
        Temp2 := CalculateObjFcnValue (NumPeriods, NumTgtTypes, NumTgtClasses, TgtClassTgt,
                TgtsInClass, Tvd, Ppen, Pdiff, TvdWgt, GoalWgt);

        Sumv := Sumv + Temp2;
        SumX1 := SumX1 + Sqr(Temp2);
        SumX2 := SumX2 + Temp2;
{
    *****
If you do not want the objective function value for each replication output to the
output file, comment out the 2 WriteLn statements in the next If..Then. Otherwise,
choose one based on your preferred method of having it output, but only choose the 2nd
one if you chose to leave in the tabular format for the output from above.
    *****
}
    If (I <= 100) Then {** Note 5 **}
    Begin
        WriteLn (SimOutFile,"Replication:",',',I,',',,"ObjFcnVal:",',',Temp2:15:4);
{
        WriteLn (SimOutFile,I,',',Temp2:15:4);}
    End; {If}
End; {For replication loop}
v := Sumv / NUMBERREPS;

```



```

If (NUMBERREPS >= 30) Then (** Note 4 **)
Begin
  StdDev := Sqrt((SumX1 - (Sqr(SumX2) / NUMBERREPS)) / (NUMBERREPS - 1));
End; {If}
WriteLn (SimOutFile); (** debug **)
WriteLn (SimOutFile); (** debug **)
{
  *****
  Pick one of the next two statements to output the number of replications. Pick the
  first one if you want to output to the screen only, pick the second one if you want to
  output to the output file, then comment the other one out.
  *****
}
{ WriteLn ('Number of Replications: ',NUMBERREPS);}
WriteLn (SimOutFile,"Number",' ','of',' ','Replications: ',' ',NUMBERREPS);
WriteLn (SimOutFile);
{
  *****
  Pick one of the next two statements to output the objective function value. Pick the
  first one if you want to output to the screen only, pick the second one if you want to
  output to the output file, then comment the other one out.
  *****
}
{ WriteLn ('Objective Function Value (v: weighted sum of TVD): ',v:15:4);}
WriteLn (SimOutFile,"Objective",' ','Function",' ','Value",' ','(v :',' ',' ',
  "weighted",' ','sum of",' ','TVD):",' ',' ',v:15:4);
WriteLn (SimOutFile);
{
  *****
  Pick one of the two WriteLn statements in the following If..Then to output the standard
  deviation. Pick the first one if you want to output to the screen only, pick the second
  one if you want to output to the output file, then comment the other one out.
  *****
}
If (NUMBERREPS >= 30) Then (** Note 4 **)
Begin
  WriteLn ('Standard Deviation: ',StdDev:15:4);}
  WriteLn (SimOutFile,"Standard",' ','Deviation:"',' ',StdDev:15:4);
End; {If}
{
  *****
  If you are NOT sending output to the output file, comment out the following For loop.
  *****
}
For I := 1 To 50 Do
Begin
  WriteLn (SimOutFile);
End; {For} (** Note 3 **)
Close (SimInFile1);
Close (SimInFile2);
Close (SimInFile3);
Close (SimInFile4);
Close (SimOutFile);
End. {Main Program}

```

The following is the data file "sim2in.dat" used by SimStrike. The format must remain exactly as shown below, however, the data may change of course, but must be representative of the items explained below. Note: Do not include the comments in "{}" shown in the listing below. These are merely for explanatory purposes within this Appendix only. They do not exist in the actual file, nor should they be included.

```

9.0  -1.0 {TVDWGT, GOALWGT}
5      12 {no. of periods, no. of days in each period}
9      {no. of aircraft types which will actually be flying sorties}
{Next 9 lines (because there are 9 aircraft types actually flying
sorties in this case) are in the following data format:
aircraft type, no. of aircraft type starting, max. loss allowed for this
aircraft type, sortie rate (sorties per aircraft per day) for this
aircraft type}
1      96      8      1.48
2      72      4      .6
3      16      1      .6
4      12      1      .6
6      36      2      .9
7      102     8      1.09
8      129     7      1.29
9      165     9      1.39
10     72      4      .48
10     {number of target classes}
{Next 10 lines (10 target classes) are in the following data format:
target class, proportion of targets in target class to be killed to
achieve the kill goal for each time period up to the number of periods
over which SimStrike is to be run}
1      .4      .55     .7      1.0     1.0
2      .35     .6      .9      1.0     1.0
3      1.0     1.0     1.0     1.0     1.0
4      .75     .95     1.0     1.0     1.0
5      .8      .95     1.0     1.0     1.0
6      1.0     1.0     1.0     1.0     1.0
7      .2      .4      .75     1.0     1.0
8      .25     .35     .5      1.0     1.0
9      1.0     1.0     1.0     1.0     1.0
10     .4      .4      .8      .8      1.0
{Next 10 lines (10 target classes) are in the following data format:
target class, objective function penalty for not meeting the time-
scripted kill goal for the target class by the end of the period for
each period}
1      64      64      16      16      64
2      16      16      64      64      16
3      1      1      1      1      1
4      16      16      16      16      16
5      64      64      64      64      64
6      64      64      16      16      4
7      4      4      64      64      64
8      1      1      1      1      16
9      4      4      64      64      16
10     16      16      16      16      16
{Next 10 lines (10 target classes) are in the following data format:

```

target class, listing of targets which belong to the target class (this is known as target class-target correspondence in TIME STRIKE)}

```
1 8 16 17 19 37 38 61 65 66 69 70 76
2 14 15 18 20 21 25 29 30 32 36 39 41 42 46 60 62 63 64 67 68 74 75
3 55 56 57 58
4 3 4 5 6 7 10 11 12 28 47 48 71 72
5 49 50 51 77 78
6 22 26 54
7 23 24
8 9 13 27 31 40 43 44 45 59 73
9 33 34 35
10 1 2 52 53 79 80 81 82 83 84 85 86 87
```

This is the end of the file "sim2in.dat". Do not leave any blank lines at the end of the file, except for one position for the end-of-file marker. As long as the copy of this file provided with the original form of SimStrike is used as a template for changing the data contained in it, errors can be prevented.

APPENDIX B: SAMPLE OUTPUT

The sample output on the following pages is the result of a run of SimStrike using the model data set provided by Maj. Kirk Yost, USAF, as mentioned in Chapter II. It is the exact output produced by the program listing in Appendix A. One-thousand replications were done, with only one-hundred objective function results shown so the output file would be limited in size so as not to present problems when opening in the editor of a spreadsheet application (this is also eluded to in the comments within the program listing). The final results are on the last page of the sample output.

A detailed replication instance is shown for replication number one. The column headings for the most part are the same name as the variable they represent in SimStrike. In any event, the headings are self-explanatory as to what statistic they represent. These headings can be more easily understood, however, if the corresponding variable definitions in the comments just prior to the main program part of SimStrike are referred to.

Also, as mentioned in Appendix A, this output can be modified to take different forms based on the preference of the user. Please refer to the comments imbedded in the main program part of the program listing to change the output format as desired.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
189	1	4	7	70	1	8.84	0.9502	0.0018	4.85	5	1.09	8	1	1	0.5181
190	1	4	7	74	2	6.32	0.9502	0.0015	0.791	1	1.09	8	1	1	0.619
191	1	4	7	78	1	3.2	0.9502	0.0018	0.438	2	1.09	8	1	1	0.3333
192	1	4	7	79	10	2.71	0.9502	0.0016	0.692	1	1.09	8	1	0.8	0.7059
193	1	4	8	1	10	3.85	1	0.0007	8.148	0	1.29	7	7	0.8	0.7528
194	1	4	8	18	2	0.68	1	0.0007	30.401	0	1.29	7	7	1	0.5341
195	1	4	8	22	6	1.65	1	0.0005	2.045	0	1.29	7	7	1	1
196	1	4	8	51	5	1.75	1	0.0007	0.078	0	1.29	7	7	1	0.8
197	1	4	8	73	8	0.95	1	0.0008	78.617	0	1.29	7	7	1	0.2255
198	1	4	8	17	1	8.85	0.9502	0.0008	11.721	0	1.29	7	7	1	0.3735
199	1	4	8	41	2	3.06	1	0.0008	22.271	0	1.29	7	7	1	0.8537
200	1	4	8	53	10	3.57	1	0.0008	1.782	0	1.29	7	7	0.8	0.3188
201	1	4	8	62	2	3.61	1	0.0008	1.909	0	1.29	7	7	1	0.25
202	1	4	9	43	8	1.29	1	0.0008	0.971	1	1.39	9	1	1	0.5
203	1	4	9	68	1	5.93	0.9502	0.0008	8.936	8	1.39	9	1	1	0.5981
204	1	4	10	15	2	0.71	1	0.0008	22.242	20	0.48	4	2	1	0.9787
205	1	4	10	38	1	4.93	1	0.0008	6.536	5	0.48	4	2	1	0.6196
206	1	5	1	19	1	1.68	0.7665	0.0021	73.689	69	1.48	8	0	1	1
207	1	5	1	81	1	1.88	0.7665	0.0021	1.286	1	1.48	8	0	1	0.8519
208	1	5	2	5	4	7.83	1	0.0016	0	0	0.6	4	0	1	1
209	1	5	3	23	7	0.98	1	0	1.313	1	0.6	1	0	1	1
210	1	5	3	25	2	0.38	1	0	15.178	16	0.6	1	0	1	0.7308
211	1	5	3	45	8	0.43	1	0	52.749	52	0.6	1	0	1	0.8997
212	1	5	3	41	2	0.68	1	0	8.892	10	0.6	1	0	1	0.8556
213	1	5	3	42	2	0.78	1	0	6.149	4	0.6	1	0	1	1
214	1	5	3	78	5	0.98	1	0	0.513	0	0.6	1	0	1	1
215	1	5	3	75	2	2.06	1	0	0.101	1	0.6	1	0	1	0.4288
216	1	5	3	73	8	7.57	1	0	0.584	0	0.6	1	0	1	0.2255
217	1	5	4	52	10	5.47	0.8185	0.0023	0.133	0	0.6	1	0	1	0
218	1	5	4	7	4	9.39	1	0	0	0	0.6	1	0	1	0
219	1	5	6	49	5	1.71	1	0.0001	0.083	0	0.9	2	0	1	0.6
220	1	5	6	50	5	1.69	1	0.0001	0.14	0	0.9	2	0	1	0.7279
221	1	5	6	77	5	1.81	1	0.0001	2.285	2	0.9	2	0	1	0.3784
222	1	5	7	37	1	1.48	1	0.0017	5.149	4	1.09	8	1	1	0.9694
223	1	5	7	28	6	4.01	1	0.0012	1.729	2	1.09	8	1	1	0.7257
224	1	5	7	35	9	1.79	0.8596	0.0014	0.458	0	1.09	8	1	1	0.902
225	1	5	7	40	8	1.79	0.8596	0.0018	0.428	0	1.09	8	1	1	0.5238
226	1	5	7	46	2	1.79	0.8596	0.0015	2.53	5	1.09	8	1	1	0.8537
227	1	5	7	12	4	1.71	0.8596	0.0011	3.027	1	1.09	8	1	1	0.8171
228	1	5	7	33	9	0.9	0.8596	0.0022	0.896	0	1.09	8	1	1	1
229	1	5	7	54	6	0.9	0.8596	0.0012	0.059	0	1.09	8	1	1	1
230	1	5	7	38	1	6.31	1	0.0003	0.734	1	1.09	8	1	1	0.8311
231	1	5	7	60	2	6.83	0.9502	0.0018	0.637	0	1.09	8	1	1	0.7037
232	1	5	7	64	2	9.5	0.9502	0.0019	0.172	0	1.09	8	1	1	0.697
233	1	5	7	66	1	8.84	0.9502	0.0018	1.090	0	1.09	8	1	1	0.5701
234	1	5	7	68	2	9.5	0.9502	0.0019	0.139	0	1.09	8	1	1	0.5385
235	1	5	7	70	1	8.84	0.9502	0.0018	0.85	1	1.09	8	1	1	0.5181
236	1	5	7	74	2	6.32	0.9502	0.0015	0.108	0	1.09	8	1	1	0.619
237	1	5	7	79	10	2.71	0.9502	0.0016	2.085	4	1.09	8	1	1	0.9412
238	1	5	8	1	10	3.85	1	0.0007	26.394	0	1.29	7	7	1	0.7528
239	1	5	8	43	8	1.29	1	0.0008	0.111	0	1.29	7	7	1	0.5
240	1	5	8	51	5	1.75	1	0.0007	0.013	0	1.29	7	7	1	0.8
241	1	5	8	73	8	0.95	1	0.0008	21.788	0	1.29	7	7	1	0.2255
242	1	5	8	53	10	3.57	1	0.0008	5.949	0	1.29	7	7	1	0.3188
243	1	5	9	22	6	1.65	1	0.0005	0.475	0	1.39	9	1	1	1
244	1	5	9	17	1	8.85	0.9502	0.0008	2.91	2	1.39	9	1	1	0.3978
245	1	5	9	76	1	8.85	0.9502	0.0008	0.018	0	1.39	9	1	1	0.3333
246	1	5	9	62	2	3.61	1	0.0006	0.199	1	1.39	9	1	1	0.2909
247															
248	Period	Target	Pdfff												
249	1	1	0.0139												
250	1	2	0.4												
251	1	3	0.75												
252	1	4	0.8071												
253	1	5	0.05												
254	1	6	0.05												
255	1	7	0.75												
256	1	8	0.0697												
257	1	9	0.25												
258	1	10	0.75												
259	1	11	0.75												
260	1	12	0.1159												
261	1	13	0.25												
262	1	14	0												
263	1	15	0.084												
264	1	16	0												
265	1	17	0.1018												
266	1	18	0.0091												
267	1	19	0.4												
268	1	20	0.1417												
269	1	21	0.35												
270	1	22	0.1275												
271	1	23	0.0077												
272	1	24	0.2												
273	1	25	0.0038												
274	1	26	0.4071												
275	1	27	0.0068												
276	1	28	0.6984												
277	1	29	0												
278	1	30	0.0537												
279	1	31	0.25												
280	1	32	0.35												
281	1	33	0.0727												
282	1	34	0.381												

	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
188	0.25	0.1331	0.7	40	38	0	0	5	0	83	17	16	50.9	2188.7	258848.1
190	0.25	0.6321	0.7	8	13	0	0	0	0	21	0	0	27.1	352.3	257300.4
191	0.25	0.6321	0.8	2	1	0	0	0	0	3	3	1	27.1	27.1	257327.5
192	0.25	0.011	0.6	5	10	2	0	0	0	17	0	0	43.8	525.8	257853.1
193	0.125	0.1331	0.6	89	263	0	0	8	0	360	0	-6	20.4	5528.4	263381.5
194	0.5	0.1331	0.6	41	43	4	0	0	0	88	0	-11	38.6	1814.2	265195.7
195	0.5	0.6321	0.7	0	149	0	0	0	0	149	0	0	26.3	3918.7	269114.4
196	0.25	0.2835	0.6	1	4	0	0	0	0	5	0	0	32.7	130.8	269245.2
197	0.5	0.2835	0.8	79	23	0	0	0	0	102	0	4	50.1	1152.3	270387.6
198	0.25	0.1331	0.6	206	124	0	0	0	0	332	0	18	33.3	4129.2	274526.7
199	0.25	0.0028	0.9	80	371	85	0	0	1	547	0	-3	13.2	6184.4	290891.1
200	0.5	0.0019	0.6	48	13	8	1	1	0	69	0	-2	16.3	358.6	281049.7
201	0.25	0.0019	0.8	48	15	1	0	0	0	84	0	0	50.9	814.4	281864.1
202	0.25	0.0185	0.8	1	0	0	0	1	0	2	1	1	5.8	5.8	281869.7
203	0.25	0.1331	0.7	43	59	3	0	2	0	107	23	19	43.9	2809.6	284679.3
204	0.25	0.1331	0.8	2	91	1	0	0	0	94	13	8	36.2	3514.4	288183.7
205	0	0.0019	0.9	94	427	0	0	0	0	521	14	113	33.3	14219.1	302412.8
206	0.5	0.1331	0.8	0	470	36	0	0	0	506	76	0	24.3	12295.8	314708.6
207	0	0.0019	0.6	4	23	0	0	0	0	27	0	0	32.4	745.2	315453.8
208	0	0.0019	0.8	0	8	0	0	1	0	10	0	0	17.8	178	315631.8
209	0.5	0.0028	0.9	0	25	27	0	0	0	52	0	0	50.1	2605.2	318237
210	0.25	0.0185	0.9	42	108	8	0	0	0	158	5	1	27	3078	321315
211	0.5	0.2835	0.8	6	37	2	0	0	0	45	15	6	39.1	1524.9	322639.9
212	0.25	0.0028	0.9	79	374	94	0	0	0	547	3	1	31	14508	337347.9
213	0.25	0.0185	0.8	0	20	4	0	0	0	24	3	3	31	744	338091.9
214	0.25	0.011	0.6	0	22	2	0	0	0	24	0	0	38.9	885.6	338977.5
215	0.25	0.6321	0.7	4	2	0	0	1	0	7	2	2	29.6	88.8	339066.3
216	0.5	0.2835	0.8	79	23	0	0	0	0	102	0	0	63.7	1465.1	340531.4
217	0	0.0019	0.6	2	0	0	0	0	0	2	0	0	45	0	340531.4
218	0	0.0019	0.8	9	0	0	0	0	0	9	0	0	12.2	0	340531.4
219	0.5	0.2835	0.6	1	3	0	0	1	0	5	0	0	22	68	340619.4
220	0.5	0.2835	0.6	3	7	0	0	1	0	11	0	0	22	178	340785.4
221	0.25	0.0328	0.6	92	51	4	0	1	0	148	4	4	28.8	1612.8	342408.2
222	0.5	0.0019	0.7	5	86	58	0	0	0	149	2	1	37.7	5428.8	347837
223	0.5	0.0652	0.8	31	79	3	0	0	0	113	4	1	4.4	360.8	348197.8
224	0.25	0.0055	0.8	5	35	11	0	0	0	51	0	0	41.9	1927.4	350125.2
225	0.5	0.0028	0.9	10	8	3	0	0	0	21	0	-1	38.4	422.4	350547.6
226	0.5	0.2835	0.8	6	34	0	0	1	0	41	4	1	42	1470	352017.6
227	0.5	0.011	0.9	14	48	19	1	0	0	82	0	-5	2.1	140.7	352158.3
228	0.5	0.0019	0.6	0	30	25	0	0	0	55	0	0	48.8	2739	354887.3
229	0	0.0019	0.6	0	11	0	0	0	0	11	0	0	48.5	533.5	355430.8
230	0	0.0019	0.6	88	432	0	0	1	0	521	6	6	24.3	10521.9	365952.7
231	0.5	0.1331	0.8	8	18	1	2	0	0	27	0	-4	34.9	663.1	366615.8
232	0.25	0.1331	0.7	8	22	0	2	1	0	33	0	0	36.4	837.2	367453
233	0.25	0.1331	0.7	46	59	0	0	2	0	107	0	-3	29.6	1805.6	369258.6
234	0.25	0.1331	0.7	12	11	0	0	3	0	28	0	-2	39.1	547.4	369806
235	0.25	0.1331	0.7	40	43	0	0	0	0	83	0	0	39.1	1681.3	371487.3
236	0.25	0.6321	0.7	8	13	0	0	0	0	21	0	0	16	208	371685.3
237	0.25	0.011	0.6	1	13	3	0	0	0	17	4	4	45	720	372415.3
238	0.125	0.1331	0.6	89	263	0	0	8	0	360	0	0	12	3252	375667.3
239	0.25	0.0185	0.8	1	0	0	0	1	0	2	0	0	22.7	22.7	375690
240	0.25	0.2835	0.6	1	4	0	0	0	0	5	0	0	38.9	147.8	375837.6
241	0.5	0.2835	0.8	79	23	0	0	0	0	102	0	0	63.7	1465.1	377302.7
242	0.5	0.0019	0.6	48	13	8	1	1	0	69	0	0	26.2	576.4	377879.1
243	0.5	0.6321	0.7	0	149	0	0	0	0	149	0	0	24.6	3865.4	381544.5
244	0.25	0.1331	0.8	200	131	1	0	0	0	332	9	8	24.3	3207.8	384752.1
245	0.25	0.6321	0.8	2	1	0	0	0	0	3	0	0	16	16	384788.1
246	0.25	0.0019	0.8	45	18	1	0	0	0	64	3	3	63.2	1200.8	385668.9
247															
248															
249															
250															
251															
252															
253															
254															
255															
256															
257															
258															
259															
260															
261															
262															
263															
264															
265															
266															
267															
268															
269															
270															
271															
272															
273															
274															
275															
276															
277															
278															
279															
280															
281															
282															

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
283	1	35	0												
284	1	36	0.0061												
285	1	37	0.004												
286	1	38	0.0679												
287	1	39	0.0022												
288	1	40	0.0565												
289	1	41	0.01												
290	1	42	0.0167												
291	1	43	0.25												
292	1	44	0.25												
293	1	45	0.0058												
294	1	46	0.0329												
295	1	47	0.75												
296	1	48	0.75												
297	1	49	0.8												
298	1	50	0.5273												
299	1	51	0												
300	1	52	0.4												
301	1	53	0.0522												
302	1	54	1												
303	1	55	1												
304	1	56	1												
305	1	57	1												
306	1	58	0												
307	1	59	0.0278												
308	1	60	0.35												
309	1	61	0.4												
310	1	62	0.1												
311	1	63	0												
312	1	64	0.0167												
313	1	65	0												
314	1	66	0.129												
315	1	67	0												
316	1	68	0.0423												
317	1	69	0												
318	1	70	0.0145												
319	1	71	0.75												
320	1	72	0.75												
321	1	73	0.0245												
322	1	74	0.35												
323	1	75	0.2071												
324	1	76	0.4												
325	1	77	0.773												
326	1	78	0.2583												
327	1	79	0.0471												
328	2	1	0												
329	2	2	0.4												
330	2	3	0.774												
331	2	4	0.8071												
332	2	5	0.25												
333	2	6	0.15												
334	2	7	0.95												
335	2	8	0.2167												
336	2	9	0.35												
337	2	10	0.95												
338	2	11	0.95												
339	2	12	0.072												
340	2	13	0.35												
341	2	14	0												
342	2	15	0.0681												
343	2	16	0												
344	2	17	0.294												
345	2	18	0.35												
346	2	19	0.0381												
347	2	20	0.1833												
348	2	21	0.6												
349	2	22	0												
350	2	23	0.0154												
351	2	24	0.4												
352	2	25	0.1												
353	2	26	0.2832												
354	2	27	0.1068												
355	2	28	0.8884												
356	2	29	0												
357	2	30	0.3037												
358	2	31	0.35												
359	2	32	0.6												
360	2	33	0												
361	2	34	0.381												
362	2	35	0												
363	2	36	0.0013												
364	2	37	0.0131												
365	2	38	0.0989												
366	2	39	0.2341												
367	2	40	0.0167												
368	2	41	0.0113												
369	2	42	0.0167												
370	2	43	0.35												
371	2	44	0.35												
372	2	45	0.0167												
373	2	46	0.1122												
374	2	47	0.95												
375	2	48	0.95												
376	2	49	0.55												

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
377	2	50	0.4045												
378	2	51	0.15												
379	2	52	0.4												
380	2	53	0.0522												
381	2	54	0												
382	2	55	1												
383	2	56	1												
384	2	57	1												
385	2	58	0												
386	2	59	0.1278												
387	2	60	0.2296												
388	2	61	0.0315												
389	2	62	0.35												
390	2	63	0												
391	2	64	0.0242												
392	2	65	0												
393	2	66	0.2228												
394	2	67	0												
395	2	68	0.3308												
396	2	69	0												
397	2	70	0.2247												
398	2	71	0.95												
399	2	72	0.95												
400	2	73	0.1837												
401	2	74	0.0762												
402	2	75	0.4571												
403	2	76	0.55												
404	2	77	0.673												
405	2	78	0.075												
406	2	79	0.0471												
407	3	1	0.0306												
408	3	2	0.8												
409	3	3	0.824												
410	3	4	0.8571												
411	3	5	0												
412	3	6	0.2												
413	3	7	1												
414	3	8	0.3887												
415	3	9	0.5												
416	3	10	1												
417	3	11	1												
418	3	12	0.0976												
419	3	13	0.5												
420	3	14	0												
421	3	15	0.0064												
422	3	16	0												
423	3	17	0.3807												
424	3	18	0.2406												
425	3	19	0.0083												
426	3	20	0.1917												
427	3	21	0.9												
428	3	22	0												
429	3	23	0.0385												
430	3	24	0.75												
431	3	25	0.3038												
432	3	26	0.2655												
433	3	27	0.2568												
434	3	28	0.9384												
435	3	29	0												
436	3	30	0.6037												
437	3	31	0.5												
438	3	32	0.9												
439	3	33	0												
440	3	34	0.381												
441	3	35	0												
442	3	36	0.0975												
443	3	37	0.0154												
444	3	38	0.0673												
445	3	39	0.2406												
446	3	40	0.0238												
447	3	41	0.0408												
448	3	42	0.025												
449	3	43	0.5												
450	3	44	0.5												
451	3	45	0.0111												
452	3	46	0.2656												
453	3	47	1												
454	3	48	1												
455	3	49	0.6												
456	3	50	0.2727												
457	3	51	0.2												
458	3	52	0.8												
459	3	53	0.4522												
460	3	54	0												
461	3	55	1												
462	3	56	1												
463	3	57	1												
464	3	58	0												
465	3	59	0.0556												
466	3	60	0.2333												
467	3	61	0.0333												
468	3	62	0.05												
469	3	63	0												
470	3	64	0.1121												

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
471	3	65	0												
472	3	66	0.2794												
473	3	67	0												
474	3	68	0.6308												
475	3	69	0												
476	3	70	0.3747												
477	3	71	1												
478	3	72	1												
479	3	73	0.3137												
480	3	74	0.281												
481	3	75	0.7571												
482	3	76	0.7												
483	3	77	0.6351												
484	3	78	0												
485	3	79	0.0941												
486	4	1	0.0472												
487	4	2	0.8												
488	4	3	0.824												
489	4	4	0.8571												
490	4	5	0												
491	4	6	0.2												
492	4	7	1												
493	4	8	0.6667												
494	4	9	1												
495	4	10	1												
496	4	11	1												
497	4	12	0.122												
498	4	13	1												
499	4	14	0												
500	4	15	0.0213												
501	4	16	0												
502	4	17	0.6265												
503	4	18	0.4659												
504	4	19	0												
505	4	20	0.125												
506	4	21	1												
507	4	22	0												
508	4	23	0												
509	4	24	1												
510	4	25	0.2756												
511	4	26	0.2832												
512	4	27	0.7568												
513	4	28	0.9384												
514	4	29	0												
515	4	30	0.7037												
516	4	31	1												
517	4	32	1												
518	4	33	0												
519	4	34	0.381												
520	4	35	0.096												
521	4	36	0.0692												
522	4	37	0.0403												
523	4	38	0.1804												
524	4	39	0.3406												
525	4	40	0.4286												
526	4	41	0.1463												
527	4	42	0.125												
528	4	43	0.5												
529	4	44	1												
530	4	45	0.2667												
531	4	46	0.1707												
532	4	47	1												
533	4	48	1												
534	4	49	0.2												
535	4	50	0.2727												
536	4	51	0.2												
537	4	52	0.8												
538	4	53	0.4812												
539	4	54	0												
540	4	55	1												
541	4	56	1												
542	4	57	1												
543	4	58	0												
544	4	59	0.1111												
545	4	60	0.1481												
546	4	61	0.1481												
547	4	62	0.75												
548	4	63	0												
549	4	64	0.303												
550	4	65	0												
551	4	66	0.4019												
552	4	67	0												
553	4	68	0.3846												
554	4	69	0												
555	4	70	0.4819												
556	4	71	1												
557	4	72	1												
558	4	73	0.7745												
559	4	74	0.381												
560	4	75	0.8571												
561	4	76	0.6667												
562	4	77	0.8486												
563	4	78	0												
564	4	79	0.0941												

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
565	5	1	0.2472												
566	5	2	1												
567	5	3	0.824												
568	5	4	0.8571												
569	5	5	0												
570	5	6	0.2												
571	5	7	1												
572	5	8	0.6667												
573	5	9	1												
574	5	10	1												
575	5	11	1												
576	5	12	0.1828												
577	5	13	1												
578	5	14	0												
579	5	15	0.0213												
580	5	16	0												
581	5	17	0.6024												
582	5	18	0.4659												
583	5	19	0												
584	5	20	0.125												
585	5	21	1												
586	5	22	0												
587	5	23	0												
588	5	24	1												
589	5	25	0.2992												
590	5	26	0.2743												
591	5	27	0.7566												
592	5	28	0.9364												
593	5	29	0												
594	5	30	0.7037												
595	5	31	1												
596	5	32	1												
597	5	33	0												
598	5	34	0.361												
599	5	35	0.098												
600	5	36	0.0892												
601	5	37	0.0336												
602	5	38	0.1668												
603	5	39	0.3406												
604	5	40	0.4762												
605	5	41	0.1444												
606	5	42	0												
607	5	43	0.5												
608	5	44	1												
609	5	45	0.1333												
610	5	46	0.1483												
611	5	47	1												
612	5	48	1												
613	5	49	0.2												
614	5	50	0.2727												
615	5	51	0.2												
616	5	52	1												
617	5	53	0.6812												
618	5	54	0												
619	5	55	1												
620	5	56	1												
621	5	57	1												
622	5	58	0												
623	5	59	0.1111												
624	5	60	0.2983												
625	5	61	0.1481												
626	5	62	0.7031												
627	5	63	0												
628	5	64	0.303												
629	5	65	0												
630	5	66	0.4299												
631	5	67	0												
632	5	68	0.4815												
633	5	69	0												
634	5	70	0.4819												
635	5	71	1												
636	5	72	1												
637	5	73	0.7745												
638	5	74	0.361												
639	5	75	0.5714												
640	5	76	0.6667												
641	5	77	0.6216												
642	5	78	0												
643	5	79	0.0588												
644															
645	Replication:	1	ObjFcnVal:	3470171.508											
646	Replication:	2	ObjFcnVal:	4171063.232											
647	Replication:	3	ObjFcnVal:	4140666.361											
648	Replication:	4	ObjFcnVal:	4083821.498											
649	Replication:	5	ObjFcnVal:	4073270.511											
650	Replication:	6	ObjFcnVal:	4066518.933											
651	Replication:	7	ObjFcnVal:	4123966.862											
652	Replication:	8	ObjFcnVal:	4064566.446											
653	Replication:	9	ObjFcnVal:	4065580.472											
654	Replication:	10	ObjFcnVal:	4105893.224											
655	Replication:	11	ObjFcnVal:	3997702.962											
656	Replication:	12	ObjFcnVal:	4163280.136											
657	Replication:	13	ObjFcnVal:	4081784.924											
658	Replication:	14	ObjFcnVal:	4114968.924											

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
659	Replication:	15	ObjFcnVal:	3979167.878											
660	Replication:	16	ObjFcnVal:	4133906.797											
661	Replication:	17	ObjFcnVal:	4059437.72											
662	Replication:	18	ObjFcnVal:	4064040.811											
663	Replication:	19	ObjFcnVal:	4144183.82											
664	Replication:	20	ObjFcnVal:	4114547.027											
665	Replication:	21	ObjFcnVal:	4033955.253											
666	Replication:	22	ObjFcnVal:	4118109.15											
667	Replication:	23	ObjFcnVal:	4074865.598											
668	Replication:	24	ObjFcnVal:	4104057.719											
669	Replication:	25	ObjFcnVal:	3218202.77											
670	Replication:	26	ObjFcnVal:	4119873.814											
671	Replication:	27	ObjFcnVal:	4055655.809											
672	Replication:	28	ObjFcnVal:	4110403.775											
673	Replication:	29	ObjFcnVal:	4122138.386											
674	Replication:	30	ObjFcnVal:	3906799.952											
675	Replication:	31	ObjFcnVal:	4023204.893											
676	Replication:	32	ObjFcnVal:	3966794.499											
677	Replication:	33	ObjFcnVal:	4054583.331											
678	Replication:	34	ObjFcnVal:	4038997.949											
679	Replication:	35	ObjFcnVal:	4068611.826											
680	Replication:	36	ObjFcnVal:	4043854.135											
681	Replication:	37	ObjFcnVal:	4060850.303											
682	Replication:	38	ObjFcnVal:	4083528.006											
683	Replication:	39	ObjFcnVal:	4083010.418											
684	Replication:	40	ObjFcnVal:	4065258.672											
685	Replication:	41	ObjFcnVal:	4178758.029											
686	Replication:	42	ObjFcnVal:	4047517.673											
687	Replication:	43	ObjFcnVal:	4020402.63											
688	Replication:	44	ObjFcnVal:	4082329.211											
689	Replication:	45	ObjFcnVal:	4028317.318											
690	Replication:	46	ObjFcnVal:	4083542.47											
691	Replication:	47	ObjFcnVal:	4133253.865											
692	Replication:	48	ObjFcnVal:	4067929.554											
693	Replication:	49	ObjFcnVal:	4051686.295											
694	Replication:	50	ObjFcnVal:	4031584.275											
695	Replication:	51	ObjFcnVal:	3936811.143											
696	Replication:	52	ObjFcnVal:	4072785.644											
697	Replication:	53	ObjFcnVal:	4111118.07											
698	Replication:	54	ObjFcnVal:	4069812.637											
699	Replication:	55	ObjFcnVal:	4128358.631											
700	Replication:	56	ObjFcnVal:	4156480.696											
701	Replication:	57	ObjFcnVal:	4070882.067											
702	Replication:	58	ObjFcnVal:	4094969.931											
703	Replication:	59	ObjFcnVal:	4159991.225											
704	Replication:	60	ObjFcnVal:	4064475.711											
705	Replication:	61	ObjFcnVal:	4118059.753											
706	Replication:	62	ObjFcnVal:	4105091.869											
707	Replication:	63	ObjFcnVal:	3468738.6											
708	Replication:	64	ObjFcnVal:	4128833.896											
709	Replication:	65	ObjFcnVal:	4163142.513											
710	Replication:	66	ObjFcnVal:	4023805.596											
711	Replication:	67	ObjFcnVal:	4068823.521											
712	Replication:	68	ObjFcnVal:	4066281.463											
713	Replication:	69	ObjFcnVal:	4070238.861											
714	Replication:	70	ObjFcnVal:	4033958.162											
715	Replication:	71	ObjFcnVal:	4078840.1											
716	Replication:	72	ObjFcnVal:	4104199.399											
717	Replication:	73	ObjFcnVal:	3036527.9											
718	Replication:	74	ObjFcnVal:	4063157.768											
719	Replication:	75	ObjFcnVal:	4065759.857											
720	Replication:	76	ObjFcnVal:	4051164.346											
721	Replication:	77	ObjFcnVal:	4041732.83											
722	Replication:	78	ObjFcnVal:	4070052.927											
723	Replication:	79	ObjFcnVal:	4081439.007											
724	Replication:	80	ObjFcnVal:	4051679.999											
725	Replication:	81	ObjFcnVal:	4127226.148											
726	Replication:	82	ObjFcnVal:	4158445.212											
727	Replication:	83	ObjFcnVal:	4150323.018											
728	Replication:	84	ObjFcnVal:	4052817.272											
729	Replication:	85	ObjFcnVal:	4007084.461											
730	Replication:	86	ObjFcnVal:	3987224.724											
731	Replication:	87	ObjFcnVal:	4112188.974											
732	Replication:	88	ObjFcnVal:	4067670.314											
733	Replication:	89	ObjFcnVal:	4085150.86											
734	Replication:	90	ObjFcnVal:	4010501.23											
735	Replication:	91	ObjFcnVal:	4152952.787											
736	Replication:	92	ObjFcnVal:	4131853.078											
737	Replication:	93	ObjFcnVal:	2818557.241											
738	Replication:	94	ObjFcnVal:	4084504.198											
739	Replication:	95	ObjFcnVal:	4063929.653											
740	Replication:	96	ObjFcnVal:	4072446											
741	Replication:	97	ObjFcnVal:	2999360.088											
742	Replication:	98	ObjFcnVal:	4027816.571											
743	Replication:	99	ObjFcnVal:	4104539.545											
744	Replication:	100	ObjFcnVal:	4012412.851											
745															
746															
747	Number	of	Replications:	1000											
748															
749	Objective	Function	Value	(v:	weighted	sum of	TVD):	4010580.612							
750															
751	Standard	Deviation:	245326.1224												

LIST OF REFERENCES

1. Yost, Kirk A., MAJ, USAF, *The Time Strike Munitions Optimization Model*, Naval Postgraduate School, Monterey, CA, January 1996.
2. Devore, Jay L., *Probability and Statistics for Engineering and the Sciences*, Fourth Edition, Duxbury Press, 1995.

INITIAL DISTRIBUTION LIST

	No. of copies
1. Defense Technical Information Center	2
8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	
2. Dudley Knox Library	2
Naval Postgraduate School 411 Dyer Rd. Monterey, CA 93943-5101	
3. AFSAA/SAA/SAQ/SAMCI	1
1570 Air Force Pentagon Washington, DC 20330-1570	
4. Undersea Warfare, Curriculum 525	1
Code 37, Root Hall, Room 103K Naval Postgraduate School Monterey, CA 93943-5002	
5. Professor Alan R. Washburn	2
Code OR/WS Naval Postgraduate School Monterey, CA 93943-5000	
6. Major Kirk A. Yost, USAF	1
Doctoral Candidate, Department of Operations Research Naval Postgraduate School Monterey, CA 93943-5000	
7. Professor Arnold H. Buss	1
Code OR/BU Naval Postgraduate School Monterey, CA 93943-5000	