RNA Evolution Simulator
User Manual Version 1.0

Kim Lab
University of Pennsylvania

July 16, 2005
Contents

1 Introduction 3

2 Models and Parameters 5
  2.1 RNA Representation .................................................. 5
  2.2 Mutation Events ......................................................... 7
    2.2.1 Neutral Substitution for One Nucleotide ......................... 7
    2.2.2 Advantageous Substitution ....................................... 8
    2.2.3 Detrimental Substitution ....................................... 8
    2.2.4 Insertion and Deletion ......................................... 8
    2.2.5 Compensatory Mutation ....................................... 9
  2.3 Mutation Rates and Parameters ..................................... 9
    2.3.1 Neutral Substitution .......................................... 9
    2.3.2 Non-Neutral Substitution ...................................... 10
    2.3.3 Insertion and Deletion ....................................... 11
    2.3.4 Compensatory Mutation ...................................... 11
  2.4 Rate Variation Across Sites ....................................... 12
  2.5 Stochastic Model .................................................... 12

3 Implementation 15
CONTENTS

3.1 Architecture of the simulator .................................. 15
3.2 An Algorithm for Tracing the Evolution History ............... 17

4 A Short Tutorial .................................................. 19
4.1 Compile the Programs .......................................... 19
4.2 An Example ...................................................... 19

A List of Files in the Simulator Source Code ..................... 22
Chapter 1

Introduction

The RNA Evolution Simulator is developed at Professor Junhyong Kim’s lab at the University of Pennsylvania. The simulator has three major features: (1) Sites in RNA sequence do not evolve independently, but are constrained by the secondary structure of the RNA molecule. (2) The simulator is capable of simulating six different classes of mutational events including point mutation, compensatory mutation of pairs, and insertion and deletion of subsequences. Rates of the six classes of mutational events are determined using Kimura’s theory of mutation by random drift. (3) Rate variation across sites is specifically modeled by a gamma distribution. (4) The simulator traces homologous sites in all simulated sequences (including ancestral sequences) and output the “true” alignment, hence facilitating simulation studies for phylogenetic reconstruction.

This manual describes the theoretical background of the evolutionary process simulated by our program and details of the simulator architecture. In Chapter 2, we describe the six classes of mutation events for a RNA molecule with secondary structure information; we also provide description on how the mutation rates are parameterized using results from Kimura’s neutral theory, and the stochastic process governing the mutational process. In Chapter 3, we outline the implementation of our simulator, including the program architecture and the algorithm that traces the true alignment during the simulation. Chapter 4 is a short tutorial on how to use the simulator.
CHAPTER 1. INTRODUCTION

The program is written in ANSI C++, and has been successfully compiled and tested in Redhat\textsuperscript{\textcopyright} AS 3.0 using g++-3.2, and in Windows\textsuperscript{\textcopyright} XP using Visual C++ 6.0.
Chapter 2

Models and Parameters

Our RNA Evolution Simulator simulates the evolution of RNA and its secondary structure based on well established models. The simulator models six mutation events that change the sequence and/or the structure, including point mutations, compensatory mutations of nucleotide pairs, insertions, and deletions. Instead of setting arbitrary rates for various mutational events, we adopted Kimura’s neutral theory to derive biologically reasonable ratios among these rates. Unlike the usual \textit{i.i.d} models for sequence mutation, in our model, each site is constrained by RNA secondary structure. Moreover, our simulator is capable of modeling the rates across sites model even for the \textit{same} type of mutational events to reflect possible variation across sites, due to factors other than the secondary structural constraints (for example, the anticodon in a tRNA does not affect its secondary structure, but is highly conserved). In this chapter, we present a brief review of the evolutionary models and how the relationships among rates for different kinds of mutational events are derived.

2.1 RNA Representation

Our simulator adopts a parenthesized format for the RNA secondary structure as follows. The first line is a description of the molecule; it always starts with ">". The second line is the sequence of the RNA molecule. The sequence uses the standard single letter code (AUCG). The third line is the
CHAPTER 2. MODELS AND PARAMETERS

> Ecoli-O157-Wisc.trna75 (3314857–3314782)
> GGGGCTATAGCTCAGCTGGAGAGCGCTTGCATGGCATGCAAGAGTCAGCGGTTCGATCCCGCTTAGCTCCACCA

(((.((.((.((.((.((.((.((.((.((.((.((.((.((.((.((.((.((.((.((.((.((.((.

(b)

Figure 2.1: Example of the parenthesized format. (a) The sequence and the secondary structure of an Alanine (GGC) tRNA of the E. Coli (strand O157:H7) from the genomic tRNA database (http://lowelab.ucsc.edu/GtRNAdb/). (b) The corresponding secondary structure. The figure is generated using RNAPlot from the Vienna package (http://www.tbi.univie.ac.at/~ivo/RNA/).

The secondary structure of the RNA molecule. It has the same length as the second line, so positions in the two sequences are perfectly matched. The structural representation uses three kinds of letters (a left parenthesis, a right parenthesis, or a dot):

1. A dot (.) is an unmatched nucleotide.

2. The left and right parentheses are nested and balanced. Each pair of left-right parentheses represents a pairing of the corresponding nucleotides.

Figure 2.1 is an example of the parenthesized format.
2.2 Mutation Events

The simulator models six distinct classes of events:

1. Single nucleotide neutral substitution.
2. Single nucleotide advantageous substitution.
3. Single nucleotide detrimental substitution.
4. Insertion of one or more nucleotides (up to 15 at a time).
5. Deletion of one or more nucleotides (up to 15 at a time).
6. Compensatory mutation of two nucleotides that are paired in the secondary structure.

A single insertion/deletion event can introduce/remove up to 15 nucleotides. A brief explanation of events is as follows (details with diagrams can be found in the source code).

2.2.1 Neutral Substitution for One Nucleotide

The event changes a single nucleotide, and is followed by possibly dissociation/regaining of pairing in the secondary structure. The following four scenarios belong to this class:

1. A paired nucleotide changes to another nucleotide without affecting the pairing (e.g. G-C → G-U).
2. An unpaired nucleotide changes to another unpaired nucleotide (for example, the substitution occurs in a loop or a bulge).
3. Two paired nucleotides at the end of a stem dissociate after one nucleotide is changed (also called loop extension).
4. Two unpaired nucleotides at the end of an internal loop or an end loop associate (also called stem extension).
CHAPTER 2. MODELS AND PARAMETERS

We assume these four scenarios do not change the fitness of the RNA molecule (this assumption affects how the rate of the mutational event is determined).

2.2.2 Advantageous Substitution

The event changes a single nucleotide, and is followed by possibly dissociation/regaining of pairing in the secondary structure. The following two scenarios belong to this class:

1. An unpaired nucleotide inside a stem (i.e. in a bulge) becomes paired with another unpaired nucleotide after the nucleotide is changed.

2. Two paired nucleotides neither of which is adjacent to another paired nucleotide become unpaired after the substitution to one of the two nucleotides.

We assume these two scenarios increase the fitness of the RNA molecule.

2.2.3 Detrimental Substitution

The event changes a single nucleotide, and is followed by dissociation of pairing in the secondary structure. The following scenario belongs to this class:

1. A paired nucleotide in the interior of a stem changes to an unpaired nucleotide after one of the two nucleotides is changed.

We assume the scenario decrease the fitness of the RNA molecule.

2.2.4 Insertion and Deletion

An insertion inserts a contiguous sequence of up to 15 nucleotides to a single position in the RNA sequence, whereas a deletion deletes a contiguous sequence of up to 15 nucleotides in the RNA sequence. The distribution of the insertion/deletion length follows a power law distribution; more details can be found in the later section.
2.2.5 Compensatory Mutation

A compensatory mutation happens when two paired nucleotides change simultaneously through an unpaired intermediate state so the pairing status is preserved.

2.3 Mutation Rates and Parameters

We derive the relations among rates for various mutational events based on Kimura’s neutral theory [2] [3]. We make the following assumptions:

1. The population is diploidy and the mating is random. The effective size of population is $N_e$ ($N_e < \infty$) (i.e. there are $2N_e$ alleles).

2. None of the mutations is lethal, though some may be subject to selection.

3. Each nucleotide mutates at rate $\mu$ per generation.

4. We assume the selection coefficient $s$ satisfies $s \gg \mu > 0$.

5. The fitness for advantageous/detrimental mutation is additive: Assuming XX has fitness 1 and a mutation from X to x is advantageous, then Xx and xx have their respective fitness $1 + s$ and $1 + 2s$ ($0 < 2s < 1$). If the mutation is detrimental, then the respective fitness for Xx and xx are $1 - s$ and $1 - 2s$.

6. In a compensatory mutation, the fitness for X-Y and x-y (the two paired states) is 1, X-y and x-Y (the two possible unpaired intermediate states) is $1 - s$ ($0 < s < 1$).

2.3.1 Neutral Substitution

If the substitution is neutral, it is well known the substitution rate is

$$r_{neutral} = \mu_0 \times p_0 = 2N\mu \times \frac{1}{2N} = \mu$$
where \( \mu_0 \) is the probability that a mutation happens in one generation, and \( p_0 \) is the probability that fixation happens for a neutral mutation.

### 2.3.2 Non-Neutral Substitution

Shown by Kimura [?], the probability of fixation for an advantageous mutation is

\[
p = \frac{1 - \exp(-2Neq)}{1 - \exp(-2Ns)}
\]

Here the initial allele frequency for allele \( a \) is \( q = \frac{1}{Ne} \), and the fitness for \( XX, Xx, xx \) are 1, 1 + \( s \), and 1 + 2\( s \) (0 < \( s \) < 1). Therefore, the rate for an advantageous substitution is

\[
r_{\text{advant}} = Ne \mu \times p = \frac{1 - \exp(-2s)}{1 - \exp(-2Ne s)}
\]

For a detrimental substitution, the fixation probability is

\[
p = \frac{1 - \exp(2Neq)}{1 - \exp(2Ne s)}
\]

Consequently, the rate for a detrimental substitution is

\[
r_{\text{detr}} = Ne \mu \times p = \frac{1 - \exp(2s)}{1 - \exp(2Ne s)}
\]
CHAPTER 2. MODELS AND PARAMETERS

2.3.3 Insertion and Deletion

Assume the rates for insertions and deletions are lower than that for neutral substitutions, the rates for insertions and deletions can be modeled by

\[ r_{\text{insert}} = r_1 \times \mu \]
\[ r_{\text{delete}} = r_2 \times \mu \]

where \( r_1, r_2 < 1 \). The size of an insertion/deletion can be up to 15 in our simulation. The frequency of the indel length \( k \) follows a truncated power law distribution based on [1]:

\[ f(k) = mk^{-1.7}, 1 \leq k \leq 15 \]

where \( m = (\sum_{i=1}^{15} 5i^{-1.7})^{-1} \) is the normalization factor.

2.3.4 Compensatory Mutation

Assume \( s \gg \mu > 0 \), the fixation probability for compensatory mutation is

\[
p_2 = N_c \frac{8N_c\mu^2 - s - 3\mu}{s + 3\mu} e^{\frac{2s(N_c - 1)}{s + 3\mu}} \frac{\text{hypergeom}\left(1, \frac{2(4N_c\mu^2 - s - 3\mu)}{s + 3\mu}, \frac{8\mu s}{s + 3\mu}\right)}{\text{hypergeom}\left(1, \frac{2(4N_c\mu^2 - s - 3\mu)}{s + 3\mu}, \frac{8N_c\mu s}{s + 3\mu}\right)}
\]

where \( \text{hypergeom} \) is the hypergeometric distribution. The rate for a compensatory mutation is

\[ r_{\text{compen}} = N_c \mu^2 p_2 \]
CHAPTER 2. MODELS AND PARAMETERS

2.4 Rate Variation Across Sites

Based on [4], we assume the rates across sites follow a gamma distribution. The density function for a general gamma function is

$$f_X(x) = \frac{\lambda^r}{\Gamma(r)} x^{r-1} e^{-\lambda x} \quad x > 0$$

with positive parameters $r$ and $\lambda$. The mean and variance of $X$ are

$$E(X) = \frac{r}{\lambda}$$

$$Var(X) = \frac{r}{\lambda^2}$$

Assume $r = \lambda$ (so $E(X) = 1$), the density function can be reduced to

$$f_X(x) = \frac{r^r}{\Gamma(r)} x^{r-1} e^{-rx}$$

2.5 Stochastic Model

The model consists of three sets of parameters:

1. An edge-weighted rooted phylogenetic tree.

2. A RNA sequence $R$ in parenthesized format and serves as the root sequence, from which all descendant sequences and structures are derived during the simulation.

3. Parameters for the mutational rates: $\mu, N_e, s, r_1, r_2$.

Based on results in the preceding sections, rates of mutation for the six classes of events are computed using the five supplied parameters and kept constant throughout the simulation. Each site in $R$ is assigned a mutational weight by the simplified gamma distribution to simulate rate variation across sites; the weight is not changed during the simulation. The mutational weight of
CHAPTER 2. MODELS AND PARAMETERS

a newly inserted site is the average weight of its two neighbors, or the weight of its sole neighbor if insertion occurs at either end of the sequence. Notice this is a random scheme that does not take into account any structural, functional or thermodynamic information.

Each edge in the phylogenetic tree has an edge length $l$; this is the expected number of mutational events per site along the edge. Given sequence length $k$ at the ancestral end of the edge, the product $lk$ is the expected number of mutations from an ancestor sequence to its immediate child sequence. Between these two sequences are multiple intermediate sequences, such that exactly one mutation occurs between two successive sequences. Throughout the time, mutation occurs randomly, and the sequence of mutational events follow a modified Poisson process with parameter $\lambda$, the expected number of mutations occurring per generation. Parameter $\lambda$ is the summation of the expected number of the six events per generation, as computed by

$$\lambda = \sum_{i=1}^{6} r_i \times n_i$$

where $r_i$ is the mutation rate for event $i$, $n_i$ is the number of sites where an event from class $i$ can operate on (for example, compensatory mutations only happen to paired sites). The mutation does not stop until

$$\frac{l \times k}{\lambda}$$

generations have passed.

Under this Poisson process, the time $t$ between successive events, in multiple of generations, follows an exponential distribution with parameter $\lambda$. For each edge, let $T$ as the amount of time passed in the simulation along the edge ($T = 0$ for the ancestral end of the edge). Once a mutation occurs, we choose one of the six classes randomly according to their weighted rates: the probability the next mutation belongs to class $j$ is

$$\frac{w_j r_j n_j}{\sum_{i=1}^{6} w_i r_i n_i}$$

where $w_j$ is the weight summation of all sites that can have mutation event $j$. Once a mutation event is chosen, a site capable of having this
event is picked up with probability proportional to its weight. We then increase $T$ by $t$, and update $n_i$ for each class $i$ and $\lambda$. Since $\lambda$ likely changes after each mutation, the stochastic process is not a Poisson process (but similar to one).

Simulation stops for the edge when

$$T > \frac{l \times s}{\lambda}$$

at which we assign the resulting sequence to be the sequence for the child end of the edge.
Chapter 3

Implementation

In this chapter, we describe the implementation of the simulator in more detail. We first present the architecture of the simulator, then describe the algorithm for tracing evolution history and its utilization of creating “true” alignment of extant sequences and ancestor sequences.

3.1 Architecture of the simulator

The simulator program is designed using the Object-Oriented Programming (OOP) model; it is coded in C++ (Figure 3.1). Two parsers, RNAWithPairingParser and RootedTreeParser, parse the input RNA and tree files to build two objects, RNAWithPairing and RootedTree, respectively. Each node in the RootedTree is then associated with RNAWithPairing (the sequence together with the secondary structure, i.e., the object being mutated), along with other auxiliary information to form an EvolutionTree. The third input file containing all five parameters is used to initiate RNAWithPairingEvolModel, the core of the program. MutateOnTree simulates the mutational stochastic process along the evolutionary model tree, generates the results, and store them in several output files.

One advantage of the proposed architecture is that EvolutionTree is composed of RNAWithPairing, the object being mutated, and RootedTree, a general data structure for the tree topology. The
structure separates RNA-specific mutational operations from tree manipulations. Correspondingly, the class \texttt{EvolutionTree} and \texttt{MutateOnTree} are implemented as template classes. We adopt the design hoping that in the future, mutable objects other than \texttt{RNAWithPairing} can easily be developed so as to construct different \texttt{EvolutionTree} classes that simulate other kinds of evolutionary processes. The appendix contains a list of all files with description.

Figure 3.1: Simulator Architecture.
CHAPTER 3. IMPLEMENTATION

3.2 An Algorithm for Tracing the Evolution History

The data structures and algorithms for a rooted phylogenetic tree are fairly standard and are omitted here. The algorithm for RNA evolution has been documented in the last chapter; additional details can be found in the source files. In this section, we briefly describe our algorithm for reconstructing the true alignment of all RNA sequences in the simulator.

Given a root RNA sequence of length $n$, we encode its nucleotides for beginning to end by unique identifiers from 1 to $n$. The identifiers are preserved as the simulator descends along the evolutionary tree. The following are possible scenarios regarding site identifiers:

1. Substitution of a nucleotide will not change its identifier.

2. If a nucleotide is deleted, its identifier is simply removed from the identifier sequence.

3. For any inserted nucleotides, we assign new identifiers as pairs: assume a nucleotide is inserted between two nucleotides with identifier $I_1$ and $I_2$, respectively, this nucleotide’s identifier is $(I_1,I_2)$. For example, if a nucleotide is inserted between nucleotide 3 and 4, then it gets (3, 4). If another nucleotide is inserted after this one, it gets ((3, 4), 4). If a third nucleotide is inserted between these two newly inserted nucleotides, its identifier is ((3, 4), ((3, 4), 4)).

Clearly, the number of right parenthesis indicates successive insertions at the same site of the root sequence. We denote an identifier with two or more left parenthesis as a complex identifier, and an identifier with only one left parenthesis as a simple identifier; jointly as composite identifier.

Using the above scheme for identifier handling, we can trace the evolution history and recover the true alignment of all sequences (see Figure 3.2). The alignment tracing algorithm is implemented as a separated program.
**Input:** $m$ sequences with the associated identifiers  
**Output:** True alignment of the $m$ sequences generated from the simulation.  
**Algorithm:**

1. for $i=1$ to $m$: extract composite identifiers with maximal value $i$ and store them in $\text{comp}[i]$  
   \% sort composite identifiers

2. for $i=1$ to $m$:
   
   (a) Extract simple identifiers from $\text{comp}[i]$ and sort them incrementally by length
   
   (b) for all identifiers with same length, place the one with smaller minimal value ahead of others
   
   (c) set $j = 2$
   
   (d) while $\text{comp}[i]$ is not empty:
      
      i. extract complex identifiers with $j$ left parenthesis
      
      ii. insert them into sorted identifiers by recognizing the two components of a complex identifier

Figure 3.2: The algorithm for tracing the true alignment from the evolutionary history.
Chapter 4

A Short Tutorial

This chapter provides a tutorial on how to use the simulator.

4.1 Compile the Programs

The source code for three programs are provided, the simulator, the alignment program and a program for computing the maximal and minimal pairwise similarity of a sequence set. In Windows, open the project workspace .dsw file to build the executable. In Linux environment, use g++ to compile the source code, for example, to compile the simulator, use command

[localhost]$ g++ *.cpp -o rnasim

4.2 An Example

The simulator comes with a set of files used in this example, they are listed below. Users shall supply files respecting the corresponding formats.

1. parameters.txt: the parameters in sequential order are $N_e$, $s$, $\mu$, $r_1$, $r_2$, one parameter per line.
2. midtree.txt: a tree with 100 taxa in Newark. Labels of the interior vertices are optional.

3. rootrna.txt: root rna sequence in parenthesis format

To run the simulator, use command

```
[localhost]$ rnasim parameters.txt midtree.txt 1.0 rootrna.txt seq.out iden.out pair.out label.out tree.out
```

where 1.0 is the scaling factor for tree.txt, each edge in the tree is scaled by this number; The next three output files contain sequences, identifiers, pairing information, respectively, for all leaf and internal sequences; one line per sequence and is in the order listed in label.out. Note all internal vertices are renamed systematically with a starting “I”. The names of leaf vertices are the same as input tree file. User can write script to extract a subset of sequences, say, for alignment program. The scaled tree with new vertex names is recorded in tree.out, also in Newick format.

To run the alignment program, issue command

```
[localhost]$ truealign seq.out iden.out label.out aligned.out
```

where seq.out, iden.out and label.out are from the simulator, aligned.out contains aligned sequence with equal length, indels are denoted by dashes, the file is in Nexus format.

To check the similarity of aligned sequence, run command

```
[localhost]$ checkalign aligned.out
```

it reports the maximal and minimal sequence similarity, the program is easily modified to output all pairwise similarity, thus generating the distribution of all pairwise similarites. Be aware that or large dataset, this program requires a lot of memory and CPU time.
Bibliography


719, 1962.


Appendix A

List of Files in the Simulator Source Code

A list of all files in the simulator program:

1. rnasim.cpp: main program.

2. RNAWithPairing.h: head file of RNAWithPairing.cpp

3. RNAWithPairing.cpp: implementation of class RNAWithPairing

4. RNAWithPairingParser.h: head file of RNAWithPairingParser.cpp

5. RNAWithPairingParser.cpp: implementation of class RNAWithPairingParser

6. RootedTree.h: head file of RootedTree.cpp

7. RootedTree.cpp: implementation of class RootedTree

8. RootedTreeNode.h: head file of RootedTreeNode.cpp

9. RootedTreeNode.h: implementation of class RootedTreeNode
APPENDIX A. LIST OF FILES IN THE SIMULATOR SOURCE CODE

10. RootedTreeParser.h: head file of RootedTreeParser.cpp

11. RootedTreeParser.cpp: implementation of class RootedTreeParser

12. RNAWithPairingEvolModel.h: template class of RNAWithPairingEvolModel

13. Evolution.h: template class EvolutionTree and template class MutateOnTree

14. Utility.h: head file of Utility.cpp

15. Utility.cpp: implementation of various functions called by the above classes/modules.

16. Global.h: global definition