TRAC-M-TM-17-010
January 2017

Enhancement of the Logistics Battle Command Model

Architecture Upgrades and Attrition Module Development

U.S. Army TRADOC Analysis Center-Monterey
700 Dyer Road, Room 176
Monterey, California 93943-0692

TRAC-M-TM-17-010
January 2017

Enhancement of the Logistics Battle Command Model

Architecture Upgrades and Attrition Module Development

Nathan Parker
Arnold Buss

This study cost the Department of Defense approximately $191,000 expended by TRAC in Fiscal Years 15-17.
Prepared on 20170106
TRAC Project Code # 060119

U.S. Army TRADOC Analysis Center-Monterey
700 Dyer Road, Room 176
Monterey, California 93943-0692

This page left intentionally blank.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE ***(DD-MM-YYYY)***	**2. REPORT TYPE**	**3. DATES COVERED** ***(From - To)***
	Technical Memorandum	May 2015 – October 2016

4. TITLE AND SUBTITLE
Enhancement of the Logistics Battle Command Model Architecture Upgrades and Attrition Module Development

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Nathan Parker
Arnold Buss

5d. PROJECT NUMBER
060119

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
TRADOC Analysis Center - Monterey
700 Dyer Road
Monterey, CA 93940

8. PERFORMING ORGANIZATION REPORT NUMBER
TRAC-M-TR-17-010

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
TRADOC Analysis Center - Lee

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This project enhances the capabilities of the Logistics Battle Command (LBC) model in two key areas: updating the underlying software architecture from 32-bit Java (1.6) to 64-bit Java (1.8) and the development and implementation of a dynamic attrition module.

15. SUBJECT TERMS
Logistics, attrition, discrete event simulation, Simkit, LBC

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
Unclassified					Nathan Parker
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** ***(include area code)***
U	U	U	U	32	(831) 656-7580

This page left intentionally blank.

Table of Contents

Introduction

1. **Purpose.** The purpose of this memorandum is to provide documentation of the project conducted for the TRADOC Analysis Center, Lee (TRAC-LEE) by the TRADOC Analysis Center, Monterey (TRAC-MTRY). This project seeks to enhance the capabilities of the Logistics Battle Command (LBC) model in two key areas: updating the underlying software architecture from 32-bit Java (1.6) to 64-bit Java (1.8) and the development and implementation of a dynamic attrition module.
2. **Background.** The LBC model is a low-resolution, object oriented, stochastics, and discrete event model programmed in Java building largely on the Simkit library. The primary purpose of the LBC model is to support sustainment modeling for supply consumption, distribution, and demand in order to support analytic efforts routinely conducted by TRAC-LEE. The original development of LBC occurred in 2006 as research project lead by TRAC-MTRY in collaboration with TRAC-LEE and the Naval Postgraduate School.
3. **Methodology.** We separate the enhancement of the Logistics Battle Command model into two separate problems sets. The first problem set is to upgrade the underlying software architecture. Appendix B covers the methodology we use for these upgrades. The second problem set is the development and implementation of a dynamic attrition module. Appendix C covers the development and mathematical implementation of our attrition model. In Appendix D, we present an example software implementation of the adjudication portion of the attrition module in the Python coding language.
4. **Results.** We deliver the results of this project as a web accessible software repository available to the project sponsor. Additional individuals or organizations desiring access to the latest version of the Logistics Battle Command software and user manual should contact the TRADOC Analysis Center – Lee or TRADOC Analysis Center – Monterey.

Study Plan

Problem Statement

To enhance the capabilities of the Logistics Battle Command (LBC) model in two key areas: updating the underlying software architecture from 32-bit Java (1.6) to 64-bit Java (1.8), and the development and implementation of a dynamic attrition module.

Project Team

Sponsor Agency:	Morris Hayes TRADOC Analysis Center – Lee morris.g.hayes.civ@mail.mil
TRAC Lead:	Nathan Parker MAJ, AV/FA49 TRADOC Analysis Center – Monterey

nathan.l.parker8.mil@mail.mil

NPS Faculty: Dr. Arnold "Arnie" Buss
Modeling, Virtual Environments and Simulations Institute
Naval Postgraduate School
abuss@nps.edu

Constraints, Limitations, and Assumptions

- Constraints
  - The project must be completed NLT 30SEP16.
  - The final attrition model will be Unclassified.
  - Software tools will be free and open source.
  - The upgraded LBC software will to run on Windows 7 and current MAC operating systems as a minimum specification.
  - The upgraded LBC software will accept Excel and MS Access as minimum input options.
- Limitations
  - None
- Assumptions
  - Free, open source software tools are sufficient to upgrade the LBC architecture.
  - By adjusting threat density and type in the input parameters, the dynamic attrition module can represent the full array of combat environments.
  - Excel and MS Access data tables are sufficiently large to meet all future Scenario requirements.

Methodology

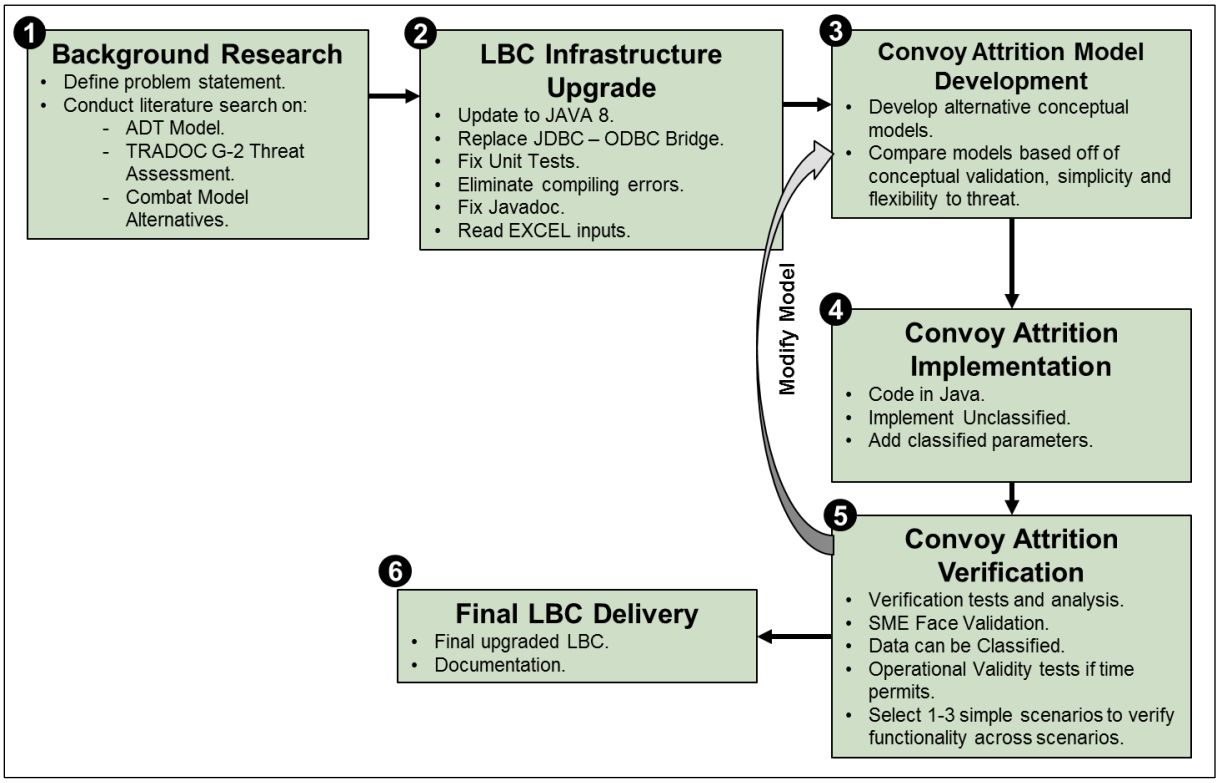

Figure A-1. Project Methodology.

Project Timeline

May 2015 – Receipt of funds.
June 2015 – Initial IPR – Review current LBC capabilities.
September 2015 – IPR #1 – Review architecture upgrade status.
December 2015 – IPR#2 – Review upgrades and attrition module requirements.
January 2016 – Conduct literature review on attrition methods and code initial model.
February 2016 – IPR #3 – Review attrition methodology development.
March 2016 – Test the attrition model on an array of mock combat scenarios.
May 2016 – Integrate attrition model in the LBC construct.
August 2016 – Final IPR – Deliver project requirements.

This page left intentionally blank.

Appendix A - Architecture Updates

Upgrading LBC to Latest 64-Bit Java Version

The previous released version of LBC, 3.15.0 only ran on 32-bit Java, which limits the size of the scenarios that LBC can execute. Furthermore, it uses on an outdated version of Java (1.6). Therefore, the first task is to upgrade LBC to the latest 64-bit Java (1.8).

Issues

The upgrade to 64-bit Java exposed several problems, the most serious of which was the fact that LBC had been using Java's JDBC-ODBC bridge to connect with input files, particularly MS Excel and MS Access. However, Java 1.8 removed the JDBC-ODBC bridge, requiring us to locate a different implementation of the database connectivity (JDBC).

The next issue is that Java 1.8's Javadoc processing implements *doclint*, which produces errors for non-HTML-compliant comments. Many of the Javadoc comments in LBC now fail *doclint* and require correction. Additionally, these discrepancies cause Java to throw thousands of warnings in addition to the outright errors that in turn cause LBC to crash.

A number of LBC's dependent libraries are out-of-date and require updating. This is particularly important for the JDOM package that parses XML files. LBC 3.15.0 uses the old JDOM library, whereas JDOM2 is the most recent version and contains a number of important features, including use of generics in processing collections of XML elements. Other libraries that require upgrades include the NpsTracCommons library.

Finally, many unit tests in LBC 3.15.0 fail, even when running in the older 32-bit mode. These errors should have been corrected prior to the release of LBC 3.15.0 and will require correction prior to the release of our updated version LBC.

In summary, we accomplish the following tasks in order to bring LBC into full compliance with Java 1.8:

1. Convert to Open Source JDBC Driver and update input processing to reflect stricter case-sensitivity compliance.
2. Upgrade all the dependent libraries to the latest versions of each, updating LBC code where necessary to reflect the new versions.
3. Fix all failing unit tests - modifying those necessary to be in full compliance and removing those that depend on the JDBC-ODBC bridge.
4. Fix all Javadoc errors and warnings resulting from Java's *doclint* program being stricter about HTML compliance.

Open Source JDBC Driver for MS Excel and MS Access

A constraint on the LBC upgrade is that all external libraries be Open Source. Therefore, while several commercial JDBC drivers exist for accessing MS Excel and MS Access files they are not available for use. Two Open Source libraries for JDBC access to Excel: xlSQL and SQLSheet. While each of these libraries provides some of the desired functionality, ultimately neither one is adequate for LBC's needs.

A third Open Source JDBC driver is available from in a separate project by the developer, titled movesDB. This pure-Java JDBC driver supports reading both Excel spreadsheets as well as Access database files. While it currently does not support writing via JDBC, that functionality is not necessary for LBC purposes. Therefore, we select movesDB as the replacement JDBC driver for the obsolete JDBC-ODBC bridge.

One consequence of this change is increase in the case-sensitivity of table names and column names. Therefore, we will modify the JDBCtoXML class in LBC, which converts inputs to XML format, to reflect this case sensitivity. Table and column names must match those listed in the LBC User Manual exactly, up to and including case.

Subsequently, we will add a ColumnLint program to LBC in order to identify and correct table and column names that are correct except for case. For example, the table "ScenarioData" is a required one, but if a table named "scenarioData" is present instead, it will produce an error. The ColumnLint program, when run on that input, will identify the error and write the correct version of the input file in a separate location.

Upgrading Dependent Libraries

We upgrade the following libraries: JDOM (to JDOM2), NpsTracCommon, Simkit, and TracBayes. Additionally, we will insert the latest version of Simkit, which requires only minor updates.

Unit Tests

The upgrade to Java 1.8 will cause a significant number of the unit test to fail due to their dependency on the JDBC-ODBC bridge. Each of these individual unit tests will require modification prior to our release of the updated version of LBC.

Javadoc

The Javadoc errors mostly consist of replacing certain symbols with the corresponding HTML code. For example, in Javadoc comments an ampersand (&) must be replaced by & and greater-than/less than signs (> and <) replaced with > and ≤, respectively.

The warnings mostly consist of missing Javadoc for parameters and/or return values for methods. Since there are thousands of these, it will require a significant amount of time to correct them. These modifications are necessary to ensure the Javadoc for the upgrade version of LBC is compliant with general standards.

Appendix B – Dynamic Attrition Methodology

Overview

The second portion of this project focuses on the development of a dynamic attrition methodology and the subsequent implementation within the Logistics Battle Command (LBC) model.

Literature Review

Two historic attrition methodologies and a current model proposed by analysts at TRAC – Fort Leavenworth (TRAC-FLVN) inform the development of the dynamic attrition methodology we propose for LBC. Each of these methodologies or models contributes a specific part to the logical development of our dynamic attrition methodology.

The first historic attrition methodology we consider is the Lanchester Equations, specifically the variation proposed by Deitchman.[1] This work provides an initial option for modeling attrition between two very imbalanced forces such as an ambush style attack. However, the deterministic and continuous nature of the Lanchester equations makes them incompatible with the discrete event construct of LBC. Bullard further advances this methodology by developing a stochastic variation of the Lanchester equations.[2] The primary concept we adopt from the combination of these theories is the ability to use a ratio of relative combat power scores to determine the transition probability in a discrete Markov process.

The second historic attrition methodology we consider is the Hughes Salvo equations. While these equations find their widest use in the naval warfare community, we observe several features that can inform our dynamic attrition methodology. The Hughes Salvo equations consider each warship as a discrete entity with multiple attribute including staying power, striking power, defensive capability, and more depending on the embellishment under consideration.[3] From the Hughes Salvo equations, we draw the concept of treating the attacking and defending forces as holistic entities. Additionally, the calculation a relative combat power for each step of the engagement sequence is an independent event.

We also consider the Attrition Distribution Tool (ADT), developed by TRAC-FLVN, since one of its intended purposes is to develop attrition distributions for use in non-combat models such at LBC. Our primary concern with carrying the ADT forward as a primary component of our dynamic attrition model is the high-resolution inputs the model requires. We find it unlikely that future modelers, especially those conducting logistic analysis, will have access to fidelity of data necessary to population the model. These complexities also make any sensitivity analysis or scenario excursions more difficult. The ADT developers propose a zone construct for modeling

[1] S. J. Deitchman. *A Lanchester Model of Guerrilla Warfare*. Operations Research 10(6):818-827. http://dx.doi.org/10.1287/opre.10.6.818, 1962.
[2] L. Billard. *Stochastic Lanchester-type Combat Models I.* Technical Report Naval Postgraduate School. http://hdl.handle.net/10945/30208, 1979.
[3] W.P. Hughes Jr. *Two Effects of Firepower: Attrition and Suppression.* Military Operations Research. Vol. 2, No. 1, Winter 1996.

the array of enemy forces through the different battlefield areas logistic type elements are likely to operate. We find this construct extremely useful and plan to use it in our methodology.

We believe that the primary method of attack against a logistics element is an ambush style attack. Two different types of ambush style attacks, the annihilation ambush and the harassing ambush, will form the basis for how we model enemy actions in our dynamic attrition model.[4] In the annihilation ambush, the attacker's primary objective is to exact maximum damage on the convoy so they will risk becoming decisively engaged. In the harassment ambush, the attacker's primary objective is to maintain combat power so they will not risk becoming decisively engaged.

Conceptual Development

Our goal is to provide a dynamic attrition model that sufficiently represents the impacts of attrition on logistics operations and has the following characteristics: uses level of resolution similar to that of LBC, uses input data readily available to TRAC-LEE analysts, has a low computational overhead and is easy to understand for both analysts and warfighters. In addition to the literature discussed above, the model development will draw heavily on the combat experience of the TRAC-MTRY analysts for both conceptual development and face validation.

Red Force Representation

We will represent the area of operations as a series of zones within which the enemy elements will execute attacks against convoys. To best match LBC's level of resolution we will model attrition at the vehicle level for blue (friendly) elements and the crew served weapon (CSW) for red (enemy) elements. We leave the specific definition of a CSW up to the analyst using the model but we envision the definition including medium and heavy machine guns, rocket propelled grenades, anti-tank rockets, and improvised explosive devices. We also implement an aggression level parameter for each zone that will control the mixture of annihilation and harassing type ambushes allowing the model to cover an array of enemy force objectives. Figure C-1 provides a depiction of this implementation concept.

[4] Headquarters Department of the Army. *Opposing Force Tactics*. Training Circular 7-100.2. Government Print Office, Washington D.C. 2011.

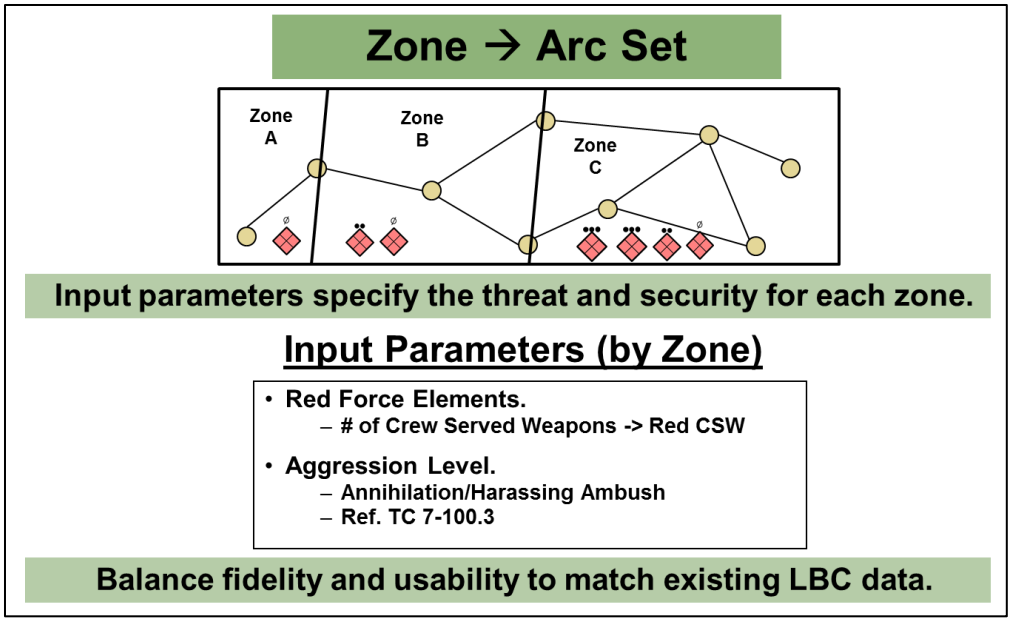

Figure C-1. Concept sketch for the array for Red Force elements.

We use a simple discrete event construct to model the red forces cycle of attack emplacement, execution and recovery for each red force element. Each red force element has five parameters: the number of CSW in the element, the element's effectiveness level [0-1], the mean recovery time if the element's previous action is not the execution of an ambush, the mean recovery time in the element's previous action is the execution of an ambush, and the mean time the element will spend in an attack position before it withdraws. Each attack element randomly chooses an arc, from the arc set of their zone, on which to set up their ambush each time they become active. The event graph in Figure C-2 depicts this process.

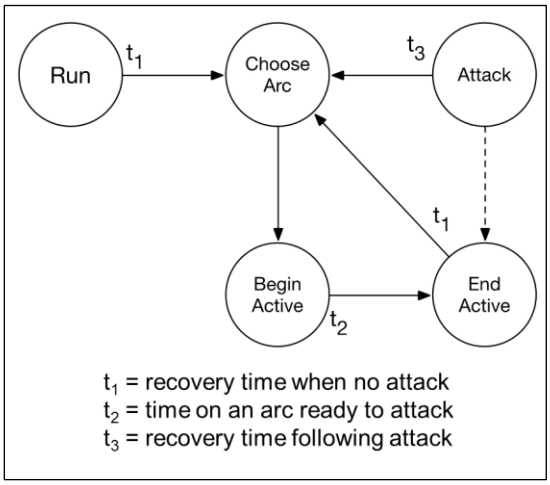

Figure C-2. Event Graph for the generation of Red Force attacks.

Blue Force Representation

The existing LBC model has the capability of representing multiple vehicle types within the blue force structure. We leverage this capability and add a Blue Escort Vehicle type. Similar to the Red CSW concept, we leave the definition of what is an Escort Vehicle up to the model user. Here we envision an armored vehicle with a crew served weapons system whose sole responsibility is to provide security for the logistics convoy. We include functionality in the model to allow the model user to define the force structure of the Escort Vehicles and to establish business rules regarding the requirements for Escort Vehicles in each convoy.

Model Component Integration

The red and blue force representations discussed above, along with the attack adjudication methodology we will develop later in this paper, each correspond to separate model components. At the software implementation level we employ a loosely coupled, modular design using the Java event listener construct to integrate these separate components. This modular design provides flexibility for further development and customization since it is easy to switch out one or more modules as along as the input and output formats remain the same. Figure C-3 depicts the simplicity of this modular design.

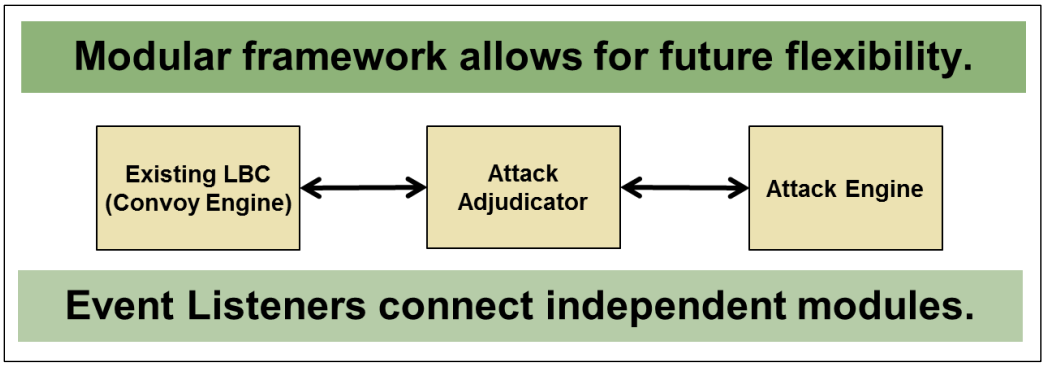

Figure C-3. Modular framework for individual model components.

Attack Adjudication Methodology

To limit the computational complexity our attack adjudication method invokes the Markov property so that the outcome of any step of the attack sequence depends only on the current conditions. We use a relative comparison of the current red and blue combat powers, along with a random number draw, to determine who suffers the loss in each step. To further limit complexity, we do not model the time component within the attack sequence, since the magnitude of time passage in an ambush style attack is insignificant given LBC's level of resolution. Figure C-4 depicts the inputs, process, and outputs steps of the attack adjudication sequence. Subsequence sections will develop each process step in additional detail.

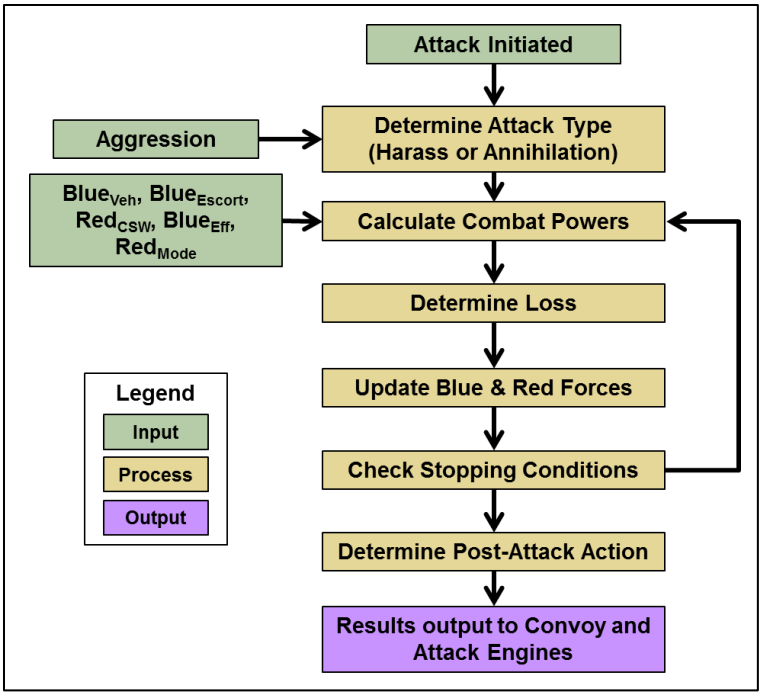

Figure C-4. Concept sketch of the attack adjudication methodology.

Determining Attack Type

We determine the attack type, annihilation or harassment, using the aggression level parameter from the associated zone and a random number draw from a uniform distribution. Equation 3-1 shows how we determine each attack type.

$$\text{Attack Type} = \begin{cases} \text{Annihilation, } X < \dfrac{\text{Agression Level}}{100} \\ \text{Harrassment, otherwise} \end{cases} \qquad \text{(C-1)}$$

$$\text{Where } X \sim U[0,1]$$

We employ a generalized concept of each attack type to develop the engagement sequence and stopping conditions associated with each type. These concepts originate from the *Opposing Force Tactics* training circular with further refinement and validation from the TRAC-MTRY military analysts. We provide a brief explanation of each attack type in the sections below along with a consolidated reference in Figure B-5. The limitations and structures we place on each attack type are somewhat arbitrary and future users may desire to modify these features to fit individual needs.

Harassment Attack Type

For the harassment attack, we assume that the red force will immediately disengage if the blue force returns effective fire. For the attack sequence, we assume that the attacker will target the logistic elements of the convoy to maximize the impacts on the logistics system. Additionally, we assume that the attacker is not subject to attrition but rather use the determination of a red force loss as a stopping condition without actually enforcing the loss. We limit the maximum amount of damage that the red force can inflict to represent their intent to quickly withdraw from the engagement area and not risk decisive engagement. The two stopping conditions for the harassment attack are if Blue losses reach the maximum allowable level or if an engagement outcome is a Red loss (though the loss is not enforced).

Annihilation Attack Type

For the annihilation attack, we assume that the red force is seeking a decisive engagement and will press the attack until they expend all their ammunition, wipe out the Blue convoy or loss half of their force. For the attack sequence, we assume that the attacker will target the escort elements of the convoy first to eliminate their ability to mount an effective defense. We limit the number of blue vehicles that the red force can destroy to represent limitations in ammunition supply and the limited kill zone that a given size force can establish. The three stopping conditions for the annihilation attack are if Blue losses reach the maximum allowable level, if the Blue convoy is annihilated or if the Red losses reach the maximum allowable level.

Harassing Attack	Annihilation Attack
• Red targets Blue Vehicles first.	• Red targets Blue Escorts first.
• $\boldsymbol{Blue\ Loss_{Max} = Ceiling(1.5 * Red_{CSW})}$	• $\boldsymbol{Blue\ Loss_{Max} = 3 * Red_{CSW}}$
• Red not subject to attrition.	• $\boldsymbol{Red\ Loss_{Max} = Floor(.5 * Red_{CSW})}$
• Stopping Conditions: – $\boldsymbol{Blue\ Loss = Blue\ Loss_{Max}}$ – Red "Loss"	• Stopping Conditions: – $\boldsymbol{Blue\ Loss = Blue\ Loss_{Max}}$ – $\boldsymbol{Red\ Loss = Red\ Loss_{Max}}$ – Blue convoy annihilated.

Figure C-5. Summary of attack type engagement sequence and stopping conditions.

Calculating Combat Power

We use the concept of combat power to represent the capability of each force, red and blue, to attrite the other element. The Red's crew served weapon systems and Blue's escort vehicles provide the base input for each calculation. For the blue combat power ($Blue_{CP}$) calculation (Equations C-2 and C-3), we account for both the overall force effectiveness and the effect of the dispersion of escort vehicles within the convoy. For the red combat power (Red_{CP}) calculation (Equations C-4 and C-5), we only adjust for the overall effectiveness. Here we use a random effectiveness coefficient from a triangular distribution using one user-defined parameter. We use

this distribution as an example to show how one might add additional distributional data to the adjudication methodology.

$$Blue_{CP} = Blue_{Escorts} * Blue_{Effectiveness} * Vehicle\ Ratio \tag{C-2}$$

$$Vehicle\ Ratio = \frac{Blue_{Escorts}}{Blue_{Escorts} + Blue_{Logistic\ Vehicles}} \tag{C-3}$$

$$Red_{CP} = Red_{CSW} * Red_{Effectiveness} \tag{C-4}$$

$$Red_{Effectiveness} = \min\left[Tri(Red_{Mode} - .2, Red_{Mode} + .2, Red_{Mode}), 1\right] \tag{C-5}$$

Determine Loss for each Engagement Step

We determine the loss for each engagement step using a ratio of the red and blue combat powers and a random draw from a uniform distribution. Equation C-6 shows how we determine the loss for each engagement step.

$$\text{Engagement Loss} = \begin{cases} \text{Blue}, X \le \dfrac{Red_{CP}}{Red_{CP} + Blue_{CP}} \\ \text{Red, otherwise} \end{cases} \quad \text{Where } X \sim U[0,1] \tag{C-6}$$

Post-Attack Actions

The last step in our attrition adjudication methodology is to determine the post attack actions for the convoy. We use a simple logic flow based on the collective combat experience of the TRAC-MTRY analysts. If the attack occurs in the last 35% of the route then the convoy with continue mission not matter the outcome of the attack. If the attack occurs in the first 65% of the route, then the convoy must have at least two escort vehicles and greater than 50% of its original logistic vehicle to continue mission, otherwise the convoy will return to base.

Iterative Engagement Sequence

We treat each step of the engagement sequence as an independent event so that the probability that the next loss is red or blue depends only on the current force structure and the parameters that remain constant for the engagement sequence. Figure C-6 shows all possible iterative steps and outcomes for both the annihilation and harassment attack examples. The asterisk on the right branches of the harassment attack example indicates that the loss of the Red CSW is notational and serves only as a stopping condition.

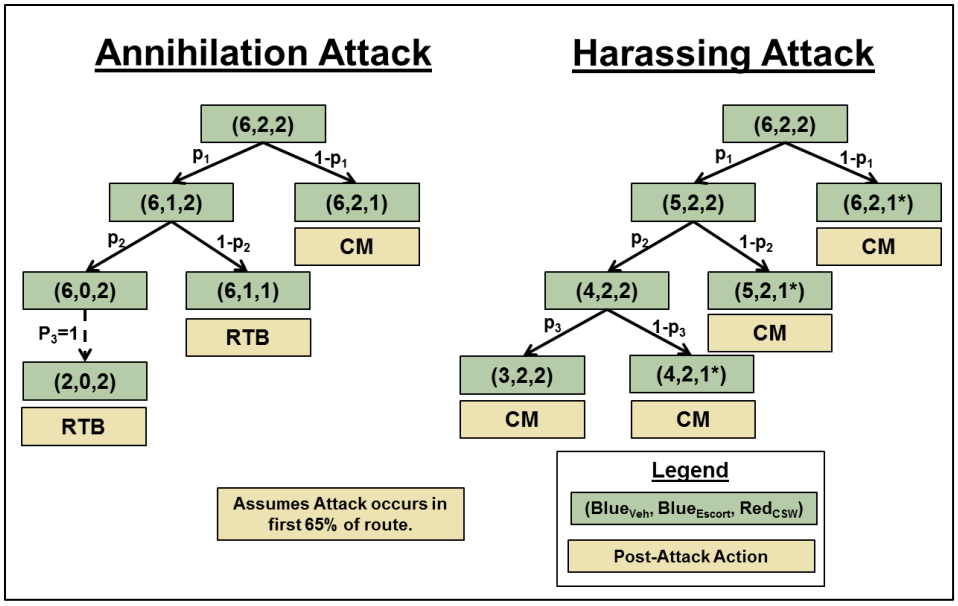

Figure C-6. Two examples of the attack adjudication sequence.

Appendix C – An Example Implementation of the Engagement Adjudication Methodology in Python

Overview

This appendix contains commented Python code for an example implementation of the engagement adjudication methodology discussed in Appendix C. This adjudication implementation requires the following six inputs: number of Blue logistics vehicles, number of Blue escort vehicles, number for Red crew served weapons, Red's aggression level, Blue's effectiveness level, and the mode of Red's effectiveness level. From these inputs the methodology produces the following outputs: remaining Blue logistic vehicles, remaining Blue escort vehicles, remaining Red crew served weapons and Blue's post attack actions.

The first section below documents the various functions that serve as the base for the adjudication implementation. The second section includes the code necessary to run the implementation from command lining using a comma separated value file to provide the inputs.

Attrition Functions

The Python code we include below contains the various functions that form the foundation of this implementation of the adjudication methodology. Where appropriate, the code references the associated equitation or figures from Appendix C.

```
#Import the math and random libraries.

import random
import math


def getBlueCombatPower(blue_veh, blue_escort, blue_eff):
    """Calculate the Blue Combat Power at each state."""
    #Calculate escort to log veh ratio (Equation C-2).

    veh_ratio = blue_escort/float(blue_veh + blue_escort)

    #Calculate Blue Combat Power (Equation C-3).

    BlueCP = blue_escort * blue_eff * veh_ratio

    return BlueCP


def getRedCombatPower(red_csw, red_eff):
    """Calculate the Red Combat Power at each state."""
    #Calculate Red Combat Power (Equation C-4).

    RedCP = red_csw * red_eff

    return RedCP

def getRedEff_2(red_eff):
    """Simplified version using only the mode as input.
    Still use tri dist with low, high calc'd off the mode."""
```

```
    #Calculate Red Effectiveness (Equation C-5).
    low = red_eff - .2
    high = red_eff + .2

    redEffOut = random.triangular(low, high, red_eff)

    if redEffOut > 1:
        redEffOut = 1
    else:
        pass

    return redEffOut


def whoLost(BlueCP, RedCP):
    """Determine if loss for a given state change is Red or Blue
    based on the relative combat powers."""
    #Calc probability that loss is blue (Equation C-6).

    pBlueLoss = RedCP/float(RedCP+BlueCP)

    #Pull random ~U(0,1)

    roll = random.random()

    if roll <= pBlueLoss:
        loss = "Blue"
    else:
        loss = "Red"
    return loss




def getAttackType(aggression):
    """Determine whether attack is harrass or annihiliation based on aggression coeff."
""
    #Equation C-1

    roll = random.random()

    if roll <= aggression:
        attack = "Annihilation"
    else:
        attack = "Harrass"

    return attack


def harassAttack(blue_veh, blue_escort, red_csw, blue_eff, red_eff):
    """Assumptions:
        - Red is targets Log Vehicles first.
        - Loss of Blue Log Vehicles limited to ceiling(1.5 x red_csw).
        - Red not susceptible to attrition.
        - Stopping conditions:
            - Red "Loss".
            - Max attrition reached.
    """
    #Figure C-5.

    i_blue_veh = blue_veh
```

```
    i_blue_escort = blue_escort

    blue_veh_lost = 0
    blue_escort_lost = 0

    max_blue_loss = math.ceil(1.5 * red_csw)

    while (blue_veh_lost + blue_escort_lost) < max_blue_loss:

                r_blue_veh = i_blue_veh - blue_veh_lost
                r_blue_escort = i_blue_escort - blue_escort_lost

                BlueCP = getBlueCombatPower(r_blue_veh, r_blue_escort, blue_eff)
                RedCP = getRedCombatPower(red_csw, red_eff)

                loss = whoLost(BlueCP, RedCP)

                if loss == "Blue":
                    if r_blue_veh > 0:
                        blue_veh_lost += 1
                    elif r_blue_escort > 0:
                        blue_escort_lost += 1
                    else:
                        break
                else:
                    break

            f_blue_veh = i_blue_veh - blue_veh_lost
            f_blue_escort = i_blue_escort - blue_escort_lost

            outcome = (f_blue_veh, f_blue_escort, red_csw)

            return outcome

def annihilationAttack(blue_veh, blue_escort, red_csw, blue_eff, red_eff):
            """Assumptions:
                - Red targets Escorts first.
                - Red is susceptible to attrition.
                - Stopping conditions:
                    - Red loss = floor(.5 * red_csw)
                    - Max attrition reached - attrition limited to 3 x red CSW
                        *Captures ability of blue veh outside kill zone to avoid attack
                    - Blue wiped out.
            """
            #Figure C-5.

            i_blue_veh = blue_veh
            i_blue_escort = blue_escort
            i_red_csw = red_csw

            blue_veh_lost = 0
            blue_escort_lost = 0
            red_csw_lost = 0

            ##If, then acception for when initial CSW = 1
            if i_red_csw == 1:
                max_red_loss = 1
            else:
                max_red_loss = math.floor(.5 * red_csw)

            max_blue_loss = min(3* i_red_csw, i_blue_veh + i_blue_escort)
```

```
152.
153.           while (((blue_veh_lost + blue_escort_lost) < max_blue_loss) and (red_csw_los
    t < max_red_loss)):
154.
155.               r_blue_veh = i_blue_veh - blue_veh_lost
156.               r_blue_escort = i_blue_escort - blue_escort_lost
157.               r_red_csw = i_red_csw - red_csw_lost
158.
159.               BlueCP = getBlueCombatPower(r_blue_veh, r_blue_escort, blue_eff)
160.               RedCP = getRedCombatPower(r_red_csw, red_eff)
161.
162.               loss = whoLost(BlueCP, RedCP)
163.
164.               if loss == "Blue":
165.                   if r_blue_escort > 0:
166.                       blue_escort_lost += 1
167.                   elif r_blue_veh > 0:
168.                       blue_veh_lost += 1
169.                   else:
170.                       break
171.               else:
172.                   red_csw_lost += 1
173.
174.           f_blue_veh = i_blue_veh - blue_veh_lost
175.           f_blue_escort = i_blue_escort - blue_escort_lost
176.           f_red_csw = i_red_csw - red_csw_lost
177.
178.           outcome = (f_blue_veh, f_blue_escort, f_red_csw)
179.
180.           return outcome
181.
182.  def getPostAttackAction(i_blue_veh, r_blue_veh, r_blue_escort):
183.           """Determine if convoy continues mission (CM) or returns
184.           to base (RTB).  If attack occurs after .65 of route covered then
185.           always CM.  If in the first .65 convoy must retain >=.5 of original
186.           logistics vehicles and at least 2 escorts to CM."""
187.
188.           loc_roll = random.random()
189.
190.           if loc_roll >= .65:
191.               action = "CM"
192.           elif r_blue_escort < 2:
193.               action = "RTB"
194.           elif (r_blue_veh/float(i_blue_veh)) < .5:
195.               action = "RTB"
196.           else:
197.               action = "CM"
198.
199.           return action
200.
201.
202. def runAttack(blue_veh, blue_escort, red_csw, aggression, blue_eff, red_mode):
203.           ####Using simplied function for red_eff, only requires red_mode input.
204.           red_eff = getRedEff_2(red_mode)
205.
206.           attackType = getAttackType(aggression)
207.
208.           if attackType == "Annihilation":
209.               outcome = annihilationAttack(blue_veh, blue_escort, red_csw, blue_eff, r
    ed_eff)
210.
```

```
            else:
                outcome = harassAttack(blue_veh, blue_escort, red_csw, blue_eff, red_eff
    )

            action = getPostAttackAction(blue_veh, outcome[0], outcome[1])

            output = ((blue_veh, blue_escort, red_csw), outcome, action)

            return output
```

Execution Code

This section contains the Python code necessary to run our implementation from the command line. This execution code expects to receive a comma separated value (csv) file containing the model inputs. The csv file should have a header row and have columns in the following order: Blue logistic vehicles, Blue escort vehicles, Red crew served weapons, Red's aggression level, Blue's effectiveness level, and the mode of Red's effectiveness level triangular distribution. Figure D-1 shows an example of an appropriately formatted input file.

	A	B	C	D	E	F
1	Blue_Veh	Blue_Escort	Red_CSW	Aggression	Blue_Eff	Red_Mode
2	4	2	1	0	0.5	0.8
3	6	2	1	0	0.5	0.8
4	8	2	1	25	0.5	0.8
5	4	4	1	25	0.5	0.8
6	6	4	1	75	0.5	0.8
7	8	4	1	75	0.5	0.8
8	4	2	2	100	0.5	0.8
9	6	2	2	100	0.5	0.8
10	8	2	2	0	0.5	0.8
11	4	4	2	0	0.5	0.8
12	6	4	2	25	0.5	0.8
13	8	4	2	25	0.5	0.8
14	4	2	4	75	0.5	0.8
15	6	2	4	75	0.5	0.8
16	8	2	4	100	0.5	0.8
17	4	4	4	100	0.5	0.8
18	6	4	4	0	0.5	0.8
19	8	4	4	100	0.5	0.8

Figure D-1. Example format for the input comma separated value file.

```
#Import the sys and csv libraries.
import sys
import csv
## Attrition_Part1_v1 is the .py file containing the functions.
import Attrition_Part1_v1 as AttritionFunctions

## initial set input parameters for cmd line running
# the below if/else statement serve as a logic check
# if the user fails to enter the proper number of arguments in the command line
if len(sys.argv) != 4:
    sys.stderr.write('expecting Command Line Arguements: input_filename, output_filenam
e, number of iterations')
    sys.exit(1)

else:
```

```
15.
16.      input_filename = sys.argv[1]
17.      output_filename = sys.argv[2]
18.      iterations = int(sys.argv[3])
19.
20. row_cnt = 0
21.
22. result_list = []
23.
24. with open(input_filename, 'r') as f:
25.     r = csv.reader(f)
26.
27.     #skip header row
28.     r.next()
29.
30.     #read in file line by line
31.     for row in r:
32.
33.         row_cnt += 1
34.
35.         #Assign values to variables
36.         blue_veh = int(row[0])
37.         blue_escort = int(row[1])
38.         red_csw = int(row[2])
39.         aggression = float(row[3])
40.         blue_eff = float(row[4])
41.         red_mode = float(row[5])
42.         #red_high = float(row[6])
43.         #red_mode = float(row[7])
44.
45.          #Run the specified number of iterations
46.         for i in range(iterations):
47.
48.             result = AttritionFunctions.runAttack(blue_veh, blue_escort, red_csw, aggre
    ssion, blue_eff, red_mode)
49.
50.             result_list.append(result)
51.
52. output = open(output_filename , 'wb')  #establish output.csv file with handle
53.
54. #Write Header
55. output.write('Index,Design Point, Blue_Veh,Blue_Escort,Red_CSW,r_B_v,r_B_e,r_R_csw,Post
    _Attack_Action\n')
56.
57. for i in range(len(result_list)):
58.
59.         output.write(str(i) +',')
60.         output.write(str(result_list[i][0][0]) + '_' + str(result_list[i][0][1]) + '_'
    + str(result_list[i][0][2]) + ',')      #Write string of design point (Blue Veh, Blue_E
    scort,Red_CSW)
61.         output.write(str(result_list[i][0][0]) +',')    #write inital Blue Veh
62.         output.write(str(result_list[i][0][1]) +',')    #write inital Blue Escort
63.         output.write(str(result_list[i][0][2]) +',')    #write inital Red CSW
64.         output.write(str(result_list[i][1][0]) +',')    #write final Blue Veh
65.         output.write(str(result_list[i][1][1]) +',')    #write final Blue Escort
66.         output.write(str(result_list[i][1][2]) +',')    #write final Blue CSW
67.         output.write(str(result_list[i][2]) +'\n')      #write Post Attack Action
68.
69. output.close()  #always close
```

Calling the Execution Code from the Command Line

To call the execution code from the command line the following structure is required:

python Attrition_Part2_v2.py test_design.csv output.csv 1000

After identifying the program used this command line call includes the python file contain the execution code as the first argument. The second and third arguments are the input and output csv files, respectively. The forth agreement is the number of iterations being run.

This page left intentionally blank.

Appendix D - References

L. Billard. *Stochastic Lanchester-type Combat Models I.* Technical Report Naval Postgraduate School. http://hdl.handle.net/10945/30208, 1979.

S. J. Deitchman. *A Lanchester Model of Guerrilla Warfare*. Operations Research 10(6):818-827. http://dx.doi.org/10.1287/opre.10.6.818, 1962.

W.P. Hughes Jr. *Two Effects of Firepower: Attrition and Suppression.* Military Operations Research. Vol. 2, No. 1, Winter 1996.

This page left intentionally blank.

Appendix E - Glossary

ADT	Attrition Distribution Tool
CSW	Crew Served Weapon
FLVN	Fort Leavenworth
LBC	Logistics Battle Command
MTRY	Monterey
TRAC	TRADOC Analysis Center