VERIFICATION AND VALIDATION OF AGENT-BASED AND EQUATION-BASED SIMULATIONS AND BIOINFORMATICS COMPUTING: IDENTIFYING TRANSPOSABLE ELEMENTS IN THE *Aedes aegypti* GENOME

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

by

Ryan C. Kennedy, B.S.

Gregory R. Madey, Director

Graduate Program in Department of Computer Science and Engineering
Notre Dame, Indiana
April 2006

VERIFICATION AND VALIDATION OF AGENT-BASED AND EQUATION-BASED SIMULATIONS AND BIOINFORMATICS COMPUTING: IDENTIFYING TRANSPOSABLE ELEMENTS IN THE *Aedes aegypti* GENOME

Abstract

by

Ryan C. Kennedy

This work centers around two subjects: 1) verification and validation of simulation models and 2) identifying transposable elements through novel approaches.

Performing verification and validation on simulations has become increasingly important as they are used in more and more applications. Agent-based simulations are becoming very popular and the verification and validation work previously performed on them is minimal. Here, we perform verification and validation techniques, including docking and visualization, on an agent-based and an equation-based model and contrast the effectiveness of the techniques used.

Identifying transposable elements in a genome is important for many reasons, such as when performing evolutionary studies or when annotating genomes. Here, we identify transposable elements in the newly released *Aedes aegypti* genome using innovative approaches, while utilizing some standard bioinformatics tools.

DEDICATION

To my Family

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

I would especially like to thank Dr. Gregory R. Madey for providing me with the opportunity to pursue advanced study and for guiding me along the way.

I would also like to thank Xiaorong Xiang and Dr. Yingping Huang for their help in getting me started and also for developing the core of the NOM Model and its interface. Thank you to Dr. Patricia A. Maurice, Dr. Stephen E. Cabaniss, Dr. Benjamin F. Turner, and Leilani A. Arthurs for their contributions to the NOM project.

Further thanks go to Scott E. Christley and Tim Schoenharl for general guidance and for helping me format this document.

I would like to thank my committee, Dr. Frank H. Collins, Dr. Amitabh Chaudhary, and Dr. Thomas F. Cosimano, for their valuable contributions.

The bioinformatics aspect of this thesis would not have been possible without the help of Dr. Frank H. Collins and James R. Hogan. Their help in providing me with a solid base in the area made this work possible. Further, Trevor M. Cickovski played an integral role in a portion of the bioinformatics aspect.

It is also important to thank the National Science Foundation, Information Technology Research/(ITR/AP-DEB), grant number 0112820, the Center for Environmental Science and Technology (CEST) at Notre Dame, and the University of Notre Dame Office of Research. They each provided funding for my work.

CHAPTER 1

INTRODUCTION

1.1 Overview

The work presented here consists of two distinct parts. Part one regards verification and validation of agent-based and equation-based simulations. Part two concerns identifying transposable elements through the use of some novel approaches and is in the area of bioinformatics computing. We next provide a brief introduction to these chapters, including our motivations as well as some of our conclusions.

1.2 Verification and Validation of Agent-based and Equation-based Simulations

As the use of simulations, specifically agent-based simulations, by researchers grows [71], there is more and more need for the verification and validation of such simulations [62]. While there has been extensive research regarding the verification and validation of simulations using traditional engineering methods [6, 47, 76], such as equation-based modeling, there has been little work performed regarding the verification and validation of agent-based simulations. Part one of this thesis explores verification and validation techniques through two very different case studies - an agent-based scientific model and an equation-based economic model.

The issue of verification and validation for simulation models is very important. Because the typical goal of a simulation model is to mimic a real-world phenomenon, no real value can be gained from the simulation unless it is sufficiently accurate. Determining when this is the case is not trivial. To attack this problem, we evaluate some typical verification and validation techniques and apply them to both an agent-based and an equation-based model. Specifically, we look at docking and visualization as effective verification and validation tools. This approach allows us to contrast some documented verification and validation techniques on the two case studies. This is especially valuable because the case studies are inherently very different internally, but have similar external objectives. We offer a brief survey of how valuable we found the techniques for each simulation case study and offer some insight about our results. The results of this part of the thesis have been presented and published (in part) in the proceedings of three peer-reviewed, national conferences [44, 46, 87].

1.3 Bioinformatics Computing: Identifying Transposable Elements in the *Aedes aegypti* Genome

The bioinformatics field is growing at a sharp rate [12, 13], as evidenced by the exponential growth in the amount of data stored in bioinformatics databases [30, 60]. It is bringing with it many opportunities for computer scientists, as computer scientists have the skills necessary for major advancement in the field. In short, bioinformatics is the study of solving biological problems with the collective expertise of many fields, most notably computer science and mathematics. Typically, bioinformatics is concerned with the study of DNA and its complex properties.

Genomes are generally very large in size. For example, the human genome is made up of about three billion base pairs. Even less "complex" organisms, such as the mosquito *Aedes aegypti* have over one billion base pairs [17]. Transposable elements are found within many genomes and typically have the ability to replicate and insert themselves anywhere in the genome. This makes them useful as gene vectors to biologists. The *Aedes aegypti* genome is specifically important to study because it carries the yellow fever and dengue viruses. Its genome was recently released [60], so there has been little work done to date regarding locating transposable elements within it. In this thesis, we design and utilize novel approaches to locate transposable elements within the *Aedes aegypti* genome. Our goal is not only to identify transposable elements, but to do so using techniques not normally employed by biologists, thus advancing the state-of-the-art in this field.

1.4 Organization

The remainder of this thesis is organized as follows. Chapter 2 describes verification and validation of simulations through two case studies. Chapter 3 involves bioinformatics computing and our search for transposable elements through innovative methods. A conclusion for both chapters is presented in Chapter 4. We conclude this thesis with supplementary material in the appendices.

CHAPTER 2

VERIFICATION AND VALIDATION OF AGENT-BASED AND EQUATION-BASED SIMULATIONS[1,2]

2.1 Introduction

Agent-based simulations are quickly becoming a choice tool for many researchers [71]. They can offer numerous benefits over some traditional simulation methods and tend to be more adept at modeling natural phenomena. To build a typical simulation model, a researcher first creates a conceptual model. A conceptual model is how the model will be abstractly represented in the computer. From here, the model is coded. Unfortunately, the process stops here for many researchers, as extensive verification and validation on simulations, specifically agent-based, is often overlooked. This is partially due to the fact that new theory and techniques are needed for such stochastic models [62]. However, existing verification and validation techniques can be used on or adapted for agent-based simulations. In this chapter, we explore verification and validation techniques for both an agent-based and an equation-based model, offering insight regarding the techniques. We specifically examine docking and visualization as effective verification and validation techniques. This is accomplished through two case studies.

[1]The NOM Model was developed by a number of researchers in a variety of fields [2, 37, 39, 85, 86]. A further discussion of the NOM project can be found in Appendix A.1.

[2]Portions of this chapter have been published in the proceedings of several conferences [44, 46, 87]

In the first case study, we look at verification and validation of an agent-based model. Specifically, we look at docking the simulation model against an independently developed simulation model. We also use the visualization method. The second case study looks at an equation-based model. We compare this model's results and performance through two different programming languages. We also utilize numerous other verification and validation techniques for both case studies and offer some insight as to which have been the most valuable.

2.2 Simulations

A simulation is an imitation of a real-world process [10]. This imitation is usually done with a computer through the use of a conceptual model. A conceptual model generally refers to the computer representation of the system that the researcher has chosen to model. The goal of every simulation is to accurately represent the behavior of the real-world system while providing feedback and insight in a manner that would otherwise be infeasible. For example, a chemical experiment that takes months to complete in the laboratory may take only hours or days to complete with a computer simulation. It is appropriate to think of simulations as parts of the scientific method - we use them to help us check our assumptions or hypotheses as well as to possibly predict future behavior. As is also the goal with the scientific method, we utilize simulations to help us acquire new knowledge.

The literature lists many reasons why computer simulations are valuable [10, 58, 78], a collection of which are presented below:

1. Simulations allow for the timely study of phenomena that would otherwise be impractical to study. For example, the evolution of natural organic matter

over long periods of time can be simulated in far less time than the actual experiments would take to perform.

2. Simulations can model theoretical behavior that cannot be replicated in the laboratory. An example of this would be a simulation model that tracked the historical migration patterns of icebergs or continental drift.

3. Simulation inputs can be tweaked to determine the outcome or effect on a real-world system without harming the real-world system. This would be applicable if a researcher wanted to simulate the spread of a disease across a population without harming the population.

4. Experimentation with simulations can confirm understanding. For example, a simulation model that mimics the population dynamics of a group of animals could allow researchers to examine particular entities of the model and follow them over time. This would further the understanding of the system.

5. Simulations can be used as prototypes for new experiments before real-world implementation. For example, a disaster recovery team could simulate any sort of disaster and their response tactics, allowing them to choose the best approach.

6. Modern systems are sometimes so complex that their internal workings can only be studied through simulations. The best example here is nature's vast complexity. We cannot possible mimic every aspect of an environment through a simulation model. Our simulation models take our best approximations of this behavior and replicate it.

As evidenced above, simulations can be a very valuable tool to researchers. It is also important to note that there are cases where a simulation would not be

appropriate. Banks and Gibson [8] list ten rules for when simulations should not be used. A collection of the more meaningful ones for our purposes are summarized below:

1. Simulations should not be used when common sense can solve the problem or when the problem can be solved analytically in reasonable time.

2. Simulations should not be used when the cost of developing the simulation model is more than the cost of experimentation.

3. Simulations are not useful when system behavior is too complex or unknown.

2.2.1 Advantages and Disadvantages

There are many advantages to using simulations for scientific study [10]. Aside from the fact that simulations allow one to model the behavior of a real-world system without harming or altering the real-world system, simulations typically run and produce results faster than the real-world system being studied, if such a system exists. Additionally, simulations are useful in testing the influence of different variables both on the system as a whole and in regard to one another. Further, a simulation is helpful when performing hypothesis tests or when testing situations that would be unrealistic or impractical in the real-world.

Simulation studies also have some inherent disadvantages. Banks, et al. [10] list four specific disadvantages. Namely, they list that simulation models are difficult both to build and to interpret. This is true to an extent, but experienced programmers will find model building straightforward. In addition, interpreting and analyzing the results of a simulation may take some time, but, in many cases, this amount of time will be less than if the scientist had done the actual real-world

experiment. Although there are some valid disadvantages to using a simulation, as long as the simulation is used to model a system that simulations are advantageous for (some requirements of which we have listed above), building a simulation can be very useful to scientists.

2.2.2 Building a Simulation Model

We have already described simulations as being built upon a model. In most cases, scientists start out with a conceptual model, or a model with which they intend to accurately represent the system they are studying. This conceptual model typically goes through many phases and revisions as the simulation is being built. Oftentimes, scientists will recognize a problem with their conceptual model or a way to improve it and then implement the change. This technique is part of the aforementioned scientific method. Once the scientist has sufficient confidence in the conceptual model, it is often referred to as simply the model. At this point, the model is the representation of the system the scientists are studying. This representation is simply for the study of the system through simulation. Accurately representing a model that exactly matches a real-world phenomenon is extremely difficult, if not impossible.

There is often an inherent randomness present in simulation models. There are many reasons for this randomness, the main one of which is that real-world systems are far too complex to accurately represent through a computer simulation model. First, we include randomness to cover our ignorance or uncertainty. In many of the systems we model, we have little idea about the underlying mechanics. We build simulation models to try to help us to understand these characteristics and to experiment with them. If done properly, we will learn about real-world sys-

tems through our simulation models. Second, randomness is included for decision making. If we have a simulation that models ants foraging for food, we have to give the ants the ability to make decisions. If the simulation had them do the same thing every time, nothing would be learned from the simulation after the first run, as it would behave the same each time. Randomness is included so that we can mimic the real-world in the best way we know how. Lastly, measurement error or quantum effects are accounted for by randomness. Simulations cannot have the precision of real-world systems because of both the limitations of computers and of our own knowledge. They also cannot represent entities as accurately as a real-world system, so we include an inherent randomness. These examples are not meant to be looked at as limitations of simulation models, but as reasons why simulation models are created the way they are. In fact, this randomness is part of what makes simulations unique and powerful.

2.2.2.1 Simulation Model Types

The literature [10, 50] has divided simulation models into the following overlapping subcategories:

Static vs. Dynamic Static simulation models are representative of a system at a specific time. An example of a static system is one that solves complex analytical problems that are infeasible with other methods. Dynamic simulation models are representative of a system over time, such as population dynamics.

Deterministic vs. Stochastic Deterministic simulation models produce results determined by the provided inputs. In such simulation models, probability does not play a role. An example would be a simulation that models a

student going to class at a specified time every day. Stochastic simulation models involve random variables and produce different results with each random seed. Our model with the student would be stochastic if we add a certain probability as to when and whether the student will arrive to class.

Continuous vs. Discrete Continuous simulation models characterize systems continuously over time. An example would be the population dynamics in a predator-prey simulation model. Discrete simulation models characterize systems at specific points in time. An example would be people paying tolls at a toll booth.

For the purposes of this study, we further classify simulation models into the following subcategory:

Agent-based vs. Equation-based Agent-based simulation models have individual entities, called agents that drive the simulation. They are good at modeling systems with emergent properties. Equation-based simulation models are equation driven and are adept at modeling mathematically based phenomena. We next elaborate on these subcategories.

2.2.3 Agent-based Simulations

Agent-based simulations, also known as individual-based simulations, are built in a bottom-up approach. They have recently gained popularity [71] and are proving to be very powerful. In an agent-based simulation, an agent can be thought of as any acting component in the system. Each agent is treated as an entity, having its own properties and behaviors. These can be influenced by the environment and by other agents, among others. The interactions between agents and their environment over time often lead to emergent properties within the

system. Time is typically represented in the form of time steps; namely, each agent usually has a chance to change its properties and interact with other agents and the environment once every time step. An advantage of agent-based simulations is that they are easily extensible. Adding agents to the model with different inherent properties is straightforward. Additionally, agent-based simulations are rather intuitive to code, as they are modeled in the same manner that we tend to think about systems.

2.2.4 Equation-based Simulations

Equation-based simulations are built in a top-down approach. They have been around considerably longer than agent-based simulations and are therefore much more mature. A difference between equation-based simulations and agent-based simulations is that equation-based simulations do not tend to lead to emergent system properties as often as agent-based simulations. For example, equation-based simulations are good at modeling systems governed by underlying mathematical properties or formulas. This is somewhat of a limitation, as more complex systems that cannot be approximated by equations are tough to build. Also, changing overall properties of an equation-based simulation is often difficult, as it may require a new mathematical model; however, tweaking parameters in an equation-based simulation is relatively simple. In this respect, equation-based simulations are rather simple and straightforward. In general, equation-based simulations are very good at modeling known systems with aggregate behaviors or systems simply governed by mathematical rules.

2.3 Verification and Validation Process

Simulations are most useful when they realistically estimate or emulate a phenomenon; deciding when this is the case is not trivial. To get the most meaningful information from a simulation, a simulation must first be verified and validated. This means that the data produced by the simulation is to some degree indiscernible from real-world data. More specifically, verification of a simulation model refers to solving the model right, as in the abstract model correctly matching the chosen phenomenon. This is typically achieved through comparison studies. Validation is solving the problem right, meaning getting the right model by choosing an abstract model that accurately represents the phenomenon [10]. This is typically achieved through iterative calibration of the model. The extent to which verification and validation techniques are performed is dependent on several factors, as noted by Sargent and shown in Figure 2.1 [75]. The most important factor is usually the cost, which is very high if the intended model confidence is high. Typically, developers choose the most cost-effective way to achieve the highest model confidence.

The verification and validation process is a critical part of the model development process - the best simulation results are achieved when verification and validation techniques are performed as the model is being built. Banks, et al. [10] have verification and validation as integral members of their overall process to performing a simulation study, as shown in Figure 2.2. Applying verification and validation techniques in tandem to a model will help develop a more valuable model. Sargent [75] mentions that the typical verification and validation process begins when the simulation model development begins. As is usually the case, the general process starts with identification of the research question at issue. From

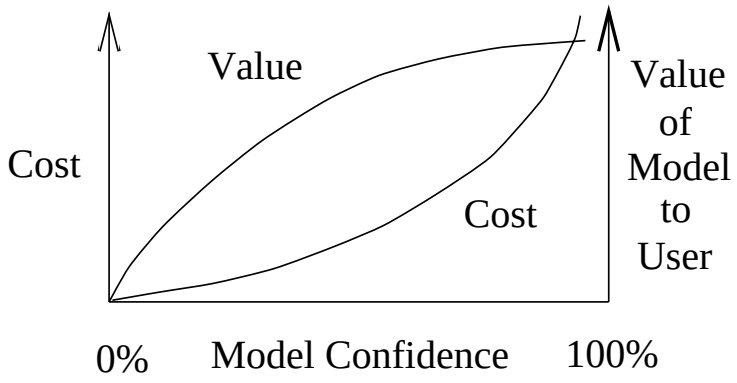


Figure 2.1. Model Confidence and Cost Trade-offs (adapted from [75]). An approach to determine the amount of verification and validation to perform on a simulation. The trade-off between cost and value must be examined. Typically, researchers choose the most cost-effective technique.

there, an initial conceptual model can be developed, which, through programming, eventually leads to the computerized simulation model. If the process is successful, the simulation will appropriately answer the initial research question. Verification and validation should be performed at each of these steps. For example, the developer should not move from the conceptual model to the computerized simulation unless the conceptual model has been properly validated. As the process progresses, additional verification and validation takes place. An adapted version of Sargent's [75] and Huang's [39] verification and validation process model is shown in Figure 2.3. We have adapted it to fit agent-based scientific and equation-based economic simulations. It is important to note that the relationships shown in Figure 2.3 are all two-way, showing the importance of feedback among stages.

We mentioned before that simulation is part of the scientific method. Along the same lines, the validation of a simulation is analogous to the validation of a scientific theory [59]. Much work has been performed regarding verification

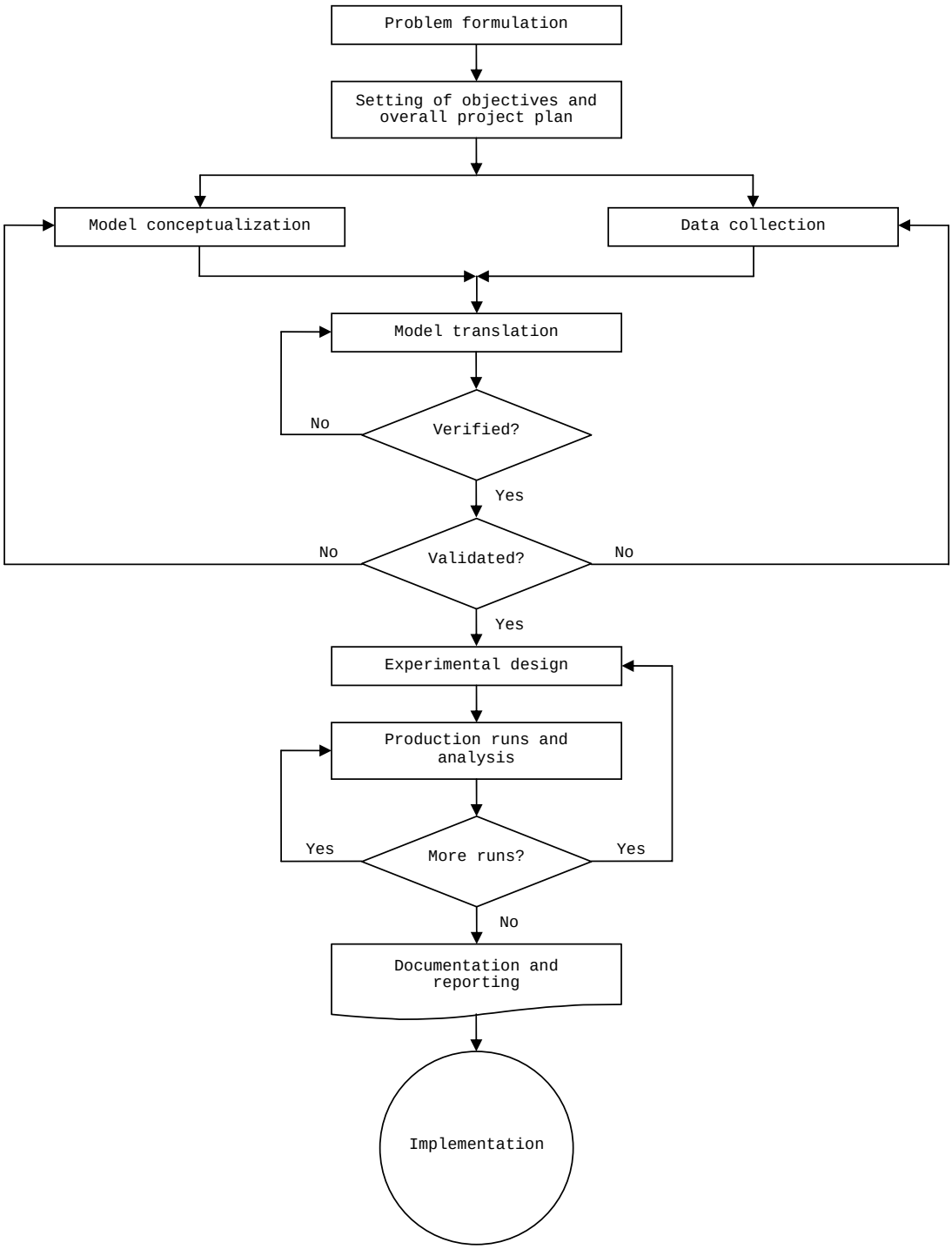


Figure 2.2. Typical Steps in a Simulation Process (adapted from [10]). Notice how verification and validation play integral roles in the process and that the process does not enter the final stages until it has been validated.

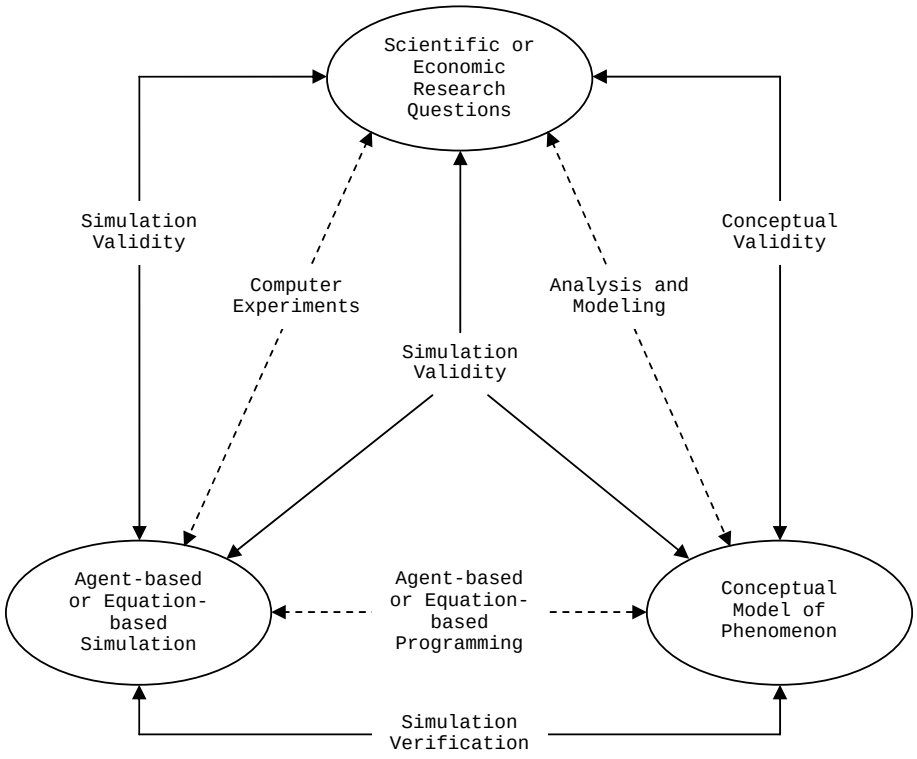


Figure 2.3. A Verification and Validation Process for Scientific and Economic Simulations (adapted from [39, 75]). Notice how the stages are all interconnected and that the process is cyclical.

and validation of simulation models [4, 6, 47, 48, 63]. However, most of these simulations have been for equation-based or discrete-event simulations [27, 34, 67]. Because verification and validation is newer to the agent-based and social science simulation fields, the work done in this area is less mature. For example, a recent National Science Foundation Blue Ribbon Panel had the following to say regarding verification and validation:

> New theory and methods are needed for handling stochastic models and for developing meaningful and efficient approaches to the quantification of uncertainties. As they stand now, verification, validation, and uncertainty quantification are challenging and necessary research areas that must be actively pursued [62].

While verification and validation is very important, some researchers, such as Box, go as far as saying that all models are wrong, with only some being useful [16]. Fortunately, there are many techniques historically used in mature disciplines that can be adapted to fit the needs of these new modeling techniques, bringing more confidence to their results and making them more useful. Most notably, Balci [5, 7] and Sargent [74–76] have outlined techniques to use to perform verification and validation. We next describe some general techniques and their applications. It is important to note that no single approach can be applied to all models because the purposes of simulations are so varied. Instead, combinations of techniques are used to increase model confidence.

2.3.1 Verification and Validation Techniques

We next outline a general verification and validation process, based on the use of subjective and quantitative techniques. This process is one we developed; it utilizes many techniques found in the literature [6, 9, 76].

2.3.1.1 Subjective Techniques

Although subjective validation techniques can be formalized, they are typically used for initial quick-and-dirty validation. Because they largely rely on the judgment of domain experts, they often require less rigorous effort than quantitative validation techniques. They are most useful in detecting early flaws in the simulation model and are most often used with exploratory simulation studies. Subjective validation techniques have only recently been applied to economic and agent-based scientific simulations, even though they have been applied to typical industrial engineering purposes for many years. Balci [5] describes many such techniques; we next describe several subjective techniques, adapted from Xiang et al. [87] and Kennedy et al. [44, 46].

Black-Box Testing This technique treats the simulation model as a black-box, such that it is fed a set of inputs and produces a set of outputs. Here, we are concerned with how accurately the input is transformed to the system output.

Face Validation Face validation is a preliminary technique that is performed by asking domain experts whether the simulation model looks to be behaving in a sufficiently accurate manner. This is usually a rather informal technique, as the domain expert usually only analyzes the output to the simulation model and determines whether it is reasonable. If a visualization [32] is available, the domain expert will observe it as the simulation is running.

Internal Validity This technique is primarily concerned with the variability of the simulation model across different simulation runs. In most cases, there should not be a large amount of variability across different simulation runs.

Improper use of random number generators are a common cause for variability. Ideally, the chosen random seed for the simulation model should not introduce a bias. If it does, the validity of some aspect of the model, likely how the random number generator is used, is questionable.

Tracing Tracing involves following an entity of the system as the simulation runs. It is useful because the behavior of the entity can be traced as the simulation runs, helping to identify any anomalies in its behavior or possibly in the logic of the model.

Turing Test A Turing test involves providing domain experts with outputs from the simulation model and outputs from a real-world (or a validated) system. The domain experts are then asked if they can discriminate between the outputs.

2.3.1.2 Quantitative Techniques

Quantitative validation techniques are usually more formalized than subjective validation techniques. These statistical techniques help to increase the confidence of a model through comparisons of the simulation model output with the real-world system, with expected output, and possibly with output data of other experiments or simulation models that model the same phenomenon. Applying quantitative validation techniques to a simulation model starts by identifying the appropriate output measures [87]. Once these have been collected, they can be used for subjective validation techniques, such as face validation. They can also be analyzed through statistical tests, such as the Chi-Squared test or the Kolmorgorov-Smirnov test. All of these techniques help the simulation model developers determine if their simulation model is sufficiently accurate. Balci [5]

describes many quantitative techniques, a subset of which that are relevant to our work are next described.

Docking Docking, also referred to as model-to-model comparison or alignment, is a technique that can be used when another simulation model exists that models the same phenomenon or when one can be easily created [3]. The idea here is that the confidence in the simulation model is greatly increased when independent simulation models produce the same effective results. This technique helps to determine whether two or more models can produce the same results and is particularly valuable when the simulation models were written independently and in different programming languages. A simulation model can also be validated against real-world data to increase confidence.

Historical Data Validation This technique is used when historical data exists for a given model. This data can then be used to both build the simulation model and to determine if it behaves like the historical data.

Predictive Validation Predictive validation involves taking data from an operational system or from field or laboratory experiments and then comparing it with the behavior predicted by the simulation model.

Sensitivity Analysis and Parameter Variability Sensitivity analysis and parameter variability involve changing both the input variables and internal parameters and then evaluating the simulation model's behavior and output. If certain parameters are too sensitive and result in major changes in the simulation across different values, the accuracy of the simulation model should be examined. As always, the simulation model should adequately represent the real-world system. In this case, the results of changing specific

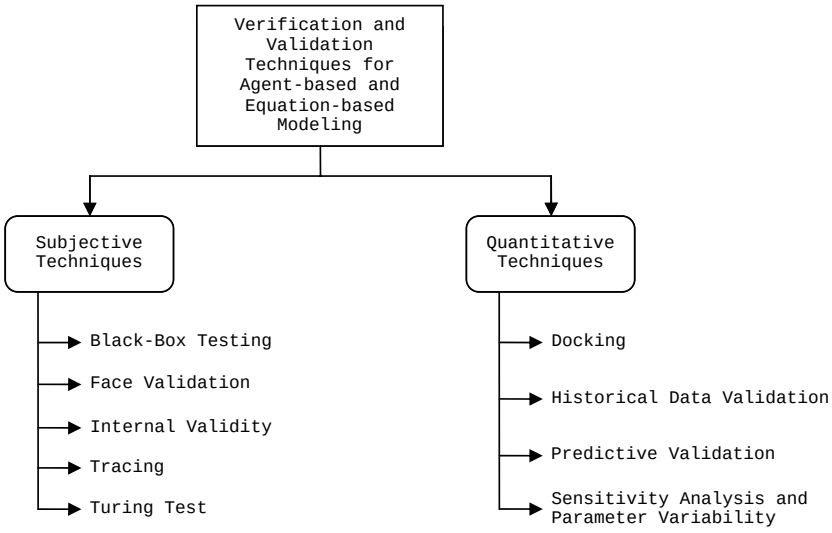


Figure 2.4. Verification and Validation Techniques. Here, we have shown a selection of the techniques we employed for our case studies. Some techniques can be categorized as both subjective and quantitative, depending on the context.

input variables or internal parameters should be the same as if they were changed in the real-world model.

A hierarchy of the techniques we have described is shown schematically in Figure 2.4.

2.4 Case Study I: An Agent-based Scientific Model

Natural organic matter, or NOM, is a heterogeneous mixture of molecules. The evolution of NOM is an important research topic in many disciplines due to its role in the evolution of soils, the transport of pollutants, and its role in the general carbon cycle [19, 20, 46, 86]. NOM is typically made up of molecules with varying properties, such as molecular weight or reactivity level. This makes NOM rather complex, making it difficult to both study in the field and to simulate.

Specifically, studying NOM in the field is difficult because of its inherent properties and location, but also because we are still relatively unfamiliar with its inner-workings. One of the main limitations when studying NOM is that it is very time-consuming. A simulation model for NOM would be very valuable to scientists and also significant in terms of the simulation discipline. We next describe our conceptual model for NOM.

2.4.1 Conceptual Model

Our conceptual model is based on the research of chemist Stephen E. Cabaniss of the University of New Mexico [19, 20]. He based his model on extensive laboratory work and observation, as well as detailed knowledge from the literature. His work resulted in a program, called AlphaStep [1], that simulates NOM, the details of which will be discussed later. The basic model includes the use of precursor molecules such as cellulose, lignin, and protein in a controlled environment. This environment is largely controllable; variables such as temperature, pH, light intensity, and bacterial density are all modifiable. We next describe the components of our NOM Model in more detail, as adapted from Xiang et al. [87] and Kennedy et al. [44, 46].

Agents The agents in the model are molecules. Each molecule has its own properties, such as location, reaction probability, and elemental formula. The elemental formula is based on the number of C, H, O, N, S, and P atoms present and this gives rise to the molecule's molecular weight. The number of functional groups within a molecule is also included.

Behavior Agents in the model move about on a 2D grid. Depending on the simulation type, agents have have the ability to adsorp to the surface or

to react with other agents. Further discussion of the simulation types can be found in Appendix A.1. Each molecule has its own reaction probability, giving rise to chemical reactions. These reactions can change the structure of the molecule, result in new molecules, or result in the molecule disappearing from the system. In all, there are twelve types of chemical reactions that the molecules can undergo. These first- and second-order reactions are described in Table 2.1. We next describe the type of reactions that the molecules can undergo.

First-order reactions with a split– The reaction of predecessor molecule A results in two successor molecules B and C. Molecule B replaces molecule A in the system, while molecule C is created in the empty cell closest to molecule B.

First-order reactions without a split– The reaction of predecessor molecule A changes its structure.

First-order reactions with the disappearance of a molecule– Predecessor molecule A undergoes a reaction and is removed from the system.

Second-order reactions– Two molecules A and B react to form a new molecule C. Molecule C replaces molecule A in the system, while molecule B is removed.

Space The agents move about on a 2D geometrical space, represented as a grid. Each grid cell can hold multiple molecules, up to a specified threshold. Depending on the simulation type, the space may be open, meaning molecules enter through the top and leave through the bottom, or closed, meaning the space is bounded.

TABLE 2.1

CHEMICAL REACTIONS

Name	**Type**
Alcohol C-O-H Oxidation	First-order without a Split
Aldehyde C=O Oxidation	First-order without a Split
Aldol Condensation	Second-order
Amide Hydrolysis	First-order with a Split
Decarboxylation	First-order without a Split
Dehydration	First-order with a Split
Ester Condensation	Second-order
Ester Hydrolysis	First-order with a Split
Hydration	First-order without a Split
Microbial Uptake	First-order with the disappearance of a Molecule
Mild C=C Oxidation	First-order without a Split
Strong C=C Oxidation	First-order with a Split 50% of the time

Reaction Probabilities Each molecule has an associated reaction probability. This probability is continually changing and is based on both intrinsic and extrinsic factors. The intrinsic factors are related to molecular structure, such as the number of a specific functional group present. The extrinsic factors are more related to the environment and rely on factors such as light intensity, concentration of inorganic chemical species, availability of surfaces, presence of extracellular enzymes and microorganisms, and the presence and reactivity of other NOM molecules. We use probabilistic functions to combine these factors and to generate individual probabilities.

Molecular Properties Molecular properties are calculated from the elemental composition of a given molecule. They also give rise to functional group counts and reactivity levels. These properties were useful in helping us calibrate the conceptual model.

Simulation Process The simulation models the evolution of NOM over time. Specifically, it is a stochastic synthesis model. This means that the system is represented by a set of values with a certain probability distribution, making the evolution of the system dependent on a series of probabilistic discrete events. The system uses a uniformly distributed random number generator that generates a number for each molecule that influences the behavior of the molecule, such as whether it will undergo a reaction. This process takes place at each time step. If new molecules are created, they are added to the process.

2.4.2 Implementations

Stephen E. Cabaniss implemented his model in Pascal and released it as a Windows program called AlphaStep [1]. We coded our model, the NOM Model, with Java [41], J2EE[40], and the Repast toolkit [69]. Java was chosen because of its compatibility both with Repast and with our online deployment of a web-based interface for the NOM Model. Repast is a toolkit that facilitates agent-based modeling and provides rich visualization capabilities. The four different simulation types for the NOM Model that we implemented are discussed in Appendix A.1. AlphaStep is a close match to our NOM *No-flow Reaction* Model.

There are a few differences between AlphaStep and the NOM *No-flow Reaction* Model. The biggest difference is that there is no concept of space in the AlphaStep model. Instead, the molecules are represented as members of a list. If there were space in AlphaStep, the representation would be that all the molecules are on the same cell, each equally likely to react with any other molecule. A summary of this and all relevant differences are listed in Table 2.2.

2.4.3 Validation

The validation techniques presented previously were followed when performing verification and validation on the NOM Model and are presented next.

2.4.3.1 Subjective Analysis

We started our verification and validation study with the help of domain experts. The first goal was to obtain face validation of the model. This was accomplished by having the domain experts analyze the underlying mechanisms of our conceptual model. Once we passed this test, we coded the agent-based simulation

TABLE 2.2

IMPLEMENTATION DIFFERENCES

Feature	**AlphaStep**	**NOM *No-flow Reaction* Model**
Animation	No	Yes
First Order Reaction with a Split	Add to Molecule List	Choose Nearest Neighbor
Initial Population	Actual Number of Molecules	Percentage Distribution of Molecules
Platform	Windows	Linux
Programming Language	Pascal	Java
Running Mode	Standalone	Web-based or Standalone
Second Order Reaction	Randomly choose from the Molecule List	Choose nearest Neighbor
Simulation Toolkit	None	Repast
Spatial Representation	None	2D Grid

model. As we were coding the model, we performed numerous subjective analysis techniques, most notably tracing, code walk-through, boundary testing, and input-output testing. These all helped us verify the accuracy of our simulation model.

The most useful technique we used was visualization. We displayed a visualization of the molecules and their interactions as they moved through the system, as shown in Figure 2.5. The left portion of the image shows the molecules, which are color-coded according to molecular weight. Adsorbed molecules (molecules stuck to the surface) are shown as solid circles and desorbed molecules (free-moving) are hollow circles. This screen shot is from the NOM *Flow Sorption* Model, in which the molecules can adsorp or desorp as they flow through the system from the top to the bottom. Domain experts were able to look at this part of the visualization and determine whether system dynamics looked appropriate as the simulation ran. The right portion of the image shows a real-time graph for the molecular weight distribution of the system. This is shown for both adsorbed and desorbed molecules. The peaks in the graphs have been observed in laboratory [2] to gradually move from right to left as the simulation progresses. By observing our NOM *Flow Sorption* Model, the domain experts were able to verify this expected behavior in the model, greatly increasing confidence.

2.4.3.2 Quantitative Analysis

The most effective quantitative method we used for this case study was docking. The process began with us taking a good look at the AlphaStep simulation model and its underlying mechanisms. We then compared output of the two simulations for similar starting states. Initially, the simulations produced very different

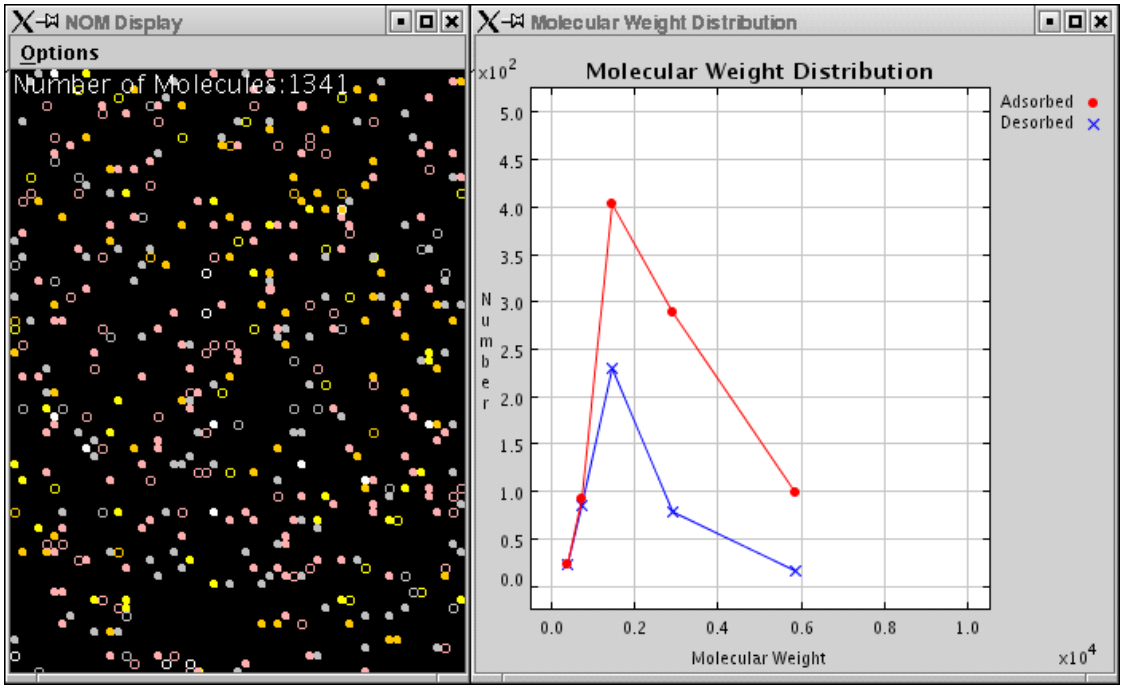
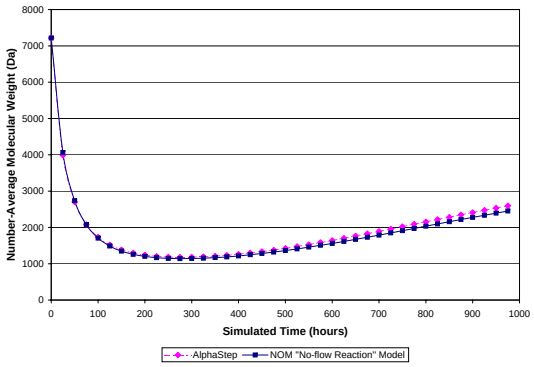


Figure 2.5. NOM *Flow Sorption* Model Visualization. These shots are for the NOM *Flow Sorption* Model. Solid circles on the left portion indicate adsorbed molecules and hollow ones indicate desorbed molecules. In this system, molecules enter at the top and flow through and out the bottom. The colors of the molecules are dependent on their molecular weight. The graph on the right portion tracks molecular weight distribution for adsorbed and desorbed molecules in real-time. This visualization helps obtain face validity.

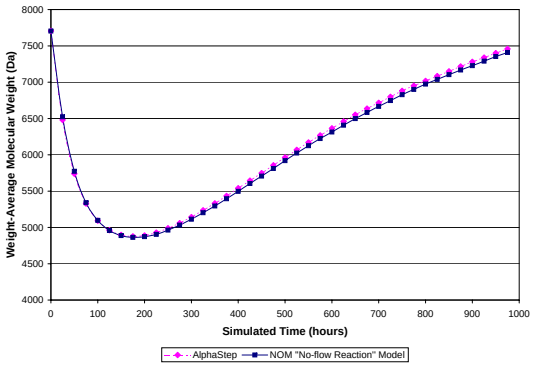
results. This caused us to further examine our model, as the AlphaStep model had been calibrated with some laboratory data. We discovered an error with our random number generator and several problems with our probability calculations. Additionally, some of our molecules were being represented incorrectly. Along the way, we continued to run experiments to see if the results had been docked. This proved to be a difficult task, but we soon identified all problems and were very pleased with the results.

To do the formal docking of the NOM Model against AlphaStep, we performed fifty replications of the same effective experiment in each model, with the only difference being the random seed. We traced many of the variables through each of the replications and then took ensemble averages and plotted the results. Figure 2.6 shows this for M_n (number-average molecular weight), M_w (weight-average molecular weight), weight percentage of Carbon, total mass of Carbon, and the total number of the molecules in the system. These graphs increased the confidence in both models, particularly because the underlying mechanisms are so different.

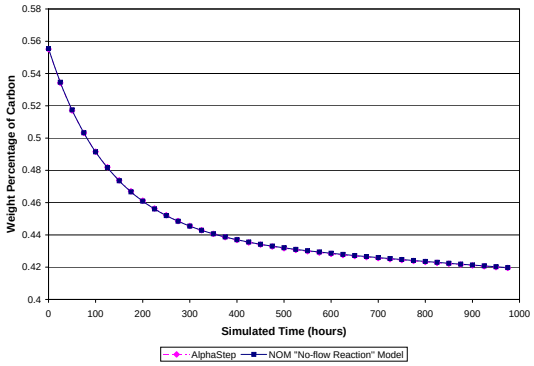
We also wanted to make sure that the NOM Model had sufficient internal validity. Our initial efforts helped us determine a problem in our use of random number generators. Figure 2.7 shows the total number of molecules in the system at the end of twenty-five independent runs before and after fixing the problem. The only difference between the runs is the random seed used. Another way we tested for internal validity was to look at the total number of molecules in the system after 1000 simulated hours. We replicated this experiment 450 times, with the only difference being the random seed. Figure 2.8 shows the distribution we obtained. The Chi-Squared test shows that, when the data is compared to the



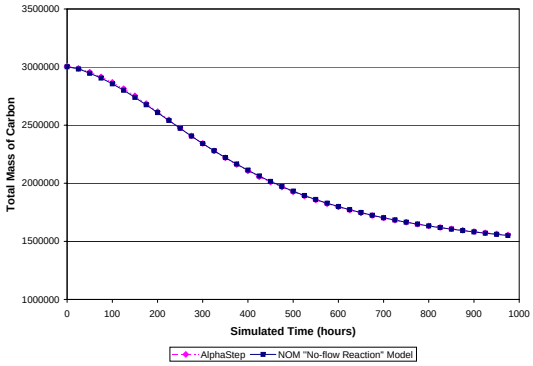
(a) Number-Average Molecular Weight



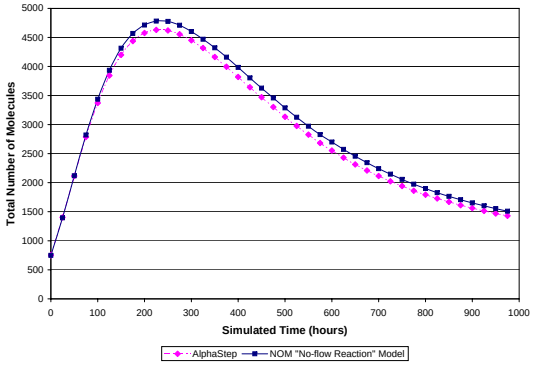
(b) Weight-Average Molecular Weight



(c) Weight Percentage of Carbon



(d) Total Mass of Carbon



(e) Total Number of Molecules

Figure 2.6. Ensemble averages compassion between AlphaStep and the NOM *No-flow Reaction* Model. Each graph represents ensemble averages at forty points across fifty different simulation runs. Panels (a) - (e) show different variables across the same data set.

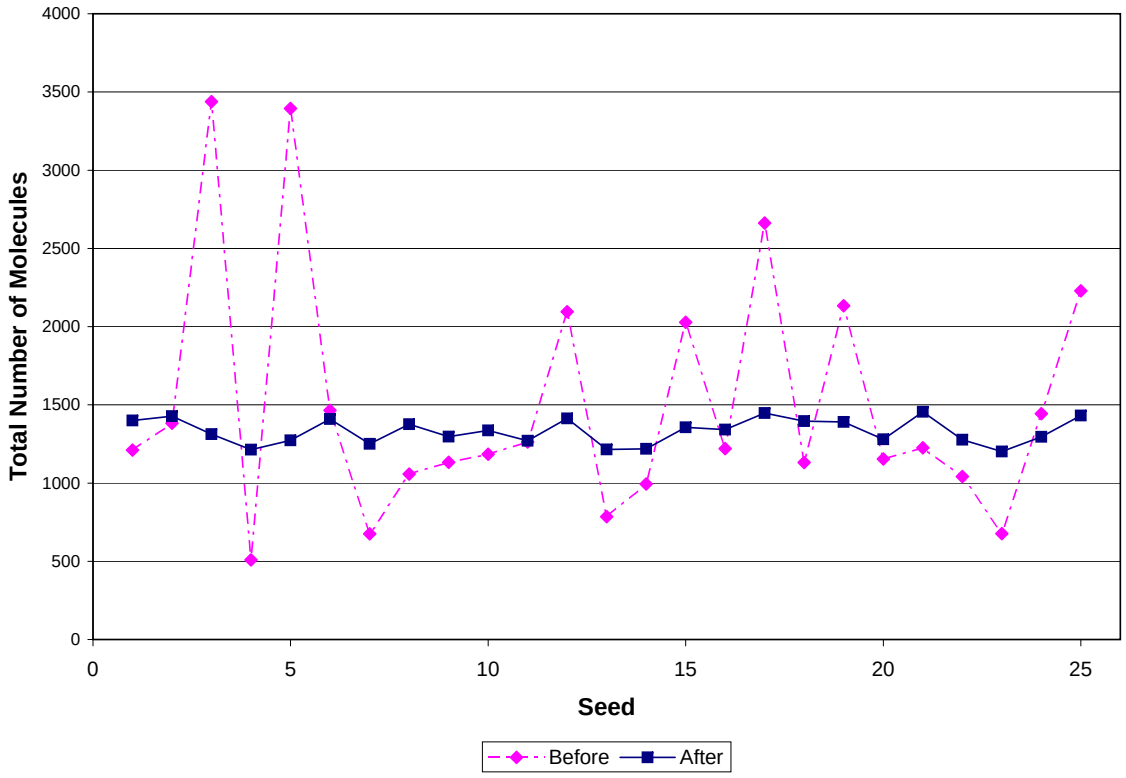


Figure 2.7. Random Number Generator Problem. Face validity tests helped us determine a problem with our use of random number generators. As shown here, we initially had a very scattered distribution. After correction of our problem, the distribution is much more stable.

hypothesis of normality, the variation observed in the data is not statistically significant.

2.4.4 Discussion

This case study has demonstrated the usefulness of performing verification and validation techniques on an agent-based model, specifically the use of docking and visualization. When feasible, docking two very different simulation models

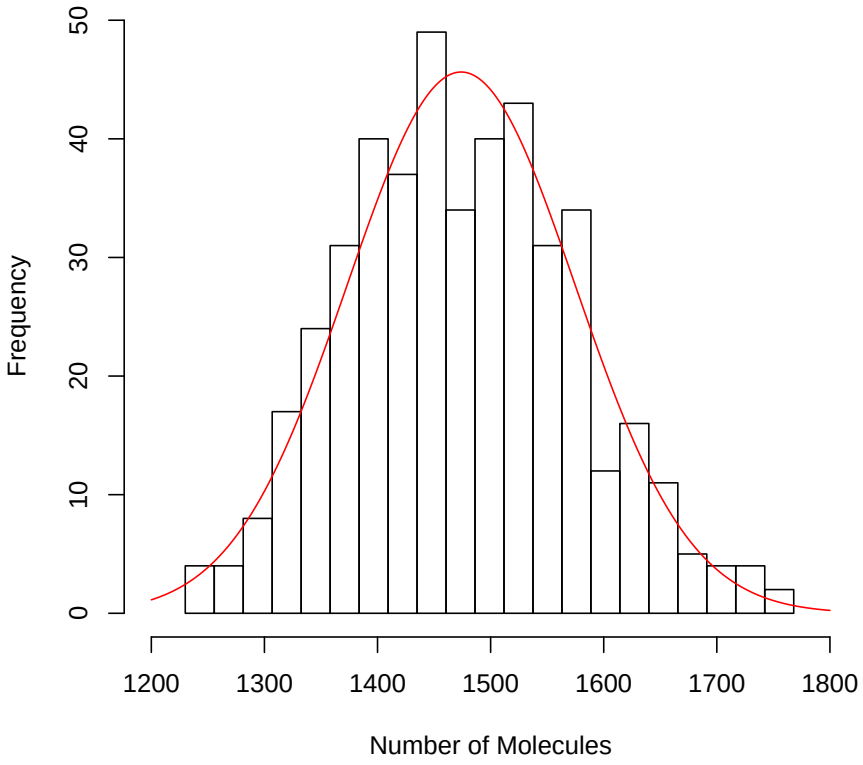


Figure 2.8. Histogram representing the total number of molecules in the system after 1000 simulated hours. The distribution is a normal distribution, as supported by the Chi-Squared test. The data is over 450 simulation runs, each with a different random seed.

and showing that they produce the same effective results greatly adds to the confidence of both models. Visualization can also aid this effort. This case study has emphasized some of the techniques we used and outlined how verification and validation can be applied to agent-based simulations.

2.5 Case Study II: An Equation-based Economic Model

Our economic model simulates Ramsey problems. Ramsey problems are concerned with setting specific economic variables, money growth and tax rate, to obtain the best social welfare for an economy [68].

2.5.1 Conceptual Model

Cosimano and Gapen have created a simulation model that uses nonlinear projection methods to solve Ramsey problems by calculating the real or nominal interest rate under optimal money growth and tax rates [25]. This is accomplished through residual equations that utilize bivariate Chebyshev polynomials. At the core of the model, are four equations, shown in Equations 2.1 - 2.4, representing money growth, tax rate, labor, and the LaGrange multiplier. These policy function equations are later transformed with Chebyshev polynomials for use in our simulation model. Additionally, we utilize Gauss-Hermite quadratures to evaluate the conditional expectations. We adapted and modified this model for our second case study.

$$\widehat{\mu}_{t+1}(\theta_t, g_t, b) = \sum_{i=1}^{n_\theta} \sum_{j=1}^{n_g} b_{ij} \Psi_{ij}(\theta_t, g_t) \tag{2.1}$$

$$\widehat{\tau}_t(\theta_t, g_t, d) = \sum_{i=1}^{n_\theta} \sum_{j=1}^{n_g} d_{ij} \Omega_{ij}(\theta_t, g_t) \tag{2.2}$$

$$\widehat{H}_t(\theta_t, g_t, q) = \sum_{i=1}^{n_\theta} \sum_{j=1}^{n_g} q_{ij} \Phi_{ij}(\theta_t, g_t) \tag{2.3}$$

$$\widehat{\lambda}_{gt}(\theta_t, g_t, v) = \sum_{i=1}^{n_\theta} \sum_{j=1}^{n_g} v_{ij} \Gamma_{ij}(\theta_t, g_t) \tag{2.4}$$

2.5.2 Implementations

The model was initially written in Matlab [53], but we ported it to C++ [18]. The model was verified and validated in Matlab before it was converted to C++. Both implementations rely on the same core equations and use the same methods, but the mechanisms for doing so are very different. The main reason for this is that Matlab is a much higher-level language than C++ and that with the desired added complexity of a fifth equation, representing capital, execution in Matlab would be infeasible. This, along with the opportunity for an additional verification and validation study, motivated our decision to port the code.

The internal mechanisms used in Matlab and in the C++ versions are very difficult. Matlab has many built in helper functions, most notably the LAPACK [49], or linear algebra package, that are easy to use for our purposes. C++ does not have many of these functions inherently built in, so we utilized the standard template library as well as the GNU Scientific Library, GSL [33]. Additionally, these helper functions had to be heavily modified for use in the C++ version because the Matlab versions are all overloaded in the sense that you can pass them different data types, such as a vector or a matrix. This correlates to the

fact that variables are not declared in Matlab like they are in C++, meaning where you can call or use a variable in Matlab without specifically noting the data type, C++ requires you to explicitly handle the data type, be it by overloading or explicitly noting it. These differences strengthened the results of our verification and validation study. Sample code for both the C++ and Matlab implementations can be found in Appendix A.2.

2.5.2.1 Performance

It is important to note the significant speedup obtained by converting the model from Matlab to C++. Table 2.3 and Figure 2.9 show the running time for the simulation over five, fifty, and five-hundred iterations.

2.5.3 Validation

The same validation techniques we utilized for the first case study were applied to this case study. In this case, the techniques helped us identify initial problems with the porting of our code.

TABLE 2.3

RUNNING TIME FOR MATLAB AND C++ IMPLEMENTATIONS

	5 Iterations	**50 Iterations**	**500 Iterations**
Matlab	58 seconds	568 seconds	8872 seconds
C++	2 seconds	17 seconds	176 seconds

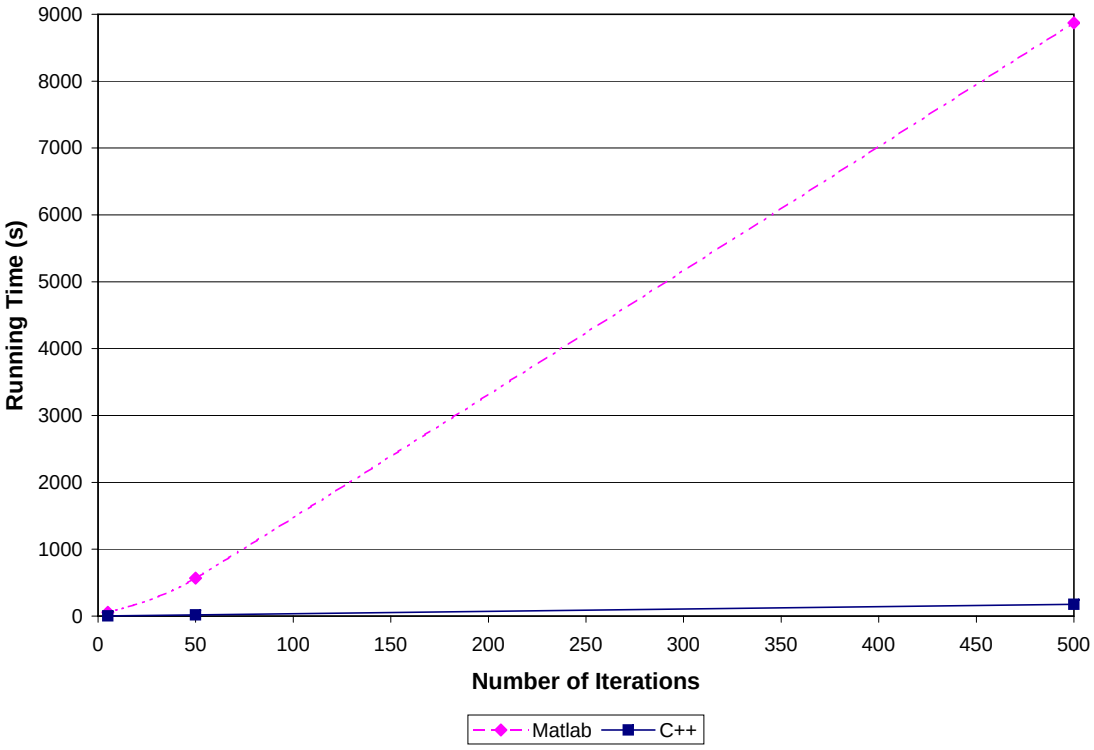


Figure 2.9. Performance Comparison. Here, we show the vast speedup (about thirty times faster) for the model with the C++ implementation.

TABLE 2.4

FACE VERIFICATION

	Matlab	**C++**	**Steady State**
LaGrange Multiplier	0.138	0.138	0.138
Labor	0.309	0.309	0.309
Money Growth	-0.009	-0.009	-0.009
Tax Rate	0.188	0.188	0.188
Cash Good	0.486	0.486	0.485
Credit Good	0.621	0.621	0.620
Real Interest Rate	0.009	0.009	0.009

2.5.3.1 Subjective Analysis

Subjective validation techniques were performed throughout the code porting process. These techniques were especially helpful in porting the code, as we used techniques such as tracing to make sure entities, such as the LaGrange multiplier, behaved the same in both models. The checking of the control flow helped us validate the new C++ model against the Matlab model. It also helped us determine where our code was going wrong. Cosimano and Gapen [24] present steady state data for the model that we used to validate our model against. This data, as well as output data for both implementations is shown in Table 2.4. We presented this model to domain experts, helping us to achieve face validity for the models. Since the core of the simulation is equation-based, meaning the output values should be consistent across implementations, we describe our technique as more of a "face verification" technique in discerning the accuracy of our models.

2.5.3.2 Quantitative Analysis

This study is unique in that the output values are given according to known equations. This lead to us performing output checks and tracing entities as they progressed through the simulation model. Essentially, we validated the C++ version of the code against the Matlab version, which had previously been validated itself. Additionally, we validated both the C++ and the Matlab versions against the steady state data from Cosimano and Gapen [24]. These techniques allowed us to gain more confidence in our simulation model, while also initially helping us identify some bugs.

2.5.4 Discussion

This case study allowed us to demonstrate the feasibility and usefulness of docking as a verification and validation technique. Typically, rewriting another simulation in another language is only done if the model is relatively simple or if something else is gained in the process. Here, we were motivated by the significant speed advantage of C++, as well as the ability to perform a more rigorous verification and validation effort.

2.6 Conclusion

Through two very different case studies, this chapter has shown how verification and validation techniques can be applied to simulation models. We specifically looked at docking and visualization and showed their effectiveness. We have shown that similar verification and validation techniques can be applied to models, regardless of the underlying model structure.

In general, it is important to have a concise abstract representation of the

model in mind when designing a simulation. This representation will lead to effective implementation decisions. As decisions are made, the entire lifetime of the model should be considered. Additionally, verification and validation techniques should be considered. Some general ratings for the techniques we used in our case studies are shown in Table 2.5. Possible ratings are Fair, Good, Very Good, and Excellent. It is important to note that the ratings shown here are specific to our case studies and may vary in other instances. To conclude, the techniques we used in our case studies were the ones that gave us the most model confidence and value for the lowest cost, the relationship of which was shown in Figure 2.1.

TABLE 2.5

GENERAL RATINGS FOR OUR CASE STUDIES

	Agent-based (Stochastic)		**Equation-based (Deterministic)**	
	Cost	**Effectiveness**	**Cost**	**Effectiveness**
Face Validation and Face Verification	Low	Very Good	Low	Very Good
Turing Test	Low	Very Good	Low	Good
Internal Validity	Moderate	Very Good	n/a	n/a
Tracing	Moderate	Fair	Moderate	Excellent
Black-Box Testing	Low	Good	Low	Very Good
Docking	Moderate	Very Good	Moderate	Good
Historical Data Verification	Moderate	Very Good	Moderate	Very Good
Sensitivity Analysis and Parameter Variability	Moderate	Good	Moderate	Good
Prediction Validation	Moderate	Good	Moderate	Fair

CHAPTER 3

BIOINFORMATICS COMPUTING: IDENTIFYING TRANSPOSABLE ELEMENTS IN THE *Aedes aegypti* GENOME[1,2]

3.1 Introduction

We begin this chapter with a brief introduction to the biological concepts that are necessary to understand as we start discussing the field of bioinformatics and the goals of this chapter. After a short discussion of some of the research areas, tools, and technologies often used in bioinformatics, we introduce the *Aedes aegypti* mosquito and transposable elements. We then proceed to explain the relevance of finding transposable elements and how they help as tools in biological research, namely evolutionary biology. We explain a basic approach to finding transposable elements and then offer two approaches we developed when searching for transposable elements in the *Aedes aegypti* genome. We explain the advantages and disadvantages of all approaches and offer a proposed hybrid method. We conclude the chapter with a discussion of our results.

[1]The primary focus for this study was the computational process aspect. Frank H. Collins, PhD, and James R. Hogan served as the primary biological investigators.

[2]Portions of this chapter have appeared in technical reports [22][45].

3.2 Biological Foundations

All life revolves around tiny entities called cells. These cells serve varying purposes, but they each have the ability to replicate; within each cell, there is enough information for the cell to make a complete copy of itself. Moreover, each cell also contains the mechanisms necessary to perform the replication [42]. Jones and Pevzner liken this to a car factory that gathers the raw materials, prepares the materials, and assembles a copy of itself, all while making cars at the same time [42]. Because cells are the basic reaction vesicles in the body, understanding their inner-workings would lead to a greater overall understanding as to how the body functions, which would be very valuable to scientists.

Although cells come in many shapes and sizes and with varying functions, each has three common components: DNA, RNA, and protein molecules. DNA, or deoxyribonucleic acid, is often described as the building block of life, as it contains the genetic material governing how the cell works. DNA is composed of the nucleotides adenine, guanine, cytosine, and thymine and is a double-stranded, helical molecule. RNA, or ribonucleic acid, is composed of a single strand of the nucleotides adenine, guanine, cytosine, and uracil. It is produced to transfer pieces of the DNA strand to other locations in the cell. Proteins are molecules made up of amino acids that produce enzymes that can be thought of as the laborers of the cell. This is because they perform functions varying from assembling strands of nucleotides to signaling other cells.

The double-stranded helical structure of DNA lends itself to replication. To replicate, a chromosome in the DNA "unzips" and then an enzyme called DNA polymerase, which is prevalent throughout the cell, attaches itself to one of the strands. It then moves along the strand of DNA, attracting nucleotides that com-

plement the one it is currently working on. These nucleotides hydrogen bond to one another and the process continues until the chromosome is copied. This process concurrently happens on the other original strand, completing the replication.

It is also important to understand the central dogma of molecular biology. This outlines the general process by which proteins are generated from DNA. Figure 3.1 denotes this process. We start with DNA, which is made up of four nucleotides. RNA is produced through a process called transcription. First, DNA "unzips" and then an enzyme called RNA polymerase binds complimentary nucleotides to one strand of DNA, starting at the promoter region. This process continues until the RNA polymerase reaches the terminator region, at which point the RNA strand breaks off and the DNA strands "zip" back together. This completes the transcription part of the process. Next, the RNA undergoes translation to produce a polypeptide chain. Here, a molecular machine called a ribosome does most of the work. The RNA strand produced by the transcription process is more specifically called messenger RNA, or mRNA. Another type of RNA, transfer RNA, or tRNA, continually floats around within the nucleus. Ribosomes move along the mRNA strand, reading groups of three nucleotides, or codons at a time. Each codon encodes for an amino acid. There are sixty-four possible combinations of nucleotides, but only twenty different amino acids. Figure 3.2 shows which codons refer to which amino acids. As the ribosomes move along the mRNA, they decipher the codons and the tRNA molecules bring the correlating anticodons. These amino acids are assembled into a chain, called a polypeptide chain, which is folded to make up the protein. This process continues until the stop codon is encountered.

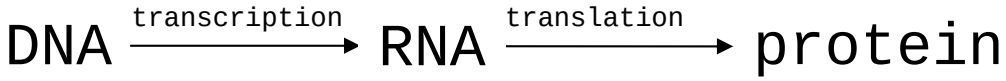


Figure 3.1. Central Dogma of Molecular Biology. DNA undergoes transcription to produce RNA, which in turn undergoes translation to produce protein.

	U		**C**		**A**		**G**	
U	UUU UUC	Phenylalanine	UCU UCC	Serine	UAU UAC	Tyrosine	UGU UGC	Cysteine
	UUA UUG	Leucine	UCA UCG		UAA UAG	Stop	UGA UGG	Stop Tryptophan
C	CUU CUC CUA CUG	Leucine	CCU CCC CCA CCG	Proline	CAU CAC	Histidine	CGU CGC CGA CGG	Arginine
					CAA CAG	Glutamine		
A	AUU AUC	Isoleucine	ACU ACC	Threonine	AAU AAC	Asparagine	AGU AGC	Serine
	AUA AUG	Methionine	ACA ACG		AAA AAG	Lysine	AGA AGG	Arginine
G	GUU GUC GUA GUG	Valine	GCU GCC GCA GCG	Alanine	GAU GAC	Aspartic acid	GGU GGC GGA GGG	Glycine
					GAA GAG	Glutamic acid		

Figure 3.2. Genetic Code. A codon is represented by a set of three nucleotides. Each codon codes for an amino acid.

3.3 Bioinformatics

Bioinformatics is the collective study of numerous fields and techniques used to solve biological problems. Specifically, researchers in the fields of biology, computer science, mathematics, and statistics are generally involved. Bioinformatics is mainly involved with the study of DNA and its underlying properties. Computer science is playing an ever-increasing role in bioinformatics, as the study and analysis of large sequences of data lend itself to the computer science discipline. Bioinformatics aims to develop a better understanding of the function of genes through the use of advanced, yet easy to use, (web) interfaces. We next describe most of the major bioinformatics research areas and then the tools and technologies currently used.

3.3.1 Research Areas

The following are several of the main research areas in bioinformatics.

Genome Annotation Genome annotation refers to locating genes in a sequence and then giving biological meaning to those regions. Genome annotation is often classified into functional and structural annotation.

Sequence Alignment Sequence alignment is the comparison of two or more sequences to one another. The goal of sequencing is to find similarities among the sequences. Sequencing is usually used in evolutionary studies. The two main types of sequence alignment are global and local alignment. Global alignment involves finding the best alignment of two sequences over the entire sequence, while local alignment concentrates on aligning shorter fragments in higher conserved regions. There is also a technique called

multiple sequence alignment. Here, many different sequences are all aligned against one another to help study evolutionary relationships.

Sequencing Sequencing is the study of finding the structure of a given sequence. For example, sequencing would be finding and listing the nucleotides from a short sequence of DNA. This is typically performed using a method known as Sanger sequencing [73]. Sanger sequencing uses a chain termination method to sequence. Another common technique is to perform shotgun sequencing on the Sanger sequencing results [57, 73]. Here, a sequence is divided into many small, random fragments and then sequenced using the chain termination method. Shotgun sequencing was used in the sequencing of the human genome [83].

Genome Assembly Genome assembly refers to the process of assembling many short DNA sequences together to form the original chromosome they once composed. The short sequences are often generated through shotgun sequencing.

3.3.2 Tools and Technologies

We next describe some of the prevalent tools and technologies used in bioinformatics, most of which we have utilized in our work.

Perl Perl [64], or practical extraction report language, is a scripting language that lends itself to the bioinformatics field largely because of its rich parsing capabilities. Perl is an interpreted language, but is rather fast in its applications. It is often used to write scripts for web site development and is finding an increasingly larger niche in the bioinformatics field.

BioPerl BioPerl [14] is an extension of Perl that has many of the common tasks of bioinformatics research built-in. It offers a large number of Perl bioinformatics modules, such as modules to manipulate sequences or to search for specific genes. BioPerl is an ever-growing, popular tool, but we found it easier to code our own scripts for our specific needs.

BLAST BLAST [15, 57], or basic local alignment search tool, is a very popular algorithm used to compare sequences. It allows researchers to quickly compare a given sequence with sequences in a large database, yielding results with the best hits according to an expectation value.

Hidden Markov Model A hidden Markov model (HMM) is a probabilistic model used for statistical analysis of problems [28, 29, 57]. HMMs were first used for speech recognition, but work very well for bioinformatics problems, such as sequence analysis. Typically, a training set of known parameters is used to build the HMM, which will contain varying states and probabilities for transitions between them. HMMs are powerful because they consider all possible combinations and help to understand or to determine the hidden parameters in the model.

Phylogenetic Tree A phylogenetic tree [57, 65] shows evolutionary relationships among a collection of entities. Here, we use it to determine possible relationships between groups of sequences, either nucleic acids or amino acids. Branch length within the tree denotes how close two entities are related. Similar sequences will be closer together and connect to a common node, while dissimilar sequences will be farther apart.

Bioinformatics Collaboratories The collection of bioinformatics research sites

on the web is perhaps the most important tool in the bioinformatics. These sites allow researchers to easily utilize many of the tools described above. They also allow researchers to quickly share results and progress, enabling the rapid growth in the field. Specifically of note is NCBI [60], the National Center for Biotechnology Information, which contains a large collection of literature in the field as well as one of the largest collections of genome sequencing data in the world. Ensembl [30] is another important resource. Their main focus is on the annotation of eukaryotic genomes. VectorBase [82] is another important bioinformatics resource that focuses mainly on the *Anopheles gambiae* and *Aedes aegypti* genomes, but also other vectors.

Bioinformatics Algorithms Because of the immense size of most genomes, it is imperative that the tools biologists use to work with them are optimized. Much work has been done to optimize varying bioinformatics algorithms [42]. A good example is BLAST. It is probably the most widely used tool in bioinformatics. It has been optimized for speed, but lacks some of the capabilities other tools have. Some other well-known algorithms are the Needleman-Wunsch [61] and the Smith-Waterman algorithms [80]. A background in computer science or a related field is important when using or refining any of the algorithms.

3.4 *Aedes aegypti*

Aedes aegypti, shown in Figure 3.3 [82], is a mosquito native to the tropics that is capable of carrying the yellow fever and dengue viruses - yellow fever alone is estimated to kill 30,000 people each year [84]. Its unannotated genome was very recently released [60], so there has been little work done on it to date. More

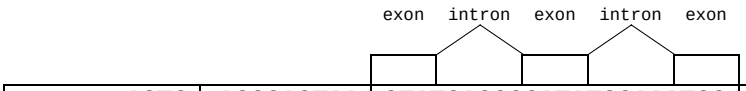
Figure 3.3. *Aedes aegypti* mosquito. From VectorBase [82].

extensive study [36, 56] has been performed on the malaria carrying mosquito, *Anopheles gambiae*, as well as other species. Studying the *Aedes aegypti* genome is important for a number of reasons. First, its genome is roughly ten times larger than the *Anopheles gambiae* mosquito. Understanding why there is such a large difference would be a major accomplishment in the field. Second, as a vector for two deadly diseases, advanced study on the *Aedes aegypti* genome, specifically how it has evolved, has the potential to help us understand how it carries the diseases, likely helping us to stop such transfers to humans. Lastly, the transposable elements found in the *Anopheles gambiae* genome have motivated the study of finding them in *Aedes aegypti* and then further understanding their significance. We next describe what transposable elements are as well as the transposable elements we targeted.

3.4.1 Transposable elements

Transposable elements are sequences of DNA that are found throughout the genome. They were first found and analyzed by McClintock [54]. Transposable elements have the ability to move about the genome and there are typically multiple copies of the same transposable elements found throughout the genome. Transposable elements often make up large portions of a particular genome, as evidenced by them representing approximately 35% of the *Homo sapiens* genome [57]. They have been evidenced to be a cause of chromosomal breakage, to have the ability to transfer genetic material, to be resistant to antibiotics, and to be able to regulate gene expression [79]. These are all motivators for the study of transposable elements.

Transposable elements found in eukaryotic genomes are typically divided into Class I, Class II, and Class III elements. Class I elements are commonly referred to as retrotransposons and follow RNA transcription techniques. They use reverse transcriptase to transcribe RNA to DNA. Class I elements are also characterized by their long terminal repeat (LTR) sequences. Class II elements are generally referred to as transposons. Transposons move about a genome with the use of the transposase enzyme. These transposons carry the capability of moving and inserting themselves pretty much anywhere in the genome. Also, Class II elements have inverted repeats at either end of the transposase enzyme. The transposase is made up of introns and exons, which mean non-coding and coding regions of the DNA. Class II elements are particularly useful to study because genes can be inserted into the transposon, allowing transgenic researchers to observe the results. In this way, transposons are used as gene vectors [43]. Class III transposable elements are very similar to Class II transposons, but are much shorter and don't



AGTC	ACGCAGTAA	GTATCACGCCATATGGAAATCG	AATGACGCA	AGTC
Target Site Duplication	Inverted Repeat	Transposase	Inverted Repeat	Target Site Duplication
4-8 bp	30-200 bp	1200-5000 bp	30-200 bp	4-8 bp

Figure 3.4. Typical Class II Transposon Structure. Class II transposons are characterized by a single transposase flanked by inverted repeats. The transposase is made up of exons and introns.

seem to contain coding sequences. This study focuses on identifying Class II transposons. Figure 3.4 shows a typical Class II transposon. Appendix B.1 shows an annotated *mariner* transposon. Descriptions of the families of the Class II transposons we study follow.

piggyBac The *piggyBac* family of transposons were first discovered in the cabbage looper moth *Trichoplusia ni* [21]. This transposon has since been found in a number of organisms and has proven to be effective as a gene vector for a variety of organisms, including *Aedes aegypti* [51].

Tc1 Like all transposons, *Tc1* transposons are flanked by inverted repeat sequences. They are good gene vectors, extensively used in the analysis of *Caenorhabditis elegans* [66]. *Tc1* transposons are typically recognized because of their DDE motif. The DDE motif refers to a sequence of amino acids. In this case they are Aspartic acid, Aspartic acid, and Glutamic acid.

pogo These transposable elements are members of the *Tc1* superfamily. They are very similar to *Tc1* transposons, but lack the DDE catalytic motif.

mariner The *mariner* transposons are characterized by a DDD motif and gen-

erally contain one exon transposase. They are widespread in invertebrates and are well characterized.

***P* element** *P* elements were first discovered in the fruit fly *Drosophila melanogaster* genome and were the first transposons to be used as a gene vector [43, 72]. Since the initial discovery of *P* elements, they have been found in several other species, most notably the malaria mosquito *Anopheles gambiae* [77]. This has motivated the search for *P* elements within the *Aedes aegypti* genome.

3.5 Approach to Identifying Transposable Elements

The goal of this study was to identify transposable elements within the newly released *Aedes aegypti* genome through the use of computer based tools and methods. Identifying transposable elements is important for a number of reasons. For example, transposable elements help with gene annotation. Also, identifying them will help lead to a better understanding of their use as a genetic manipulation tool. We next describe a basic approach to identifying transposable elements and then two approaches we used to identify transposable elements within the *Aedes aegypti* genome. This section concludes with our description of a proposed hybrid method. All of our approaches utilize common characteristics of Class II transposable elements for our searches.

3.5.1 Typical Approach

A typical bioinformatics technique to finding transposable elements within genomes is to BLAST [26] for them. This involves taking known transposable elements from other genomes and searching for them in a given genome. This

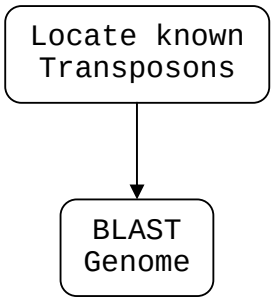


Figure 3.5. Typical Approach used to Identify Transposable Elements. Researchers often BLAST genomes using known transposons. This is an efficient technique, but improvements can be made.

approach is shown in Figure 3.5. Some limitations of this approach include not being able to search for every variation or for the entire length of a sequence.

This is a good approach to identifying known transposable elements in similar genomes, but does not account for frame shifts or poor matches in the genome. This approach also does not help researchers find new transposable elements. However, BLAST is a very popular technique to finding transposable elements [26].

3.5.2 First Approach

Because we wanted to perform a more thorough search for transposable elements, we developed a new approach that utilized multiple tools. This approach was loosely based on our previous work [22] and focused on identifying *P* elements. We improved on the typical approach by utilizing the HMMER suite of tools [35], most notably incorporating and using a HMM to help us in our search. Figure 3.6 shows the process we used for this approach. We next give a brief description of each step.

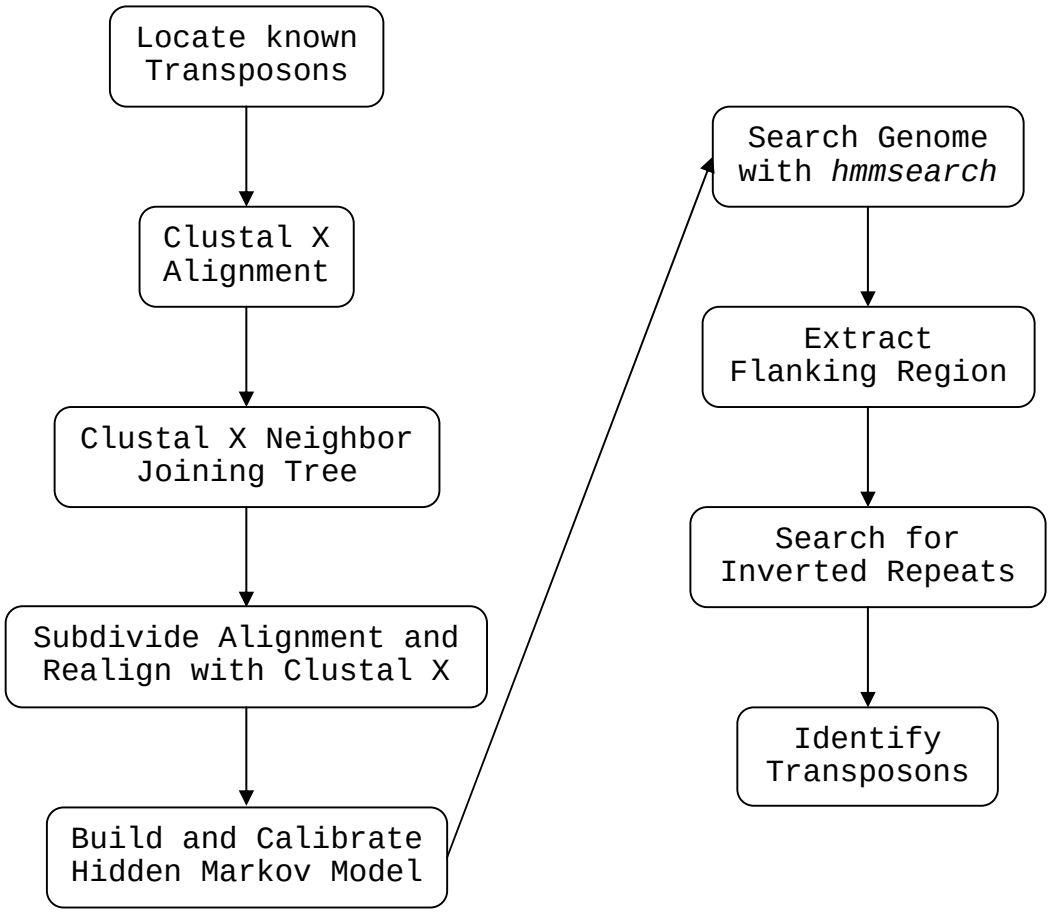


Figure 3.6. First Approach. We started by locating known transposons and then aligning them. We then built a neighbor joining tree and realigned the sequences. Next, we built and calibrated the hidden Markov model and then searched the genome with *hmmsearch*. Finally, we extracted the flanking regions so we could search for inverted repeats and identify transposons.

1. **Identify Transposons** For this step, we identified transposons from other species. This was accomplished through either a text-based search for particular elements or through a database search. This step was necessary as it identified the transposons we were searching for in the *Aedes aegypti* genome.

2. **Clustal X Alignment** Once the sequences had been visually subdivided by a graduate biology student, James R. Hogan, we proceeded to align the protein coding transposase with known sequences using Clustal X [23]. Clustal X is a general multiple sequence alignment program that produces the best matches and lines up the sequences. It works by performing pairwise alignments on the sequences, and then producing a phylogenetic tree based on alignment scores. It then utilizes dynamic programming techniques to utilize the results from the phylogenetic tree to align the sequences. Figure 3.7 shows the output from some of our work here.

3. **Clustal X Neighbor Joining Tree** We utilized Clustal X's ability to produce a neighbor joining tree. This enabled us to see how closely different sequences were related and helped with the subdividing in the next step.

4. **Subdivide Alignment and Realign with Clustal X** James R. Hogan again visually subdivided the sequences and realigned them. This time, the sequences were aligned by subgroup, meaning by the best alignment of smaller conserved regions.

5. **Build and Calibrate HMM** We used the sequences from the previous step to build and calibrate our model. For this step we used *hmmbuild* and *hmmcalibrate*, both from the HMMER suite of tools. The suite will generate a functional HMM that can be used to run against any set of sequences.

Figure 3.7. Clustal X Alignment for *P* elements. This shows a subsection of the multiple sequence alignment we performed. Much of the regions shown are highly conserved.

Appendix B.2 shows a sample portion of our HMM for *pogo* transposons.

6. **Search Genome (*hmmsearch*)** The HMMER suite of tools also includes *hmmsearch*, which utilizes the HMM to search the genome. This step produced ranked output for the sequences.

7. **Extract Flanking Region** We utilized various scripts to extract the sequences and their flanking regions. We took the results from *hmmsearch* and extracted the best hits from the NCBI *Aedes aegypti* genome. These scripts added flanking regions to either end of the hit sequences so we could search for inverted repeats. The scripts took the start and end frame as arguments and then searched the large (1 GB) genome file for the correlating supercontig and then for the region of the sequence within that supercontig. Appendix B.4 describes and shows one of the scripts we used for the extraction.

8. **Search for Inverted Repeats** We used a tool called *BLAST 2 Sequences* [81] to search for inverted repeats within the flanking regions. This program simply blasts a sequence against itself, allowing us to find and identify inverted repeats. Again, we wrote various Perl scripts to extract relevant data from the results. The scripts took into account the orientation of the sequence and were run multiple times through a shell script for efficiency.

9. **Identify Transposable Elements** Transposable elements were identified by careful examination of the highest scoring sequences by the aforementioned James R. Hogan. We submitted these sequences to BLAST and were given expectation values. He then did further inspection and we utilized various tools, such as phylogenetic trees to help us with our efforts.

There are a couple advantages to using this approach. One advantage is that we are able to identify previously unknown transposable elements. This was performed by utilizing some basic known characteristics of Class II transposons, such as their inverted repeats, when searching new genomes. Additionally, the use of tools such as Clustal X and the HMMER suite allowed us to perform a more thorough search. Clustal X helped us to identify closely related sequences and the HMMER suite allowed us to construct a HMM that gave us scored results of the hits.

There are also a couple disadvantages to this approach. The main disadvantage is that our search does not account for frame shifts. Frame shifts are common occurrences in genomes, so the fact that we do not allow for them in our searching means that we could be missing hits. Also, this technique will not help us find stop codons, introns, or exons, each of which can help to identify transposons. While this approach can be used for all the families in question, we only utilized it for the *P* element family. Lastly, this technique is much more time consuming than the first approach; however, it is also a much more thorough approach to locating transposons.

3.5.3 Second Approach

Our first approach, Figure 3.6, had a couple main drawbacks, the largest of which was that it did not account for frame shifts in the sequences. We expected we would find more transposable elements if we allowed for frame shifts. To do this, we slightly modified our approach. We began like in the previous approach, but instead of using the HMMER suite to perform the search with the HMM, we used *GeneWise* [31]. We still used the HMMER suite to build and calibrate the HMM,

we simply had *GeneWise* utilize the model. *GeneWise* is useful when comparing sequences and has good gene prediction capabilities. Among the algorithms it uses is the Smith-Waterman algorithm. *GeneWise* is more powerful than *hmmsearch* because it can account for frame shifts, gaps, and handles introns and exons. Using *GeneWise* was much more computationally expensive (we distributed it on an advanced cluster of machines), but gave us more confidence that we were doing all we could to identify transposable elements. *GeneWise* produced the highest scoring sequences, which we proceeded to extract from the genome. A diagram showing our entire process for this approach can be found in Figure 3.8. This time, we were able to utilize the approach for the five families of transposons previously described. More about *GeneWise* and portions of the *GeneWise* output can be found in Appendix B.3.

The main advantages to using this approach include the ability to handle frame shifts when searching the genome, as well as being able to identify stop codons, introns, and exons. Also, the fact that we were able to search for all families in question made for more meaningful results. A drawback to this approach is that *GeneWise* is computationally expensive. Additionally, we did not utilize the common characteristics of Class II transposons when performing this approach.

3.5.4 Hybrid Approach: A Transposable Element Discovery Methodology

We propose a methodology that combines the best elements from both our approaches. Most notably, we follow the general method of the first approach, but use *GeneWise* to search the genome. We conclude the search similar to how we conclude the first approach. This approach can be used for all the families in question in this study. While we did not explicitly use this approach, it is an

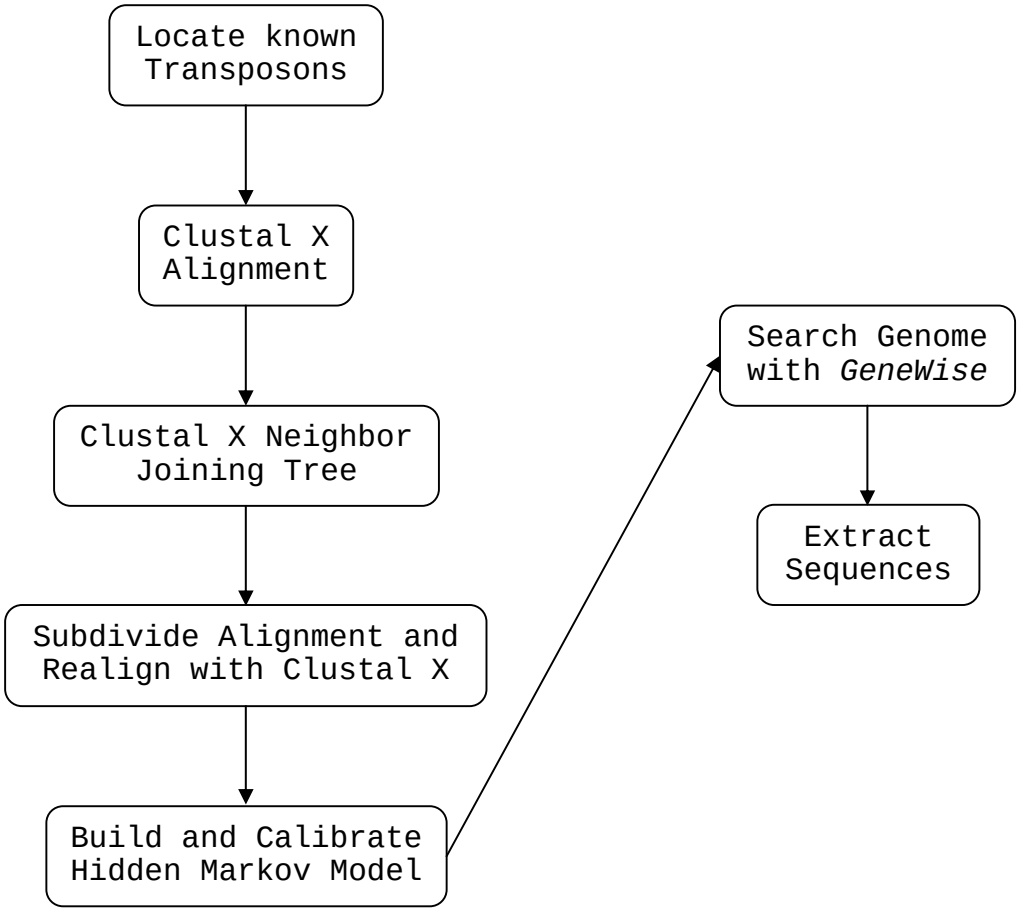


Figure 3.8. Second Approach. This approach is very similar to the first approach, except that we used *GeneWise* to search the genome and then only extracted the sequences.

extension of the approaches we used that would likely produce more meaningful results. A schematic for this process is shown in Figure 3.9.

3.6 Discussion

We found a reasonable number of transposable elements for each of the families. Table 3.1 shows the total counts for each family. It is important to note that these counts are only for the most likely transposable elements. We had many more hits than listed here, but these are the ones that had the highest probability of being transposable elements. Each hit refers to a transposase protein coding sequence.

We also present the phylogenetic tree for the *mariner* family in Figure 3.10. There appears to be two or three clades that are clustered rather closely together, indicating close relationships.

Our proposed hybrid approach would likely be more useful than our first and second approaches, as it combines the better parts of the two. The methods we both used and proposed make the use of some scripting techniques and the combination of tools we used is novel.

3.7 Conclusion

Through our work, we have identified some new transposable elements and found some known ones in the newly released *Aedes aegypti* genome through a unique computational process. Locating transposable elements in new genomes is important because it allows us to compare them to those in similar genomes and then gain understanding regarding differences. Identifying new transposable elements has the potential to help with genetic evolution studies because new transposable elements may be good gene vectors. Additionally, to our knowl-

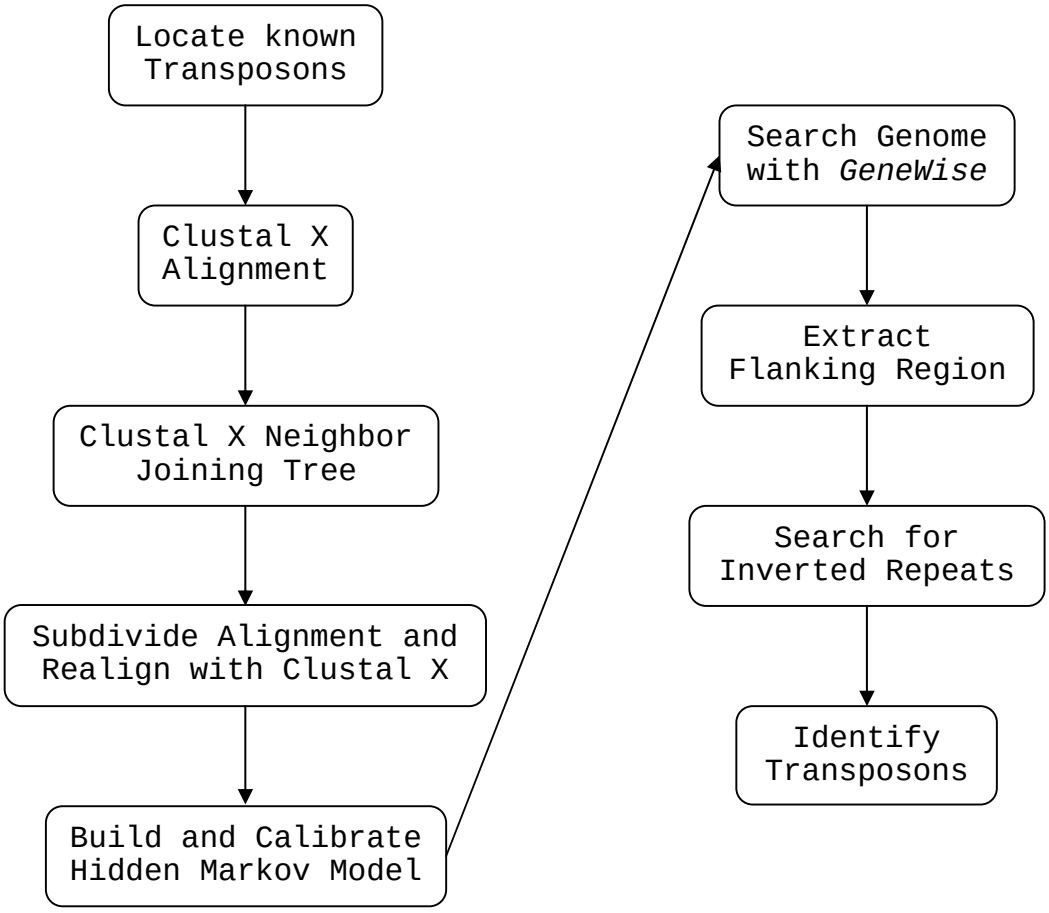


Figure 3.9. Hybrid Approach: A Transposable Element Discovery Methodology. This approach combines the better elements of the first and second approaches, making for a more complete search.

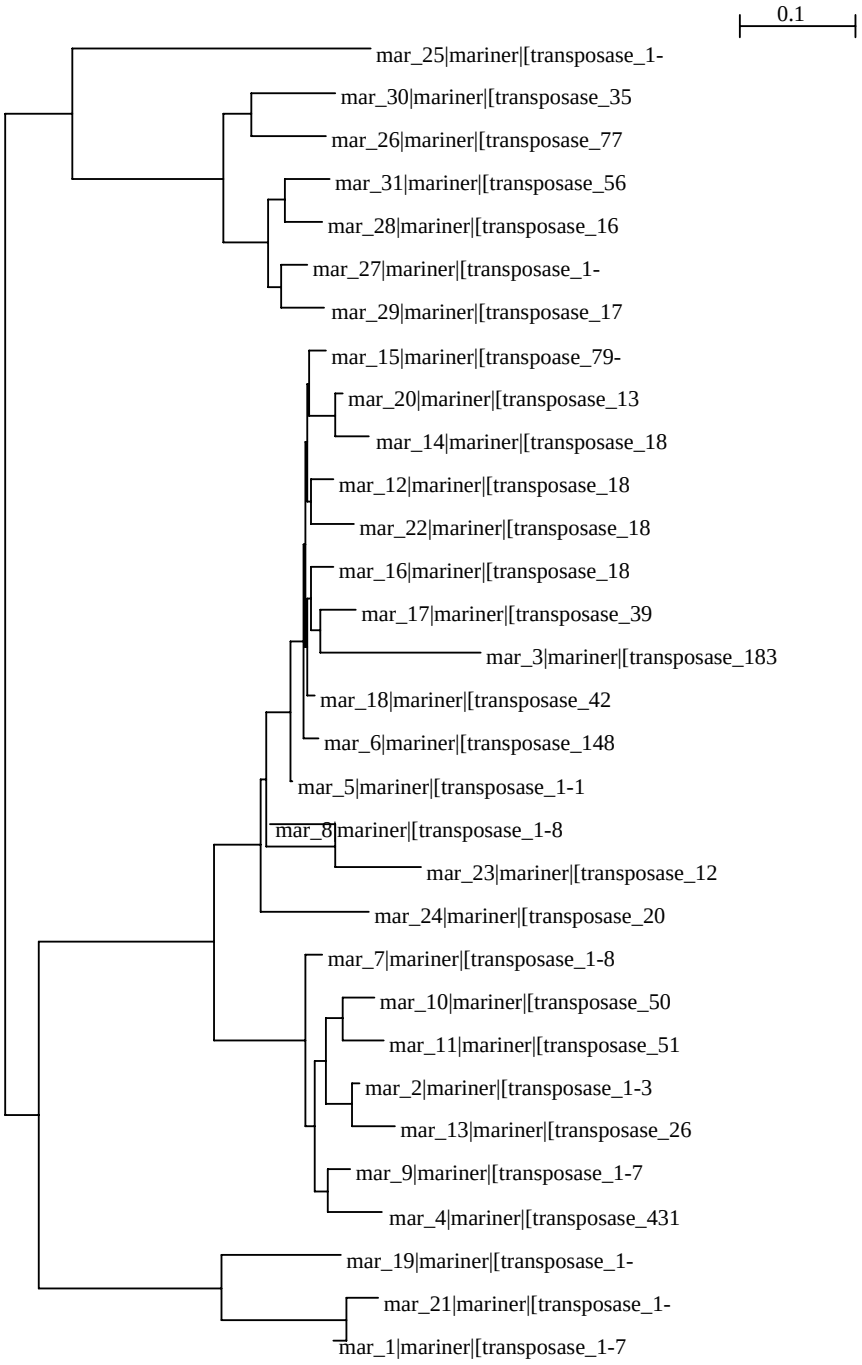


Figure 3.10. Phylogenetic tree for the *mariner* family. Clades that are clustered together indicate close relationships.

TABLE 3.1

NUMBER OF TRANSPOSABLE ELEMENTS FOUND

TE	**Count**
piggyBac	12
Tc1	72
pogo	50
mariner	25
P element	9

edge, this is the first such study using an approach like ours on the *Aedes aegypti* genome. Approaches similar to this, namely the use of HMMs, have been performed on other genomes [11, 55]. However, we have utilized and proposed unique approaches that combine typical biological methods with computer science techniques to perform a more thorough and efficient search.

We have utilized two main approaches when searching for transposable elements, one mainly through the use of the HMMER suite of tools and one that extends that toolkit and uses *GeneWise* to search the genome with the HMM. We finally proposed a hybrid approach that offers the best aspects of both techniques.

CHAPTER 4

CONCLUSION

4.1 Overview

In this thesis, we have shown that verification and validation of simulation models is both meaningful and vital to the confidence in a simulation. This has been shown through the two case studies presented. We have also shown and utilized novel and efficient approaches to locate transposable elements within a genome. We elaborate on our conclusions in the following sections.

4.2 Verification and Validation of Agent-based and Equation-based Simulations

Part one of this work centered around the verification and validation of agent-based and equation-based simulation models. We performed an in-depth study of an agent-based model, namely utilizing techniques such as docking and visualization to increase confidence in the model in a cost-effective manner. We applied multiple subjective and quantitative verification and validation techniques to the model and summarized our results. We did the same for an equation-based model, which we also ported to another language for dramatic performance gains. The study concludes with some generalized recommendations when performing verification and validation on simulation models, specifically how to use venerable techniques on the recently popularized agent-based simulation models.

This work is limited in that we focused on a specific set of verification and validation techniques. Increasing the number of techniques used would help increase confidence. Further, the amount of quantitative techniques that we used is rather small. Performing tests that add more statistical significance to our models would be valuable. Finally, the first case study is limited in that the simulation data has not been fully docked with real-world data. As such, docking the NOM Model against the AlphaStep model is only as valuable as the AlphaStep model is accurate. To this point, various iterations of this work have been presented and published in the proceedings of three peer-reviewed, national conferences [44, 46, 87].

4.2.1 Future Work

Although we have performed a good deal of work, there is still a lot of relevant work to be performed. First, a more in-depth survey of verification and validation techniques, similar to Balci's [6] work should be performed with an agent-based modeling flair. This would further increase the confidence of the results produced by such techniques. It would also offer insight as to which techniques would be valuable that we have overlooked. Second, more rigorous quantitative techniques need to be used. Difficulties in porting the second case study to C++ took longer than expected, so we were not able to perform as many statistical tests as we had hoped. In general, more statistical testing for both case studies would help increase confidence. Third, it has been suggested by Macal [52], among others, that "invalidating" simulation models can be valuable to researchers. They mention that this could eliminate some of the "validation bias." Finally, developing a more general and formalized process model to perform verification and validation on simulation models would be important. Techniques such as this would help

researchers tremendously.

4.3 Bioinformatics Computing: Identifying and Analyzing Transposable Elements in the *Aedes aegypti* Genome

Part two of this thesis relied on some of the well-known and active research areas in bioinformatics. We focused on identifying transposable elements within the newly released *Aedes aegypti* genome through the use and design of some innovative approaches. Our work utilized some of the popular tools in the field, most namely BLAST, but also made good use of lesser used tools and techniques, such as *GeneWise* and HMMs. We created a novel process model to discovering transposable elements and helped biologists automate steps in the process through the use of scripts. Our efforts have helped us to discover over one-hundred and fifty possible transposable elements within several of the families. We have done so in a timely manner and have aided the research in this area through our novel approaches.

The limitations of this work are largely limited to simply proposing the hybrid approach to finding transposable elements. While we did complete the first approach for the P elements and the second for five families of transposons, we did not tie the two together and perform the hybrid approach. Additionally, this study was rather limited in the amount of time available for our testing. We chose to use *GeneWise* for the core of our second and more thorough approach, which sacrificed performance, but produced more intelligible results.

4.3.1 Future Work

In the future, it would be valuable to fully follow the hybrid approach for the families of transposable elements presented. We could use this to compare with the results obtained from the other two approaches. Automating more portions of all the approaches presented would also be useful. Additionally, a comparison study between transposable elements that have been found in the *Anopheles gambiae* mosquito and those identified in the *Aedes aegypti* mosquito would help us understand the inherent differences between the two genomes, as well as some of the evolutionary characteristics.

APPENDIX A

CHAPTER 2 SUPPLEMENTARY MATERIAL

A.1 Case Study I

The NOM cluster is made up of eight simulation machines, a firewall, and an Oracle database. Figure A.1 shows the setup of the machines, while Figure A.2 shows the abstract view. The main interface to the cluster and the simulations is through the web. Figure A.3 shows the web page where users can enter data for the simulation. Additionally, users can view simulation results and reports online. Figure A.4 shows a sample report that is dynamically updated as the simulation runs. Further information on the NOM project is available [2, 37, 39, 85, 86].

Figure A.1. NOM Cluster of Machines. We have eight simulation machines, a web server, a firewall, and two database machines.

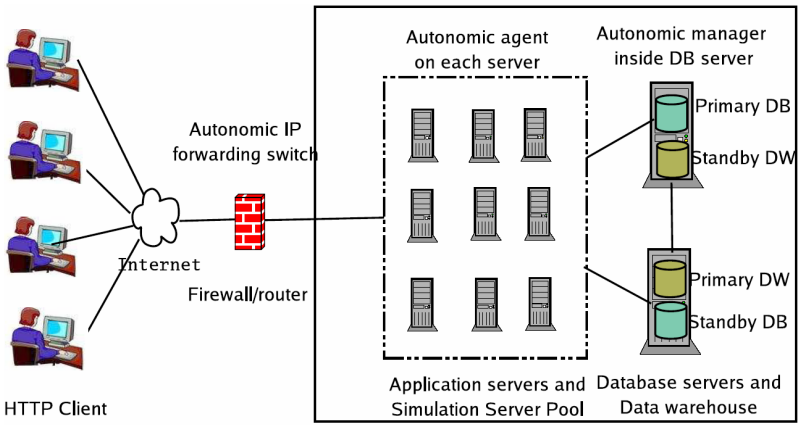


Figure A.2. Abstract Layout of the NOM Cluster (from Huang [38]).

Users have the ability to run one of four different simulation types, all of which are based on the same general conceptual model. There simulation types can be grouped into *No-flow* and *Flow* divisions or *Sorption* and *Reaction* divisions. *Flow* refers to the molecules having the ability to flow into and out of the system as it runs, while *No-flow* refers to a batch space model, meaning one where molecules cannot enter or leave the system as it runs. *Flow* experiments are sometimes called column experiments, as the molecules are present on a cylinder, with liquid flowing "down a column." Figure A.5 shows a column experiment being performed in the laboratory at the Environmental Molecular Science Institute at Notre Dame. The *Sorption* implementation refers to molecules having the ability to adsorp or desorp to the surface or to one another. This implementation is more interested on the physical characteristics and reactions of NOM. The *Reaction* implementation gives molecules the ability to undergo one of twelve chemical reactions, as shown in Table 2.1. These implementations are contrasted with AlphaStep, which is more of a *No-flow Reaction* implementation. The AlphaStep interface is shown in Figure A.6.

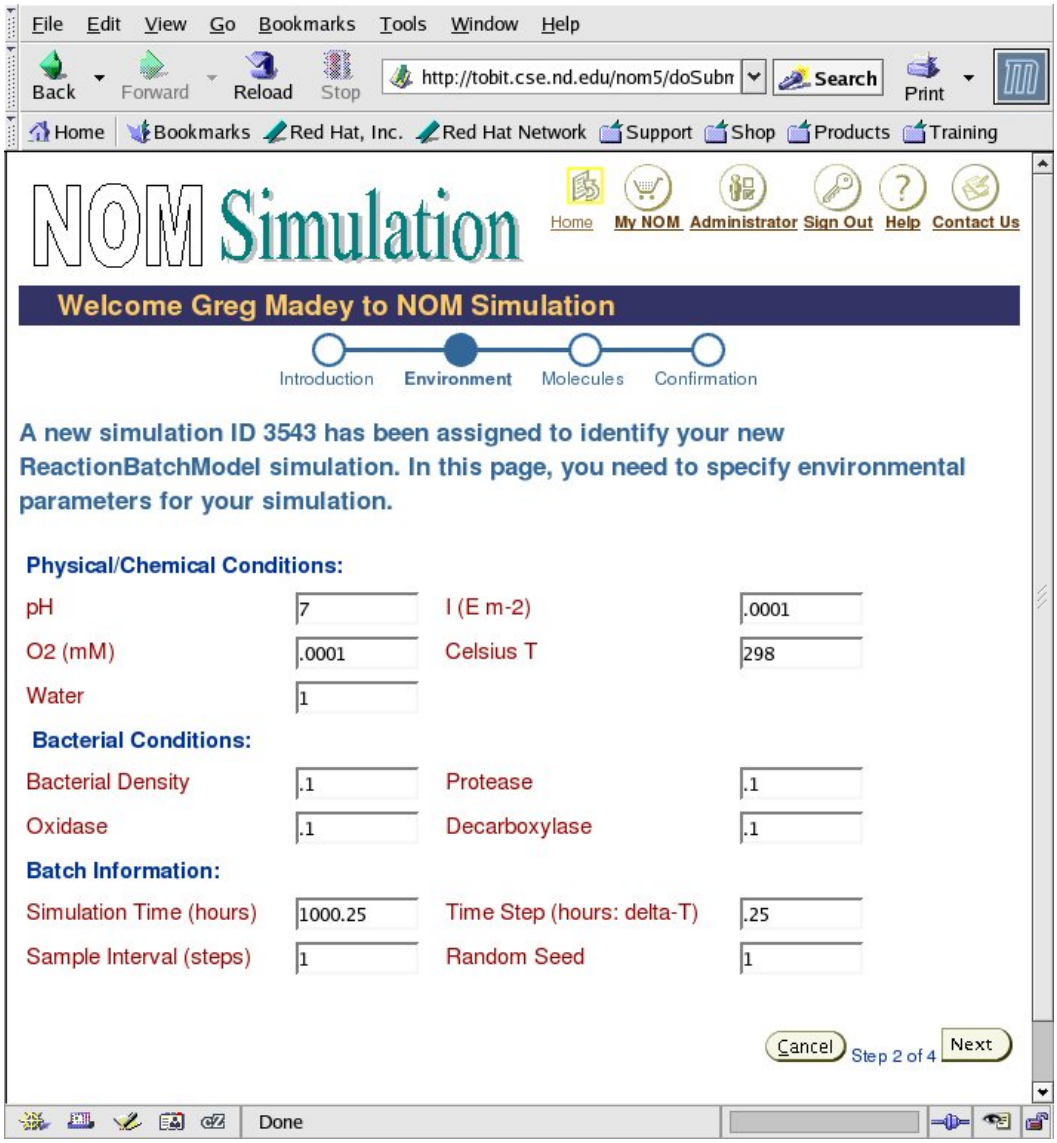
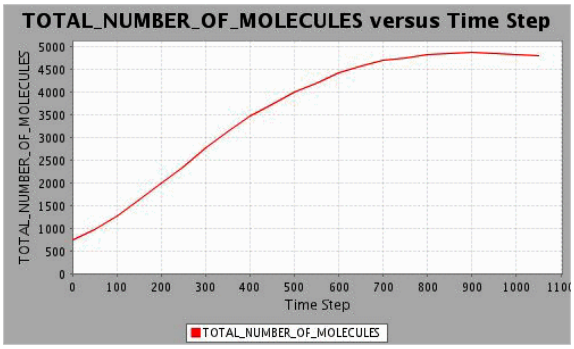
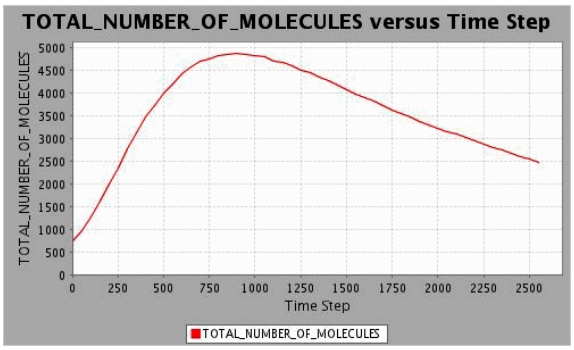


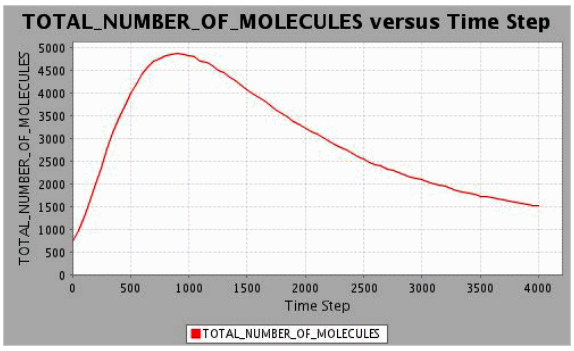
Figure A.3. Interface to the NOM Simulator. Here, the user can enter parameters for the chosen simulation. The entire simulator has a web-based front-end, allowing users to submit and view simulations.



(a) Simulation after 1000 time steps



(b) Simulation after 2500 time steps



(c) Completed Simulation

Figure A.4. Real-time Graphs of the Simulation. Here, web-based graphs (accessible through the NOM interface) are shown for a particular simulation at three points during the same simulation run. Panels (a) - (c) show the progression of the total number of molecules in the system over the duration of the simulation run.

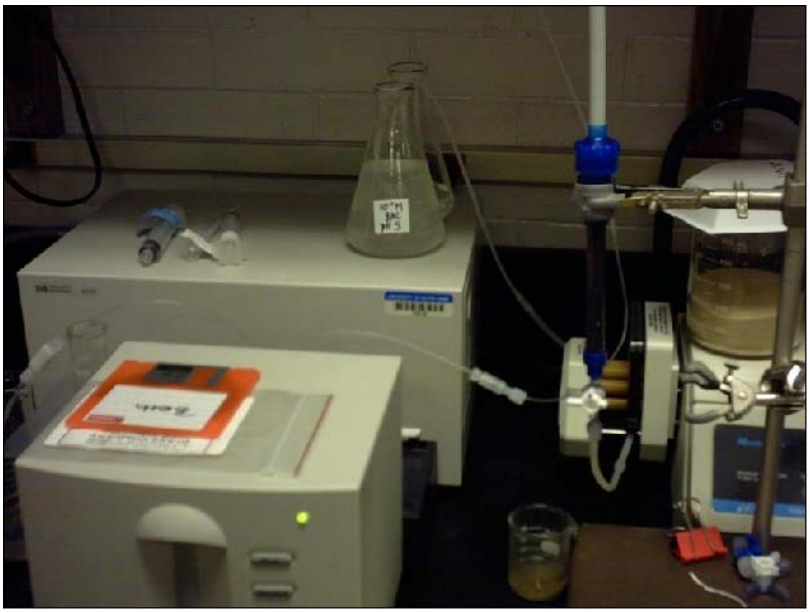
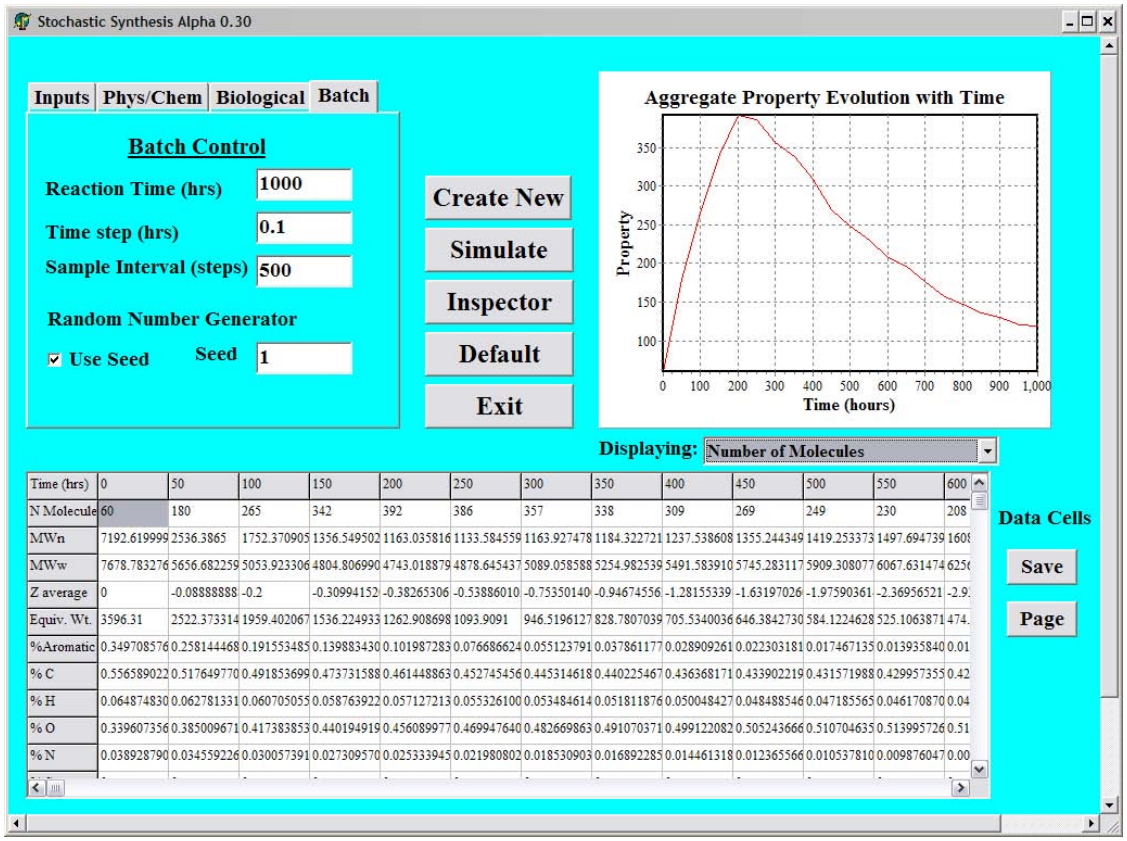


Figure A.5. Column Experiment. This image shows a *Flow* column experiment being performed in the laboratory. We attempted to mimic this in our models.



Time (hrs)	0	50	100	150	200	250	300	350	400	450	500	550	600
N Molecule	60	180	265	342	392	386	357	338	309	269	249	230	208
MWn	7192.619999	2536.3865	1752.370905	1356.549502	1163.035816	1133.584559	1163.927478	1184.322721	1237.538608	1355.244349	1419.253373	1497.694739	1608
MWw	7678.783276	5656.682259	5053.923306	4804.806990	4743.018879	4878.645437	5089.058588	5254.982539	5491.583910	5745.283117	5909.308077	6067.631474	6256
Z average	0	-0.08888888	-0.2	-0.30994152	-0.38265306	-0.53886010	-0.75350140	-0.94674556	-1.28155339	-1.63197026	-1.97590361	-2.36956521	-2.9.
Equiv. Wt.	3596.31	2522.373314	1959.402067	1536.224933	1262.908698	1093.9091	946.5196127	828.7807039	705.5340036	646.3842730	584.1224628	525.1063871	474.
%Aromatic	0.349708576	0.258144468	0.191553485	0.139883430	0.101987283	0.076686624	0.055123791	0.037861177	0.028909261	0.022303181	0.017467135	0.013935840	0.01
% C	0.556589022	0.517649770	0.491853699	0.473731588	0.461448863	0.452745456	0.445314618	0.440225467	0.436368171	0.433902219	0.431571988	0.429957355	0.42
% H	0.064874830	0.062781331	0.060705055	0.058763922	0.057127213	0.055326100	0.053484614	0.051811876	0.050048427	0.048488546	0.047185565	0.046170870	0.04
% O	0.339607356	0.385009671	0.417383853	0.440194919	0.456089977	0.469947640	0.482669863	0.491070371	0.499122082	0.505243666	0.510704635	0.513995726	0.51
% N	0.038928790	0.034559226	0.030057391	0.027309570	0.025333945	0.021980802	0.018530903	0.016892285	0.014461318	0.012365566	0.010537810	0.009876047	0.00

Figure A.6. AlphaStep Interface. Users can specify parameters for the simulation and then view results in a tabular as well as graphical form.

A.2 Case Study II

This section shows some sample code from both the Matlab and C++ implementations of the second case study. The excerpt shown is the else statement for a loop that does most of the heavy calculations in one of the functions. We first show the Matlab implementation that we were provided with in Section A.2.1 and then the C++ implementation we developed in Section A.2.2. In total, the Matlab implementation is around two-thousand lines long and the C++ implementation is around three-thousand lines long.

A.2.1 Matlab Implementation Sample Code

Below is a small section of code from the Matlab implementation that does a lot of the core calculations for the simulation.

```
else
    if ~analyticg
      if isempty(varargin)
             j         = 0;
             for    e  = eye(nv),
                 % When the 'for' statement is used with an expression,
                 % the columns of the expression are stored one at a time
                 % in the variable and then the following statements, up
                 % to the END, are executed.  Thus, the next line takes
                 % the first column of the identity matrix. Then the
                 %second, third... until n is reached.
                 xd     = x + delta*e;
                 [Fd]   = feval(FUN,xd,points);
                 j      = j + 1;
                 % The matrix below stores the differenced jacobian.  The
                 % matrix will be updated below.
                 grad(:,j) = Fd - f0;
             end;
             %Uncover the commands below to check the performance
             %of the gradient matrix.
             rcond1=rcond(grad);
      else
         grad = (feval(FUN,x*ones(1,nv)+tvec,points,varargin{:})-
                 f0*ones(1,nv))/delta;
      end
   else % use analytic gradient
      grad=feval(gradfun,x,points,varargin{:});
```

```
   end
   if isreal(grad)
      if rcond(grad)<1e-8;
          grad=grad+tvec;
      end
      dx0=-grad\f0;
      randomize=0;
   else
      if(verbose) ,disp('gradient imaginary'),end
      randomize=1;
   end
end
```

A.2.2 C++ Implementation Sample Code

Below is a small section of code from the C++ implementation that performs the same calculations as the Matlab implementation that was shown previously in Section A.2.1.

```
else {
  if(!analyticg) {
      j=0;
      for(int i=0; i<nv; i++) {
        for(int l=0; l<nv; l++)
          xd[l]=x[l]+delta_csim*e[l][i];
        Fd=projectioncommitedits(xd,points);
        j++;
        for(int m=0;m<nv;m++) {
          grad[m][j-1]=Fd[m]-f0[m];
        }
      } //end for
      rcond1=1.0 / (norm1(grad) * norm1(inv(grad)) );
  } //end if
  else { //not used
    proj=projectioncommitedits(x,points);
    for(int d=0;d<proj.size();d++)
      grad[d][0]=proj[d];
  }
  if(isreal) { //not used
    if( (1.0/(norm1(grad)*norm1(inv(grad))) ).real()<(pow(10.0,-8.0))) {
      for(int v=0;v<nv;v++)
        for(int w=0;w<nv;w++)
          grad[v][w]+=tvec[v][w];
    }
```

```
    grad=inv(grad);
    for(int aww=0;aww<nv;aww++)
      dx0[aww]=0;
    for(int u=0;u<nv;u++) //multiply grad*f0
      for(int y=0;y<nv;y++)
        dx0[u]+=grad[u][y]*f0[y];
    for(int u=0;u<nv;u++) { //multiply d0*-1
      dx0[u]*=-1;
    }

    randomize=0;
  } //end if(isreal(grad))
  else {
    if(verbose)
      fileOut << "gradient imaginary" << endl;
    randomize=1;
  }
} //end else
```

APPENDIX B

CHAPTER 3 SUPPLEMENTARY MATERIAL

B.1 Annotated *mariner* Transposon

This section shows an annotated *mariner* transposon, as submitted by Robertson [70].

```
DEFINITION  Anopheles gambiae clone Ag8 mariner transposase pseudogene,
            complete cds.
FEATURES             Location/Qualifiers
     source          1..1492
                     /organism="Anopheles gambiae"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:7165"
                     /clone="Ag8"
     source          70..1365
                     /organism="Anopheles gambiae"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:7165"
                     /transposon="mariner transposon"
     misc_feature    <1..69
                     /note="flanking DNA"
     misc_feature    70..71
                     /note="possible induced direct repeat"
     repeat_region   72..98
```

```
                     /note="left inverted terminal repeat"
     CDS             251..1293
                     /note="submitter-supplied
                     translation=ME-KEFRVLIKYCFLKRKNTVEAKTWLDNEFPD-PGKS-IIDW
                     YAKFKRGEMSTEDGERSGRPKEVVTDENIKKIHKM-LNREMKLIEIAEALKISKERV
                     GHIIHQYLDMQKLCAKWVPRELTFDQKHQRVDDSERCLPLLTRNTPEFLRRNVTMDE
                     TWLHHYTPESNRQSAQWTANGEP-PKRGKTQKSAGKVMTSVFWDANGIIFIDYLEKG
                     KTITSDYYMALLERLKVEIAAKRPHMKKKKVLFDQDNAPCHKSLRTMAKIHELGFEL
                     LPHPPYSPDLASSDFFLFSDLKRMLAGTKFD&NEEVVAETEAYFEAKPKEYYQKGIK
                     KLEGRYNRCIALEGNYVE; Conceptual translation requires
                     introduction of frameshifts (-) and stop codons (&)"
                     /pseudo
                     /codon_start=1
                     /product="mariner transposase"
     repeat_region   1337..1363
                     /note="right inverted terminal repeat"
     misc_feature    1364..1365
                     /note="possible induced direct repeat"
     misc_feature    1366..>1492
                     /note="flanking DNA"

ORIGIN
        1 cagtgacagc gtttaatcac cacacagccg agagatcaac acagctaacg aacgctccaa
       61 agtgaagcct aacaggttgg ctgataagtc cccggtctga cacatagatg gcgctgctag
      121 tattaaatgc atattatttt tatatagtac caaccttcaa atgattcgtg tcaaaatttg
      181 acgtctgtat gtcaattagt ttgtgagaca gagcgtcttc tgtcaagcaa cttttgttat
      241 tgtgaaaaaa atggaaaaaa aaggaatttc gtgttttgat aaaatactgt tttctgaaga
      301 gaaaaaatac agtggaagca aaaacttggc ttgataatga gtttccggac tcccagggaa
      361 atcaaataat tgattggtat gcaaaattca agcgtggtga aatgagcacg gaggacggtg
      421 aacgcagtgg acgcccgaaa gaggtggtta ccgacgaaaa cataaaaaaa atccacaaaa
      481 tgtcaaaatc attgaaccgt gaaatgaagt tgatcgagat agcagaggcc ttaaagatat
```

```
 541 caaaggaacg tgttggtcat atcattcatc aatatttgga tatgcagaag ctctgtgcaa
 601 aatgggtgcc gcgcgagctc acatttgacc aaaaacacca acgtgttgat gattctgagc
 661 ggtgtttgcc gctgttaact cgtaatacac ccgagttttt gcgtcgaaat gtgacaatgg
 721 atgaaacatg gctccatcac tacactcctg agtccaatcg acagtcggct cagtggacag
 781 cgaacggtga accaggttcc gaagcgtgga aagactcaaa agtccgctgg caaagtaatg
 841 acctctgttt tttgggatgc gaatggaata atttttatcg attatcttga gaagggaaaa
 901 accatcacca gtgactatta tatggcgtta ttggagcgtt tgaaggtcga aatcgcggca
 961 aaacggcccc atatgaagaa gaaaaaagtg ttgttcgacc aagacaacgc accgtgccac
1021 aagtcattga gaacgatggc aaaaattcat gaattgggct tcgaattgct tccccaccca
1081 ccgtattctc cagatctggc ttccagcgac tttttcttgt tctcagacct caaaaggatg
1141 ctcgcaggga caaaatttga ctgaaatgaa gaggtggtcg ccgaaactga ggcctatttt
1201 gaggcaaaac cgaaggagta ctaccaaaaa ggtatcaaaa aattggaagg tcgttataat
1261 cgctgtatcg ctcttgaagg gaactatgtt gaataataaa aacgaatttc gacaaaaaaa
1321 tgtgtttttc tttgttagac cggggactta tcagccaacc tgttataacc aacacattaa
1381 taaccgcctg gccttacacc atgggctgtg gcaaagtaaa gtcaaaacag cggagtgtga
1441 agacccgtta tctaccttcc attgcctcga acaactcagt tcagagcagg tc
```

B.2 Hidden Markov Model

A major portion of the approaches we used and proposed utilized HMMs. We described them in Chapter 3, but have provided an excerpt of the HMM we used for *pogo* transposons in Figure B.1.

B.3 *GeneWise*

To submit a job to *GeneWise*, we provided the program with the HMM and the FASTA format of the genome. *GeneWise* was installed on a high-performance computing cluster at the University of Notre Dame, called Bunch-of-Boxes, or BoB. This cluster consists of 256 Xeon computers and 128 64-bit Opteron computers. The entire cluster has over 5 TB of storage. Even with this massive computing power, *GeneWise* still took a few days to run because of the thoroughness in its search. It eventually gave us a lot of output data. A portion of this output has been heavily modified to fit in this thesis and is shown in Section B.3.1.

```
HMMER2.0
NAME  pogoaabig1
LENG  644
ALPH  Amino
RF    no
CS    no
MAP   yes
COM   gcg_hmmbuild -q -F -n pogoaabig1 /afs/nd.edu/user18/jhogan1/pogoaa/hmmerbuild_46.hmm /afs/nd.edu/user18/jhogan1/pogoaa/pogoaabig1.msf{*}
COM   gcg_hmmcalibrate -q /afs/nd.edu/user18/jhogan1/pogoaa/hmmerbuild_46.hmm
NSEQ  8
DATE  Tue Sep  7 10:40:17 2004
CKSUM 6040
XT      -8455     -4  -1000  -1000  -8455     -4  -8455     -4
NULT      -4  -8455
NULE     595  -1558     85    338   -294    453  -1158    197    249    902  -1085   -142    -21   -313     45    531    201    384  -1998   -644
EVD   -393.709961   0.090158
HMM        A      C      D      E      F      G      H      I      K      L      M      N      P      Q      R      S      T      V      W      Y
         m->m   m->i   m->d   i->m   i->i   d->m   d->d   b->m   m->e
         -418      *  -1991
     1   1892   -679  -1928  -1554   -935  -1518  -1163   -130  -1279   -387   3338  -1293  -1906  -1153  -1401   -745   -542    -74  -1586  -1222   145
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -    -21  -6659  -7701   -894  -1115   -701  -1378   -418      *
     2   1780  -1671     98   2173  -2302  -1250   -476  -1907   -270  -2071  -1308   -158  -1630   -122   -747   -561   -687  -1512  -2363  -1738   146
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -    -21  -6659  -7701   -894  -1115   -701  -1378      *      *
     3   2250   -603  -1750  -1659  -2470   -862  -1563  -2125  -1627  -2417  -1592  -1099  -1554  -1413  -1774   2219   -348  -1348  -2708  -2385   147
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -    -21  -6659  -7701   -894  -1115   -701  -1378      *      *
     4   -603  -1584   -541   -191  -2295  -1394   -323  -1976   2216  -1989  -1194   -360  -1667     62    129   1817   -635  -1560  -2125  -1630   148
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -    -21  -6659  -7701   -894  -1115   -701  -1378      *      *
     5   -639  -1060  -1095   -739  -1154  -1692   -684   -198   -344   -759   -313   -827  -1913   2484   -566   -867   -659   2011  -1623  -1136   149
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -    -21  -6659  -7701   -894  -1115   -701  -1378      *      *
     6   1349  -1389   1800   -236  -1684  -1520   -544  -1201   -365  -1444   2778   -458  -1769   -235   -808   -645   -610   -966  -1940  -1403   150
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -    -16  -7101  -8143   -894  -1115   -701  -1378      *      *
     7   3436  -1464  -2835  -3027  -3276  -1718  -2724  -2826  -3015  -3245  -2650  -2226  -2439  -2786  -2932  -1208  -1378  -2172  -3396  -3308   151
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -    -16  -7101  -8143   -894  -1115   -701  -1378      *      *
     8   1899   -936  -2387  -2401  -3075  -1193  -2199  -2783  -2376  -3066  -2207  -1605  -1932  -2099  -2429   2811   -726  -1850  -3305  -3047   152
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -    -16  -7101  -8143   -894  -1115   -701  -1378      *      *
     9  -1261  -1894  -1733  -1113  -2167  -2254   -921  -1653   2270  -1802   3401  -1209  -2388   -601   -352  -1356   1323  -1472  -2266  -1866   153
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -    -10  -7712  -8754   -894  -1115   -701  -1378      *      *
    10    973   2702  -1407   1119  -1359    781   -830   -927   -763    615   -494   -998  -2065   -650  -1100   -934   -722   -749  -1704  -1261   154
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -    -10  -7712  -8754   -894  -1115   -701  -1378      *      *
    11   -933  -1859  -1059   -509  -2008  -2005   -695  -1642   1021    162   2786   1199    851   -335   -814    672   -873  -1388  -2164  -1617   155
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -     -8  -8130  -9172   -894  -1115   -701  -1378      *      *
    12   -925  -1906  -1017   -474  -2087    430   -687  -1730   1057    104  -1067   1190  -2079   -313   -816   1422   -870    633  -2220  -1658   156
     -   -149   -500    233     43   -381    399    106   -626    210   -466   -720    275    394     45     96    359    117   -369   -294   -249
     -     -8  -8130  -9172   -894  -1115   -701  -1378      *      *
```

Figure B.1. *pogo* Hidden Markov Model. This figure shows a portion of the HMM that was generated by the HMMER suite of tools. Each letter in the top row of the matrix refers to an amino acid.

B.3.1 *GeneWise* Sample Output

```
Wise2 - database searching mode
Program: genewisedb version: \$Name: wise2-2-0 \$ released: unreleased
This program is freely distributed under a Gnu Public License.
   See -version for more info on copyright
Bugs and credits to Ewan Birney <birney@sanger.ac.uk>
-----------------------------------------------------


Algorithm type:        GeneWise
Search algorithm used: 623
Implementation:        Single Threaded processor (serial)
Search mode:           Single protein vs genomic db
Protein info from:     /home/biol/collins/jhogan1/hmms13106/paa.hmm
Dna info from:         /home/biol/collins/jhogan1/ncbi_ae_aegypti_genome.fasta
Start/End (protein)    default
Gene Paras:            human.gf
Codon Table:           codon.table
Subs error:            1e-05
Indel error:           1e-05
Model splice?          model
Model codon bias?      flat
Model intron bias?     tied
Null model             syn
Alignment Alg          623L


#
#WARNING!
#
# Your alignment algorithm is different from your search algorithm.
# This is probably quite sensible but will lead to differing scores.
```

```
# Use the search score as an indicator of the significance of the match
# Read the docs for more information
#

#High Score list
#Protein ID                 DNA Str  ID                          Bits Evalue
-------------------------------------------------------------------------
Protein paa                DNA [-] gi|78157278|gb|AAGE02008433.1| 508.63 5.3e-13
Protein paa                DNA [+] gi|78152614|gb|AAGE02012918.1| 469.58 3.8e-12
Protein paa                DNA [-] gi|78152125|gb|AAGE02013407.1| 426.44 3.4e-11
Protein paa                DNA [-] gi|78153301|gb|AAGE02012231.1| 410.12 7.8e-11
Protein paa                DNA [-] gi|78162597|gb|AAGE02003196.1| 373.06 5.1e-10
Protein paa                DNA [-] gi|78144592|gb|AAGE02020830.1| 371.84 5.4e-10
...continued...
Protein paa                DNA [-] gi|78150404|gb|AAGE02015117.1| 30.13 0.018
Protein paa                DNA [+] gi|78132436|gb|AAGE02032986.1| 25.09 0.023
Protein paa                DNA [-] gi|78156124|gb|AAGE02009496.1| 21.85 0.028

#Histogram
-------------------------------------------------------------------------
score    obs    exp  (one = represents 128 sequences)
-----    ---    ---
 -285      1    931|=      *
 -284      3    983|=      *
 -283     16   1032|=       *
 -282     49   1079|=       *
 -281    129   1123|==      *
 -280    503   1163|====     *
 -279   1863   1199|=========*=====
 -278   3633   1232|=========*==================
 -277   5968   1261|=========*====================================
```

```
 -276   7548   1285|==========*================================================
 -275   7363   1306|==========*=============================================
 -274   5222   1322|==========*============================
 -273   3441   1335|==========*================
 -272   2144   1344|==========*======
 -271   1442   1349|==========*=
 -270    827   1350|=======   *
 -269    762   1348|======    *
 -268    721   1342|======    *
 -267    496   1334|====      *
 -266    405   1323|====      *
 -265    462   1309|====      *
 -264    413   1292|====      *
 -263    426   1274|====     *
 -262    422   1253|====     *
 -261    401   1231|====     *
 -260    412   1207|====     *
 -259    430   1182|====     *
 -258    405   1156|====     *
 -257    464   1128|====    *
 -256    430   1100|====    *
 -255    428   1071|====    *
 -254    403   1042|====    *
 -253    434   1012|====   *
 -252    435    982|====   *
 -251    433    952|====   *
 -250    411    922|====   *
 -249    415    893|====  *
 -248    404    863|====  *
...continued...
 -153      0      9|*
```

```
 -152      1      9|*
 -151      0      8|*
>-150    104      -|=

% Statistical details of theoretical EVD fit:
              mu =  -269.6246
          lambda =     0.0507
chi-sq statistic = 89823.1562
  P(chi-square)  =          0

#Alignments
---------------------------------------------------------------------------

>Results for paa vs gi|78157278|gb|AAGE02008433.1| (reverse) [0]

genewisedb output
Score 494.73 bits over entire alignment.
This will be different from per-alignment scores. See manual for details
For computer parsable output, try genewisedb -help or read the manual
Scores as bits over a synchronous coding model

                    Alignment 1 Score 494.73 (Bits)

paa                 1 MKYCKF-CRKVVA----------VKLIHVPKCAIKRKLWEQSLGCSLGE
                       ++CK+  + VV           VK+   P    +++ W    G+S  E
                      CNWCKMVKCAVVSCSNYEGNKLKVKFFWFPNDEHRKSAWLRACGRSPLE
gi|78157278|gb-54936 tattaagatgggataatggaacagattttcaggccatgtcagtgaaccg
                      gaggattagcttgggaaagaatatattgtcaaaagaccgtgcggggcta
                      ctgtaggatggttcctcagtggaagccgtcttgtaaagggagtcgctag

paa                39 NSK--ICDTHFKASQWKSAE--KGQILKRRRLN-DAVPSKESEPEPEIV
```

```
                      ++  IC  HF + +++ +   +  +  ++RL   A+PS
                     SQHFKICSEHFCDEDYRLKDILLNTEWSKKRLKPGASPSL---------
gi|78157278|gb-54789 tcctaattgcttgggtatagactaagttaaccacggtctt
                     caatatgcaatgaaaagtaatttacagcaagtacgcccct
                     ggtcaatcatttcgccaaatttattggaaaggaacttaga

...continued...

paa              541 FSQLRQKAHGGVYDHPSPLQFKYRIRKYILGKSPEILKNKS
                     F+++R+K  GG++DHP+ L F YR+R+ ILG    +  + +
                     FGTIRSK--GGLHDHPTALEFTYRLRNSILGNIIDYFILIK
gi|78157278|gb-49904 tgaacta  ggtcgccagtgtatatcaaatgaaagttacaa
                     tgctgca  ggtaaaccctatcagtgagttgattaatttta
                     tgttgaa  aagcctccagactcagaccaatttatctatta

//
Gene 1
Gene 54936 49788
  Exon 54936 54647 phase 0
  Exon 51231 49788 phase 2
//

>Results for paa vs gi|78152614|gb|AAGE02012918.1| (forward) [1]

genewisedb output
Score 452.70 bits over entire alignment.
This will be different from per-alignment scores. See manual for details
For computer parsable output, try genewisedb -help or read the manual
Scores as bits over a synchronous coding model

                     Alignment 1 Score 452.70 (Bits)
```

```
paa                   1 MKYCKFCRKVVAVKLIHVPKCAIKRKLWEQSLGCSLGENSKICDTHFK-
                          + + + K V+ + I+++    K++   ++ +++L++  +I++T ++
                        SLRPETSTKTVNICNINSDVSELKKENIGYITTDQLAQFLEIFKTNYSC
gi|78152614|gb150530 atacgaaaaagaataaatggtgtaagaagtaaagctgcttgataaattt
                        gtgcacgcactatgatacatcataaaatgatccaatcattattacaacg
                        cgatatctagattttcttttcaaagacattaactggaataatcaattat

paa                  49 ---------ASQWKSAE-KGQIL--KRRRLN-DAVPSKESEPEPE-IVK
                                    +++ +  KG+++  K RRL  D  +++  E   +  +
                        YKKPLIPETKYEMEIVLKKGESFHFKPRRLSLDQKQKV--ELKIKELLE
gi|78152614|gb150677 taactacgaatgagagtaaggttctaccatttgcacag  gcaaagctg
                        aaacttcacaaatatttaagactatacggtctaaaaat  atataatta
                        tgacattatatagaatggataatttattaatataaaat  atataatga

...continued...

paa                 538 ENFFSQLRQKAHGGVYDHPSPLQFKYRIRKYILGKSPEILKNKS
                            E+FF+++R+K  GG++DHP+ L F YR+R+ ILG    +  + +
                            ERFFGTIRSK--GGLHDHPTALEFTYRLRNSILGNIIDYFILFK
gi|78152614|gb162600 gcttgaacta  ggtcgccagtgtatatcaaatgaaagttacta
                            agttgctgca  ggtaaaccctatcagtgagttgattaatttta
                            gcctgttgaa  aagcctccagactcagactaatttatctatta
```

B.4 *extract* Perl Script

We utilized many Perl and shell scripts to facilitate our search for transposable elements. A specifically useful script was our *extract* script. We wrote it to quickly extract pertinent information from the very large NCBI *Aedes aegypti* genome file. We provided it with the accession number, the start and end of the sequence, the

orientation, and the desired name and it pulled out the sequence from the NCBI file. If the orientation of the sequence was reverse (-1), it put the sequence in the forward orientation. Section B.4.1 shows the *extract* Perl script code. Section B.4.2 shows how we ran the *extract* script. The user is able to specify arguments and can also run the script repeatedly through shell scripts. Section B.4.3 shows an output file produced by the *extract* script. It first gives the label specified by the user and then the sequence itself, in the forward direction.

B.4.1 *extract* Perl Script Implementation

```
##read from command line
my $accession=$ARGV[0];
my $start=$ARGV[1];
my $end=$ARGV[2];
my $orientation=$ARGV[3];
my $name=$ARGV[4];
my $seq="";
my $temp="";

open(INPUT,"</Network/Servers/bio8.bio.nd.edu/Volumes/class_data/bios60579/
               data/genome/aaegypti/ncbi_aedes_aegypti_genome.fasta");
flock(INPUT,$LOCK);

while($line=<INPUT>) {
  if($line =~ />gi\|[0-9]*\|gb\|([A-Za-z0-9\.]*)\|./ ) {
    if($1 eq $accession) {
      $info=$line;
      while($line=<INPUT>) {
        if($line =~ /[ATGCatgc]+/) { $seq=$seq.$line; }
        else { cleanup(); exit; }
```

```
      }
    }
  }
}

flock(INPUT,$UNLOCK);
close(INPUT);

sub cleanup() {
  if($start>$end) { $temp=$start; $start=$end; $end=$temp; }

  $var=substr($seq,$start-1,$end-$start+1);

  if($orientation=="-1") { invrep(); }

  open(OUTDAT, ">$accession\_$name.txt");
  flock(OUTDAT,$LOCK);
  print OUTDAT "\>$name\n\n";
  print OUTDAT "$var\n";
  flock(OUTDAT,$UNLOCK);
  close(OUTDAT);
}

sub invrep() {
  $rev=reverse $var;
  $rev=~tr/GTACgtac/CATGcatg/;
  $var=$rev;
}
```

B.4.2 Sample *extract* Submission

```
perl extract.pl AAGE02001079.1 33524 34466 1 Mariner_Ele2
```

B.4.3 Sample *extract* Output File

```
>Mariner_Ele27

GATGATTGACGAAGCTTCGGTCAACTCGCTTGATCAACAAGCGAATCGCATCTTCTATT
CATTCACCCAGCTGTATGTGAAAGTCCCAAAAGCTTGAATAACACATTTGTAAATCGTAAAATGAGAAGG
AAAGGAAATTTATGATTTATGATATTTTTTTTTGCAATGAATGACTACTACAAATGTAATACACAACAAGA
AAGAAATATGTACAAGAAATGCAAATTGCAATAGAATTAAAATAAAATAAATGATTATATCAGTCTTCAC
CATCATGTGGTAAAAATCTATACTGTTATTTATTCTGTGTTGTTTGTCCTTCATTTAGAATTGGAACAAA
CTTTTGCTCATATTGTGGCGCTCCTTGTGGACGGATTTGGAAGTTCTTGGCGCCCACGTGTCGGGAATTT
TGTCAGCTTCACGTATGTATTTTTGACATTCCGCAAATCGACTGTACTTTGTAAACAATCAACATGGAAG
CCGAAAGAAGGGGAAAAATTGTGCACAGTTTTTTGGAAAATCCATTGTGGTCTGCATCTAGGCTGAACTT
TCCCAGAAATACCGTATGGCGCGTTATCAAACGGTATAAGGAAACATTGACGACGATTCGGAAGCCTCAA
GCCAATCGTCGGAGTGGAACTGTCGACCGGAAACTGCGTGGTAGGATTTTGAAGACGATTAAGAGGAAAC
CCAACCGTGATTTCGCCAAAAAATTCGGTGCTGCCCATAGTACCGTGAGGAGAACTCGACTCCGGGAAGG
AATCAAGTCGTATCGAGCTAGCCAACAGCCAAATCGGACCATAAAACAGAATAGTGTGGCCAAAATCCGT
GCTCGAAAGCTATACGACCAGGTGCTGACCAAGTTCGACGGGTGGCTTCTGATGGACGATGAAACCTATG
TCAAGGCTGACTTCGGGCAAATCCCAGGTCA
```

REFERENCES

1. AlphaStep. http://www.nd.edu/~nom/software/software.html.

2. L. Arthurs, P. Maurice, X. Xiang, R. Kennedy, and G. Madey. Agent-based stochastic simulation of natural organic matter adsorption and mobility in soils. In *Eleventh International Symposium on Water Rock Interaction*, June-July 2004.

3. R. Axtell, R. Axelrod, J. M. Epstein, and M. D. Cohen. Aligning simulation models: A case study and results. *Computational and Mathematical Organization Theory*, 1:123–141, 1996.

4. O. Balci. Validation, verification, and testing techniques throughout the life cycle of a simulation study. In *Proceedings of the 1994 Winter Simulation Conference*, pages 215–220, 1994.

5. O. Balci. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, chapter Verification, Validation, and Testing. John Wiley & Sons, New York, NY, 1998.

6. O. Balci. Verification, validation, and accreditation. In *Proceedings of the 1998 Winter Simulation Conference*, pages 41–48, 1998.

7. O. Balci. A methodology for certification of modeling and simulation applications. *ACM Transactions on Modeling and Computer Simulation*, 11(4):352–377, October 2001.

8. J. Banks and R. R. Gibson. Don't simulate when... 10 rules for determining when simulation is not appropriate. *IEE Solutions*, September 1997.

9. J. Banks and J. S. C. II. Introduction to discrete-event simulation. In *Proceedings of the 1986 Winter Simulation Conference*, pages 17–23, 1986.

10. J. Banks, J. S. C. II, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Pearson Education, Inc., Upper Saddle River, NJ, fourth edition, 2005.

11. Z. Bao and S. Eddy. Automated de novo identification of repeat sequence families in sequenced genomes. *Genome Research*, 12(8):1269–1276, August 2002.

12. Bioinformatics. http://en.wikipedia.org/wiki/bioinformatics.

13. B. Bioinformatics. http://biotech.icmb.utexas.edu/pages/bioinfo.html.

14. BioPerl. http://www.bioperl.org/wiki/main_page.

15. BLAST. http://www.ncbi.nlm.nih.gov/blast.

16. G. E. Box. Robustness in the strategy of scientific model building. In R. Launer and G. Wilkinson, editors, *Robustness in Statistics: Proceedings of a Workshop*, New York, NY, 1979. Academic Press.

17. Broad Institute *Aedes aegypti* FAQ. http://www.broad.mit.edu/annotation/disease_vector/aedes_aegypti/faq.html.

18. C++. http://www.cplusplus.com/.

19. S. Cabaniss. Modeling and stochastic simulation of nom reactions. available at http://www.nd.edu/~nom/Papers/WorkingPapers.pdf, 2002.

20. S. E. Cabaniss, G. Madey, L. Leff, P. A. Maurice, and R. Wetzel. A stochastic model for the synthesis and degradation of natural organic matter. part i. data structures and reaction kinetics. *Biogeochemistry*, 76:319–347, 2005.

21. L. Cary, M. Goebel, B. Corsaro, H. Wang, E. Rosen, and M. Fraser. Transposon mutagenesis of baculoviruses: analysis of *trichoplusia ni* transposon ifp2 insertions within the fp-locus of nuclear polyhedrosis viruses. *Virology*, 172(1):156–169, September 1989.

22. T. Cickovski, J. Hogan, and R. C. Kennedy. Analysis of *P* element transposon sequences in *Aedes aegypti*. available at http://www.nd.edu/ gmadey/bio05/Student_Papers/CickovskiHoganKennedy2.pdf, December 2005.

23. Clustal X. http://bips.u-strasbg.fr/en/documentation/clustalx/.

24. T. F. Cosimano and M. T. Gapen. Optimal fiscal and monetary policy with nominal and indexed debt. available at http://www.nd.edu/~tcosiman/Optimapolicy.pdf, September 2004.

25. T. F. Cosimano and M. T. Gapen. Solving ramsey problems with nonlinear projection methods. *Studies in Nonlinear Dynamics & Econometrics*, 9, 2005.

26. M. Coy and Z. Tu. Gambol and tc1 are two distinct families of dd34e transposons: analysis of the *Anopheles gambiae* genome expands the diversity of the is630-tc1-mariner superfamily. *Insect Molecular Biology*, 14(5):537–546, 2005.

27. D. A. Diener, H. R. Hicks, and L. L. Long. Comparison of models: Ex post facto validation/acceptance? In *Proceedings of the 1992 Winter Simulation Conference*, pages 1095–1103, 1992.

28. R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press, Cambridge, United Kingdom, 2003.

29. S. R. Eddy. Profile hidden markov models. *Bioinformatics Review*, 14(9):755–763, 1998.

30. Ensembl. http://www.ensembl.org.

31. *GeneWise*. http://www.ebi.ac.uk/wise2/documentation.html.

32. V. Grimm. Visual debugging: A way of analyzing, understanding, and communicating bottom-up simulation models in ecology. *Natural Resource Modeling*, 15(1):23–38, 2002.

33. GSL - GNU Scientific Library. http://www.gnu.org/software/gsl/.

34. I. Halachmi, A. Dzidic, J. Metz, L. Speelman, A. Dijkhuizen, and J. Kleijnen. Validation of simulation model for robotic milking barn design. *European Journal of Operational Research*, 134:677–688, 2001.

35. HMMER. http://hmmer.wustl.edu.

36. R. A. Holt, et al. The genome sequence of the malaria mosquito *Anopheles gambiae*. *Science*, 298(5591), October 2002.

37. Y. Huang. Infrastructure, query optimization, data warehousing and data mining for scientific simulation. Master's thesis, University of Notre Dame, September 2002.

38. Y. Huang and G. Madey. Autonomic web-based simulation. In *Proceedings of the 38th Annual Simulation Symposium*, April 2005.

39. Y. Huang, X. Xiang, G. Madey, and S. E. Cabaniss. Agent-based scientific simulation. *IEEE Computing in Science and Engineering*, pages 22–29, January-February 2005.

40. J2EE. http://java.sun.com/j2ee/1.4.

41. Java. http://java.sun.com.

42. N. C. Jones and P. A. Pevzner. *An Introduction to Bioinformatics Algorithms*. The MIT Press, Cambridge, MA, 2004.

43. K. Kaiser, J. W. Sentry, and D. J. Finnegan. Eukaryotic transposable elements as tools to study gene structure and function. In D. J. Sherratt, editor, *Mobile Genetic Elements*. Oxford University Press, 1995.

44. R. Kennedy, X. Xiang, G. Madey, and T. Cosimano. Verification and validation of scientific and economic models. In *The Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*. Argonne National Laboratory and The University of Chicago, October 2005.

45. R. C. Kennedy and J. Hogan. Using perl scripts to help identify *P* element transposon sequences in *Aedes aegypti*. available at http://www.nd.edu/ gmadey/bio05/Student_Papers/Report.pdf, December 2005.

46. R. C. Kennedy, X. Xiang, T. F. Cosimano, L. A. Arthurs, P. A. Maurice, and S. E. Cabaniss. Verification and validation of agent-based and equation-based simulations: A comparison. In L. Yilmaz, editor, *Proceedings of the 2006 Agent-Directed Simulation Symposium*. The Society for Modeling and Simulation International, April 2006.

47. J. P. Kleijnen. Statistical validation of simulation models. *European Journal of Operational Research*, 87:21–34, 1995.

48. J. P. Kleijnen. Verification and validation of simulation models. *European Journal of Operational Research*, 82:145–162, 1995.

49. LAPACK – Linear Algebra PACKage. http://www.netlib.org/lapack/.

50. A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, Boston, MA, third edition, 2000.

51. N. Lobo, A. Hua-Van, X. Li, B. Nolen, and J. M.J. Fraser. Germ line transformation of the yellow fever mosquito, *Aedes aegypti*, mediated by transpositional insertion of a *piggyBac* vector. *Insect Molecular Biology*, 11(2):133–139, April 2002.

52. C. Macal and M. North. Validation of an agent-based model of deregulated electric power markets. In *Proceedings of the North American Computational Social and Organization Science 2005 Conference*, Notre Dame, IN, June 2005.

53. Matlab. http://www.mathworks.com/products/matlab/.

54. B. McClintock. *The discovery and characterization of transposable elements: The collected papers of Barbara McClintock*. Garland Publishing, Inc., New York, NY, 1987.

55. M. McClure. The complexities of genome analysis, the retroid agent perspective. *Bioinformatics*, 16(2):79–95, February 2000.

56. E. Mongin, C. Louis, R. A. Holt, E. Birney, and F. H. Collins. The *anopheles gambiae* genome: an update. *Trends in Parasitology*, pages 49–52, February 2004.

57. D. M. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, Cold Spring Harbor, NY, second edition, 2004.

58. T. Naylor, J. Balintfy, D. Burdick, and K. Chu. *Computer Simulation Techniques*. John Wiley, New York, NY, 1966.

59. T. Naylor and J. Finger. Verification of computer simulation models. *Management Science*, 14:B92–B101, 1967.

60. NCBI: National Center for Biotechnology Information. http://www.ncbi.nih.gov.

61. S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, March 1970.

62. J. T. Oden, T. Belytschko, J. Fish, T. J. Hughes, C. Johnson, D. Keyes, A. Laub, L. Petzold, D. Srolovitz, S. Yip, and J. Bass. Simulation-based engineering science: Revolutionizing engineering science through simulation. http://www.ticam.utexas.edu/events/SBES_Final_Report.pdf, February 2006. National Science Foundation Blue Ribbon Panel.

63. T. I. Ören. Concepts and criteria to assess acceptability of simulation studies: A frame of reference. *Communications of the ACM*, 24(4):180–189, April 1981.

64. Perl. http://www.perl.org.

65. Phylogenetic tree. http://en.wikipedia.org/wiki/phylogenetic_tree.

66. R. H. Plasterk, Z. Izsvàk, and Z. Ivics. Resident aliens: the tc1/*mariner* superfamily of transposable elements. *Trends in Genetics*, 15(8), August 1999.

67. V. Pope and T. Davies. Testing and evaulating atmospheric climate models. *Computing in Science and Engineering*, pages 64–69, 2002.

68. Ramsey problem. http://en.wikipedia.org/wiki/ramsey_problem.

69. Repast. http://sourceforge.repast.net.

70. H. Robertson and D. Lampe. Recent horizontal transfer of a *mariner* transposable element along between diptera and neuroptera. *Molecular Biology and Evolution*, 12(5):850–862, 1995.

71. L. M. Rocha. From artificial life to semiotic agent models: Review and research directions. available at http://informatics.indiana.edu/rocha/ps/agent_review.pdf, Los Alamos National Laboratory Complex Systems Modeling Team, 1999.

72. G. Rubin and A. Spradling. Genetic transformation of drosophila with transposable element vectors. *Science*, 218(4570):348–353, October 1982.

73. F. Sanger, S. Nicklen, and A. Coulson. Dna sequencing with chain-terminating inhibitors. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 74, pages 5463–5467, December 1977.

74. R. G. Sargent. Some subjective validation methods using graphical displays of data. In *Proceedings of the 1996 Winter Simulation Conference*, pages 345–351, 1996.

75. R. G. Sargent. Verification and validation of simulation models. In *Proceedings of the 1998 Winter Simulation Conference*, pages 121–130, 1998.

76. R. G. Sargent. Some approaches and paradigms for verifying and validating simulation models. In *Proceedings of the 2001 Winter Simulation Conference*, pages 106–114, 2001.

77. A. Sarkar, R. Sengupta, J. Krzywinski, X. Wang, C. Roth, and F. Collins. *P* elements are found in the genomes of nematoceran insects of the genus *Anopheles*. *Insect Biochemistry and Molecular Biology*, 33(4):381–387, April 2003.

78. R. E. Shannon. Introduction to the art and science of simulation. In *Proceedings of the 1998 Winter Simulation Conference*, pages 7–14, 1998.

79. J. A. Shapiro. The discovery and significance of mobile genetic elements. In D. J. Sherratt, editor, *Mobile Genetic Elements*. Oxford University Press, 1995.

80. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

81. T. Tatusova and T. Madden. Blast 2 sequences, a new tool for comparing protein nucleotide sequences. *FEMS Microbiology Letters*, 174(2):247–250, May 1999.

82. VectorBase. http://www.vectorbase.org.

83. J. L. Weber and E. W. Myers. Human whole-genome shotgun sequencing. *Genome Research*, 7:401–409, 1997.

84. World Health Organization. http://www.who.int/mediacentre/factsheets/fs100/en/.

85. X. Xiang. Agent-based scientific applications and collaboration using java. Master's thesis, University of Notre Dame, May 2003.

86. X. Xiang, Y. Huang, G. Madey, S. Cabaniss, L. Arthurs, and P. Maurice. Modeling the evolution of natural organic matter in the environment with an agent-based stochastic approach. *Natural Resource Modeling*, 19(1), 2006.

87. X. Xiang, R. Kennedy, G. Madey, and S. Cabaniss. Verification and validation of agent-based scientific simulation models. In L. Yilmaz, editor, *Proceedings of the 2005 Agent-Directed Simulation Symposium*, volume 37, pages 47–55. The Society for Modeling and Simulation International, April 2005.

This document was prepared & typeset with pdfLaTeX, *and formatted with* NDdiss2$_\varepsilon$ *classfile (v1.0[2004/06/15]) provided by Sameer Vijay.*