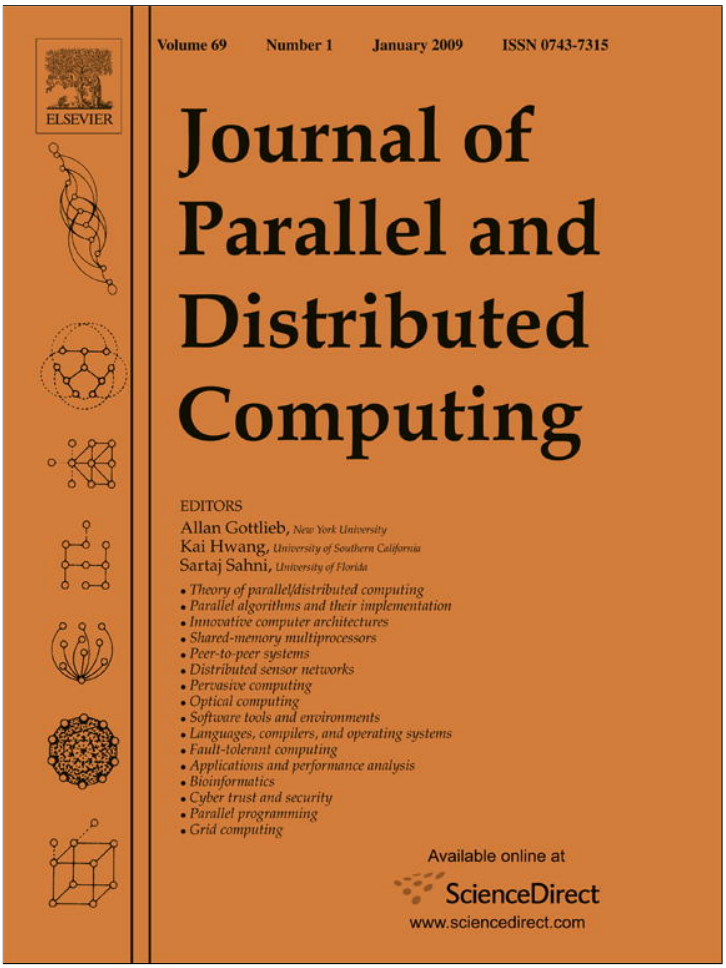


J. Parallel Distrib. Comput. 69 (2009) 80–90

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Analysis of DNA sequence transformations on grids

Yadnyesh Joshi, Sathish Vadhiyar*

Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore - 560012, India

ARTICLE INFO

Article history:
Received 1 December 2007
Received in revised form
26 April 2008
Accepted 6 June 2008
Available online 29 June 2008

Keywords:
Bioinformatics
Biology and genetics
Grid computing
Grids
Grid applications
Distributed systems
Client/Server
Distributed applications

ABSTRACT

Study of the evolution of species or organisms is essential for various biological applications. Evolution is typically studied at the molecular level by analyzing the mutations of DNA sequences of organisms. Techniques have been developed for building phylogenetic or evolutionary trees for a set of sequences. Though phylogenetic trees capture the overall evolutionary relationships among the sequences, they do not reveal fine-level details of the evolution. In this work, we attempt to resolve various fine-level sequence transformation details associated with a phylogenetic tree using cellular automata. In particular, our work tries to determine the cellular automata rules for neighbor-dependent mutations of segments of DNA sequences. We also determine the number of time steps needed for evolution of a progeny from an ancestor and the unknown segments of the intermediate sequences in the phylogenetic tree. Due to the existence of vast number of cellular automata rules, we have developed a grid system that performs parallel guided explorations of the rules on grid resources. We demonstrate our techniques by conducting experiments on a grid comprising machines in three countries and obtaining potentially useful statistics regarding evolutions in three HIV sequences. In particular, our work is able to verify the phenomenon of neighbor-dependent mutations and find that certain combinations of neighbor-dependent mutations, defined by a cellular automata rule, occur with greater than 90% probability. We also find the average number of time steps for mutations for some branches of phylogenetic tree over a large number of possible transformations with standard deviations less than 2.

1. Introduction

Study of the evolution of different species or organisms is important to biologists since it has practical applications including drug discovery, population monitoring and management [26]. Availability of DNA sequence databases [21] in the last few decades has enabled the study of evolutions at the molecular level. In terms of molecular biology, evolution can be viewed as a mutation event in which a particular DNA segment of an organism undergoes change. During evolution, a DNA segment consisting of a sequence of purines and pyrimidines (also called base-pairs) changes to a different sequence of base-pairs.

Evolution is visualized with the help of phylogenetic trees corresponding to a set of organisms. Phylogenetic trees give a picture of relatedness between various organisms. A branch in a *rooted* phylogenetic tree connecting ancestor and progeny indicates that one sequence (progeny) has evolved from the other (ancestor). While phylogenetic trees constructed out of existing packages [18] give an overall picture of the relationships, they do not give fine-level details of the way evolution might have progressed. In particular, the trees do not convey the possible effects of neighboring base-pairs on mutations of a base-pair in a DNA sequence. The trees also do not contain information about the number of time steps required for an ancestor–progeny transformation. The intermediate nodes of the trees have hypothetical DNA sequences in which the base-pairs of some segments are not known.

Some of the studies on the evolutions of species using DNA sequences have found that the mutation of a particular DNA segment is affected by its neighboring segments [2,3,11,16, 23]. While the effects of some neighboring base-pairs on the evolution of a DNA segment is known, there has been very little work [4,24] that comprehensively analyzes the effects of different neighborhoods on the evolution. Cellular automata can be used to comprehensively model and analyze these neighbor-dependent mutations. Cellular automata have been used to model physical, economical, and sociological systems [9]. They are also increasingly used for a variety of applications in bioinformatics such as predicting protein sub-cellular location and protein secondary structure [10,14,28,30]. Cellular automata have replaced partial differential equations in the area of system modeling fairly successfully. Since partial differential equations have been used for modeling evolution [20], cellular automata can also be used for successfully modeling evolution and analyzing DNA mutations. Cellular automata evolve over time through various *rules* which express the change in state of a cell in terms of its neighboring cells.

* Corresponding author.
E-mail address: vss@serc.iisc.ernet.in (S. Vadhiyar).

0743-7315/$ – see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2008.06.012

Rules found by modeling DNA mutations using cellular automata can give useful insights into the effects of neighboring base pairs on the mutations of DNA segments.

In this work, we attempt to give fine-level details of a phylogenetic tree using cellular automata. Our work is primarily intended for analyzing the cellular automata rules that may have been used for neighbor-dependent mutations of DNA sequences in the individual steps of transformation from an ancestor to a progeny in the phylogenetic tree. In addition, we also resolve the number of time steps needed for the transformation and the base pairs in some segments of the DNA sequences corresponding to the intermediate nodes.

Modeling DNA sequence mutations or transformations using cellular automata where each cell can assume one of 4 possible states requires exploration of a huge search space and needs a large number of computing cycles. Grid computing has been used for conducting large parameter search studies [5,15,22]. We have developed a grid system to conduct parallel guided explorations of cellular automata rules for sequence transformation. Our system includes techniques for load-balancing and fault-tolerance, a database for storage and retrieval of sequence transformation details, and a suite of statistics collection programs for deriving various statistics related to transformations. We demonstrate our techniques by conducting experiments on a grid comprising machines in three countries and obtaining potentially useful statistics for three HIV genomic segments corresponding to *gag*, *gagpol* and *env*.

Following are the primary contributions of work:

(1) Modeling DNA sequence mutations using cellular automata to determine the possible rules for neighbor-dependent mutations for a particular phylogenetic tree on computational grids. We also resolve other uncertainties related to the phylogenetic tree, namely, the number of time steps for evolution of a progeny from an ancestor and unknown base-pairs in intermediate sequences.
(2) Building a grid system for parallel exploration of vast number of cellular automata rules for transformations.
(3) Using the system to collect potentially useful statistics for three HIV sequences. In particular, we verified the phenomenon of neighbor-based mutations for some steps of evolution, found that some combinations of neighbor-based mutations corresponding to a rule occur with more than 90% probability, determined the average number of time steps for mutations from ancestor to progeny sequences of a phylogenetic tree with less than 2 standard deviations, and resolved certain segments of incomplete intermediate sequences.
(4) Finally, showing the benefits of grid computing for the sequence transformation application.

Section 2 gives necessary background for our work including cellular automata and phylogenetic trees. In Section 3, we compare and contrast our work with the other efforts on evolution studies. In Section 4, we detail our rule development and analysis strategies, and our assumptions regarding the rates of evolution. Section 5 explains the techniques used for searching parameters related to sequence transformations on grid platforms. Section 6 enumerates the statistics regarding transformations that we collected using a set of collection programs. Section 7 gives some of the promising results obtained for the gag HIV sequence on a prototype grid framework. Section 8 discusses various possible extensions to our work, Section 9 summarizes our work, and Section 10 gives future directions.

2. Background

In this section, we give a brief background on cellular automata, the relationship between cellular automata and DNA evolution, and phylogenetic trees.

Time steps	Cells						
0	0	1	0	1	0	0	0
1	1	1	0	1	1	0	0
2	0	0	0	0	0	1	1

Fig. 1. Evolution of Cellular Automata through time steps.

0	0	0	0	0	1	0	1	0	0	1	1
0			1			1			0		

1	0	0	1	0	1	1	1	0	1	1	1
1			0			0			1		

Fig. 2. Rule that governs the evolution of cellular automata shown in Fig. 1.

2.1. Cellular Automata

A *cellular automaton* is a regular array of identical finite state automata where the next state of an array element is determined solely by its current state and the state of its neighbors. The change of state of the array element is defined by a cellular automata rule [29]. As an example, the evolution of one dimensional cellular automata is shown in Fig. 1. Each cell can have one of the two possible states — 0 or 1. In this example, the neighborhood size is 1, i.e. the state of each cell at the next time step is dependent on the state of that cell, the state of its single left neighbor and the state of its single right neighbor. The rule that governs this evolution is depicted in Fig. 2.[1] As can be seen, the cellular automata consists of eight *transitions* corresponding to eight possible left hand side states. The number of rules possible for this particular cellular automaton is thus $2^8 = 256$. In general, one dimensional cellular automata with two states and with neighborhood size n consists of $2^{2 \cdot n+1}$ rules with each rule consisting of $2^{2 \cdot n+1}$ transitions.

2.2. DNA and Cellular Automata

DNA is a nucleic acid that contains the genetic instructions for the development and function of living things. The building blocks of the DNA polymer are *nucleotides*, which in turn consist of a *phosphate* group, a *sugar ring*, and either a *purine* or a *pyrimidine* base group. The purines are guanine (G) and adenine (A) and the pyrimidines are thymine (T) and cytosine (C). A sequence of these bases forms a strand or a sequence. We can view a DNA strand as a line of cells with each cell having one of the four values (A,G,C or T). This strand is copied exactly to produce another identical strand in the process of DNA replication. Sometimes, mutation occurs during replication giving rise to a DNA strand that is different from the original strand. These mutations are the basic mechanisms of evolution.

There are strong indications that mutations of DNA base-pairs are affected by neighboring base-pairs [2,3,11,16,23]. The exact effect of the neighboring base-pairs on the mutation of an individual base-pair is still unknown. We make an attempt to find out this relationship by modeling DNA as cellular automata in

[1] In our work, we use cellular automata with wrap-around, i.e. the left neighbor of the first cell is the last cell and the right neighbor of the last cell is the first cell.

Table 1
Left-hand sides of 64 transitions of cellular automata with neighborhood size of 1

S.No.	LHS	S.No.	LHS	S.No.	LHS	S.No.	LHS
1	AAA	17	AAC	33	AAG	49	AAT
2	CAA	18	CAC	34	CAG	50	CAT
3	GAA	19	GAC	35	GAG	51	GAT
4	TAA	20	TAC	36	TAG	52	TAT
5	ACA	21	ACC	37	ACG	53	ACT
6	CCA	22	CCC	38	CCG	54	CCT
7	GCA	23	GCC	39	GCG	55	GCT
8	TCA	24	TCC	40	TCG	56	TCT
9	AGA	25	AGC	41	AGG	57	AGT
10	CGA	26	CGC	42	CGG	58	CGT
11	GGA	27	GGC	43	GGG	59	GGT
12	TGA	28	TGC	44	TGG	60	TGT
13	ATA	29	ATC	45	ATG	61	ATT
14	CTA	30	CTC	46	CTG	62	CTT
15	GTA	31	GTC	47	GTG	63	GTT
16	TTA	32	TTC	48	TTG	64	TTT

which DNA mutations are governed by the cellular automata rules. The DNA molecule can be viewed as a one-dimensional cellular automaton, with four states per cell, corresponding to each of the base-pairs. Thus, the base of this cellular automata is 4. The number of transitions in a given rule is 4^{2n+1} where n is the number of left/right neighbors. The total number of rules that govern DNA mutations is thus $4^{4^{2n+1}}$. Thus, finding rules that could have been followed during evolutions requires exploration of huge search space.

In this work, we consider only those rules with neighborhood size of one, i.e. the transition of a base-pair in a DNA sequence during evolution depends on the base-pair and its left and right neighboring base-pairs. The 64 left-hand sides of the transitions corresponding to the neighborhood size of 1 are depicted in Table 1. The right-hand side of each transition can be one of 4 base-pairs, giving rise to a total of 4^{64} rules. We assume that during one evolution step, a single rule is applied for the entire sequence, i.e. if a base pair X appears in two different positions of the current sequence and is flanked by the same left and right neighbors in both the positions, then this base pair X transitions to Y in both positions during the next mutation step. The transition X to Y is governed by the rule applied at the mutation step. Although these assumptions do not encapsulate the myriad mechanisms that could have been followed during evolution, the assumptions are reasonable since there is evidence that a base-pair is impacted more by its immediate neighbors than its farthest neighbors [2,11].

2.3. Phylogenetics

In biology, *phylogenetics* is the study of evolutionary relatedness among various groups of organisms. A phylogenetic tree, also called an *evolutionary tree* or a *tree of life*, is a tree showing the evolutionary interrelationships among various species or other entities that are believed to have a common ancestor. The leaves of the tree represent various organisms, species, or genomic sequences. An internal node of the tree represents an abstract organism (species, sequence) whose existence is presumed and whose evolution led to the organisms at the leaves. A rooted phylogenetic tree has a root that corresponds to the most recent common ancestor to all the sequences under consideration.

Various efforts have been made to construct phylogenetic trees for a given set of DNA sequences [17,25]. However, there are certain uncertainties associated with these phylogenetic trees. The reconstruction of the sequences corresponding to intermediate nodes are not complete in the phylogenetic trees. There are several positions in the intermediate sequences where the exact base-pairs are not known. The number of time steps required for the mutations to occur is also not known explicitly. Finally, these trees do not provide any indication of neighbor-dependent mutations that may have been followed during the evolution of different sequences in a given tree. Our work tries to resolve these uncertainties.

3. Related work

There have been number of studies on the evolution of DNA sequences [2,3,11,13,16,23–25]. The work by Korber et al. [13] studies the evolution of HIV sequences using the molecular clock assumption; this hypothesis postulates that molecular change is a linear function of time and that substitutions accumulate according to a Poisson distribution.

There are a number of studies that analyze the impact of neighboring bases on the mutation of a particular base. The work by Bulmer [3] finds that there is a marked increase in the frequency of transitions from the doublet CG. There are also some smaller effects of neighboring bases on the frequencies of transitions from adenine and thymine. The work also determines that the transition frequency from either of these bases is reduced by having G on the right (or C on the left) and increased by having T on the right (or A on the left). Hess [11] also concludes that substitution rates, representing averages over those for different regions of the genome, are distributed over a 60-fold range with strong biases in certain neighbor-pair environments. Studies indicate that substitution rates vary for the same base-pair for different neighbor-pair environments. Arndt et al. [2] introduces a model of DNA sequence evolution that can account for biases in mutation rates that depend on the identity of the neighboring bases. All the above efforts clearly indicate that neighboring bases have some effect on the mutation of a particular base. But none of these studies analyze the fine-grain effects of neighboring bases during each step of evolution.

Morton et al. [16] have analyzed 1776 aligned SNP sequences generated from the nuclear genes of maize to study the effect of neighborhood compositions on mutation dynamics. Their studies have found that the A + T content of flanking nucleotides has an influence of various aspects of mutation dynamics. The sequences used in their study were pre-generated, while the sequences in our study are dynamically generated by changes in states of cellular automata. Siepel and Haussler [23] incorporate context-dependence in phylogenetic models to improve the quality of phylogenetic trees. Thus, the motivation of their work is similar to ours. Their work focuses on using their improved context-dependent phylogenetic models to estimate the pattern and rates of substitutions on the branches of a given phylogenetic tree. Their work also reports better estimates of branch lengths. In addition, their work has the potential to refine phylogenetic tree construction. They have estimated substitution rates and context effects for 160,000 non-coding sites and 3 million sites in coding regions in mammalian genomes. Our work tries to determine finer-grained context-dependent effects on individual steps of mutations.

DNA evolution has been modeled as cellular automata in the work by Shirakoulis et al. [24]. In this work, application of cellular automata rules to a DNA strand is treated as a matrix multiplication modulo 4. This strategy, however, cannot consider all possible cellular automata rules. The work by Stewart et al. [25] had prepared a global grid for studying arthropod evolution. The effort implemented a parallel version of fastDNAml [17] algorithm on a global grid using a maximum likelihood approach to construct better phylogenetic trees. While this effort deals with *constructing* a better phylogenetic tree, we try to *refine* the phylogenetic tree using grid computing. We also try to find additional information about these trees by modeling the DNA sequence evolution as cellular automata rules.

4. Methodology

In this section, we describe the techniques used for transformation of sequences from an ancestor to a progeny, and our assumptions regarding evolutionary rates to manage the number of successful transformations.

4.1. Sequence transformation on a branch

Our first step is to derive a sequence transformation methodology that, given an ancestor and a progeny sequence, starts with the ancestor sequence, applies a set of rules for neighbor-based mutations for a certain number of mutation steps to arrive at the progeny sequence. We subsequently invoke this methodology multiple times for a given ancestor–progeny pair to form different sets of rules and number of time steps corresponding to different ways of transformation from the ancestor to progeny. The next section describes our primary focus of running large number of ensemble experiments on grids by multiple invocations of the sequence transformation methodology for different ancestor–progeny pairs with the aim of deriving statistics regarding the most likely rules and number of time steps for the ancestor–progeny pairs.

We use Phylip [18] to construct phylogenetic trees. DNA sequences were downloaded from the HIV Sequence database at Los Alamos [21]. The sequences were aligned by ClustalW web interface [6]. The aligned sequences were then input to Phylip to obtain a phylogenetic tree. Each branch in the phylogenetic tree corresponds to an ancestor–progeny pair. For each branch, we apply a set of cellular-automata rules for transforming an ancestor sequence to the progeny sequence. We compare a sequence, produced during the transformation, with the progeny sequence using a similarity metric defined as the percentage of the number of base pairs in the sequence matching the corresponding base pairs in the progeny sequence. Thus, a similarity value of 1 indicates that the transformation has resulted in the progeny sequence.

A naive approach for transformation is to randomly choose a cellular-automata rule at each time step and apply the rule to the current sequence. But this approach can lead to the sequences deviating from the progeny sequence as illustrated in Fig. 3. This figure shows the similarity values of the sequences when using the naive approach for an ancestor–progeny branch corresponding to a phylogenetic tree constructed for gag sequences of HIV virus using the Phylip package. The progeny sequence in this branch has accession number X52154 and the ancestor sequence is one of the intermediate sequences generated by the Phylip package.

For successful transformation of an ancestor to a progeny, we can make use of the fact that not all the base-pairs mutate at each time step. Thus, at a given time step, we apply a random cellular automata rule only to those base-pairs in the current sequence which differ with the corresponding base-pairs in the progeny sequence. This approach can also be biologically justified since most of the *successful sequences* (which form complete proteins) are less prone to mutations than others. This *selective application of cellular automata rules* helps in the convergence of sequences to a progeny sequence as shown in Fig. 4. This figure shows the similarity value for each time step when using selective application of cellular automaton rules for the same branch. As seen in the figure, this approach completes the transformation in 141 time steps. During transformation, applications of certain cellular automata rules improve the similarity of the current sequence to the progeny sequence while applications of other cellular automata rules cause no change in the similarity. The step structure in the figure illustrates the switches or changes between these two kinds of rules.

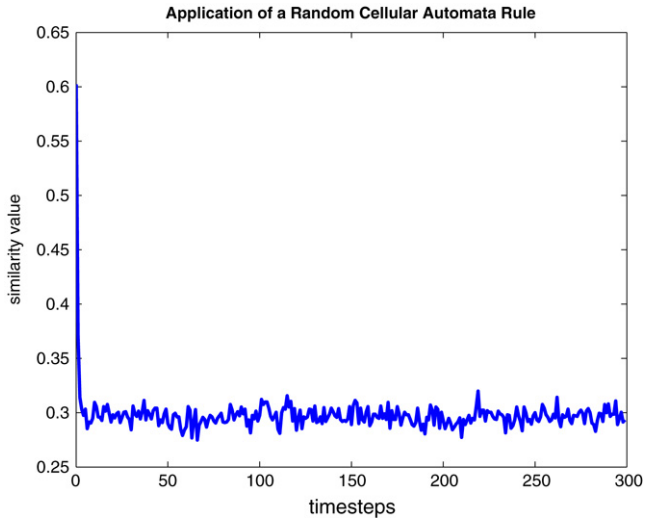


Fig. 3. Application of random cellular automata rules.

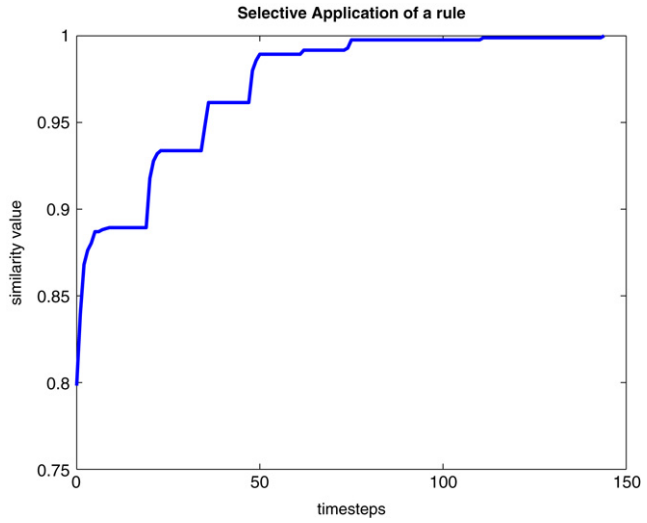


Fig. 4. Selective application of cellular automata rules.

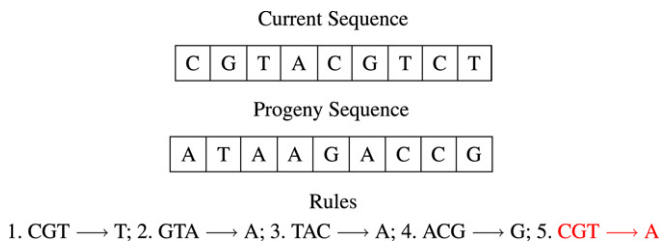


Fig. 5. Example: Dynamic formation of cellular automata rules.

In *dynamic formation of cellular automata rules*, we try to dynamically create a cellular automata rule using a sequence obtained during the transformation and the progeny sequence. Fig. 5 illustrates the dynamic formation of a rule. The current sequence on which a rule is to be applied forms the left hand sides of the transitions of the rule. The progeny sequence forms the right hand sides of the transitions. For example, in Fig. 5, the left hand side of the first transition is formed by the first three base-pairs of the current sequence, namely, CGT; and the right hand side of the transition is formed by the corresponding base-pair in the progeny sequence, T. We begin the formation of the rule from the first few base-pairs of the current and the progeny sequences. These base-pairs form a window in the current sequence and

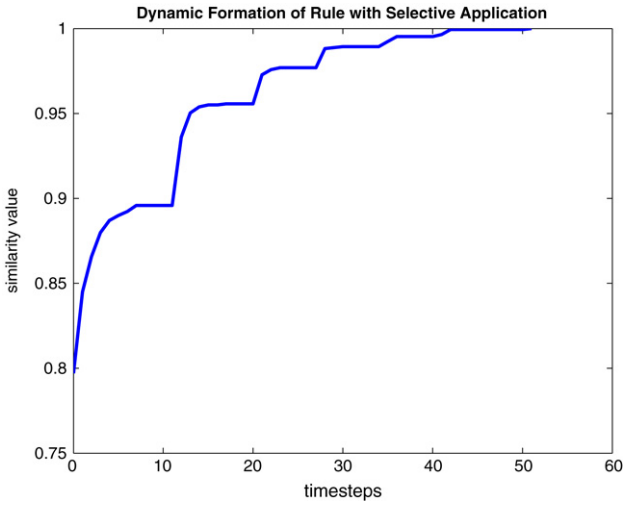


Fig. 6. Dynamic formation and selective application of cellular automata rules.

Branch 1	11	13	**10**	14	11	15		
Branch 2	20	25	**16**	21	17	18	23	20
Branch 3	25	**20**	28	21	29			
Branch 4	18	19	***17***	18	18			
Branch 5	42	**40**	41	45				
Branch 6	36	35	***35***	36	38	37		
Branch 7	**41**	48	51	49	55			

$Chains[0] = 10, 16, 20, 40, 41 (Longest\_chain); Chains[1] = 17, 35$

Fig. 7. Illustration of the greedy algorithm.

map to a single base-pair in the child sequence. The size of this window is $2 \cdot n + 1$ where n is the neighborhood size used in the cellular automata rule. In our example, the neighborhood size is one, hence the window consists of three base-pairs including a neighbor on either side of the base-pair under consideration for mutation. We slide this window over the entire parent sequence or until we find a contradicting transition during the formation of the rule. A contradicting transition is found when two windows in the current sequence containing exactly the same sub-sequence map to different base-pairs in the corresponding progeny sequence. In our example, transitions 1 and 5 contradict with each other as they try to map the same sequence, CGT, to different base-pairs. If a contradicting transition is found, we selectively apply a random cellular automata rule to the current sequence leading to a new sequence and repeat the procedure of dynamic rule formation to the new sequence. With dynamic rule formation and selective application of cellular automata rules, the number of time steps required for the transformation between the same ancestor–progeny pair is reduced considerably as shown in Fig. 6. In Fig. 6, a dynamic rule was formed at the 52nd time step leading to convergence to the progeny sequence. The difference in similarity values shown in Figs. 4 and 6 illustrate the added benefits due to dynamic rule formation.

Based on the above principles, we have written a program, *sequence transformer*, that uses selective application and dynamic formation of cellular automata rules for transformation on an ancestor–progeny branch. The sequence transformer is a fundamental component in our infrastructure. It takes as input, sequences for an ancestor and a progeny and produces as output the number of time steps required for the transformation from the ancestor to the progeny and the cellular automata rules applied during the transformation. The output also contains information regarding the exact time steps at which a cellular automata rule was applied. Initially, after alignment of the sequences, the base pairs corresponding to some segments of the ancestor sequences are not known. The sequence transformer also fills these unknown segments in the ancestor sequence with random base pairs. This assignment of unknown segments is also recorded as output of the sequence transformer.

4.2. Pseudo molecular clock assumption

Molecular clock is an assumption of constant rate of evolution [13], i.e. the rate of evolutionary change of any specified protein is approximately constant over time and over different lineages in the phylogenetic tree. Thus, according to strict molecular clock, there exists a single rate of mutation α such that

$$b_1 = \alpha \cdot t_1; b_2 = \alpha \cdot t_2; \ldots; b_n = \alpha \cdot t_n \quad (1)$$

where $b_1, b_2, b_3, \ldots, b_n$ are branch-lengths of the n branches in the phylogenetic tree and $t_1, t_2, t_3, \ldots, t_n$ are the time steps taken for mutations of the corresponding branches. Branch length is a measure of the difference between the ancestor and progeny of a branch and is obtained along with the phylogenetic tree from the Phylip package. The time steps are outputs from our sequence transformer program. The strict molecular clock assumption has been used in some phylogenetic inferences [13].

In our work, we use a pseudo-molecular clock assumption. According to this, $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n$ are related such that

$$b_1 = \alpha_1 \cdot t_1 < b_2 = \alpha_2 \cdot t_2 < \cdots < b_n = \alpha_n \cdot t_n \quad (2)$$

and

$$t_1 < t_2 < t_3 < \cdots < t_n. \quad (3)$$

This assumption is reasonable since the greater the branch lengths or greater the difference between the ancestor and progeny sequences, the more the time steps required to transform from an ancestor to a progeny.

After many invocations of the sequence transformer for different branches, we accept only those outputs of the sequence transformer that adhere to the pseudo-molecular clock assumption. To find such valid transformations, we use a greedy algorithm. The greedy algorithm starts with branches in the phylogenetic tree sorted by their branch lengths. For each branch, a linked list is maintained. Every node in a linked list corresponds to one invocation of the sequence transformer and contains the inputs and outputs for the invocation including the number of time steps taken for mutations on the branch. The greedy algorithm shown in Algorithm 1 finds a node, *prev_node*, in the first linked list having the smallest number of time steps. The *prev_node* is inserted in a *chain*. The algorithm then considers the next linked list and finds a node with the smallest time step value greater than the time step value of *prev_node*. This node now becomes the *prev_node* and is added to the *chain*. The entire procedure is then repeated for all linked lists corresponding to all the branches. During this algorithm, a branch whose linked list does not have a node with a time step value greater than that of *prev_node* may be found. Such branches are not included in the *chain*. We then try to form another *chain* which may contain the remaining branches. We repeat this procedure so that a node of every branch is included in some *chain*. At the end of this algorithm, we obtain different *chains*, each having different lengths. A chain containing all the branches in the phylogenetic tree is called *complete chain*.

The working of the greedy algorithm is illustrated in Fig. 7. The figure shows linked lists for seven branches. The numbers in the nodes of the linked lists represent the time steps. The figure shows the two chains that are produced from the greedy algorithm on the example. The numbers in **bold** font represent one chain and the numbers in ***bold-italic*** font represent another.

Algorithm 1 Greedy Algorithm for Chain Formation

Input: Array of linked lists L[] corresponding to branches. N, the number of branches
Output: *Longestchain*
{*Chains*[]: Array of linked lists with each element holding a chain }
{*inChain*[]: array of N elements where *inChain*[i] indicates if branch i belongs to a chain}

```
1:  chainCount ⇐ 0; for i ⇐ 0 to N − 1 do inchain[i] ⇐ 0 end for
2:  for i ⇐ 0 to N − 1 do
3:    if inchain[i] == 1 then
4:      continue
5:    end if
6:    prev_node ⇐ node in L[i] such that node.length is smallest
7:    Insert prev_node in Chains[chainCount]; inchain[i] ⇐ 1;
      tmp_node = NULL
8:    for j ⇐ i + 1 to N − 1 do
9:      tmp_node ⇐ node in L[j] such that node.length >
        prev_node.length and node.length is smallest
10:     if tmp_node ≠ NULL then
11:       prev_node ⇐ tmp_node; Insert prev_node in
          Chains[chainCount]; inchain[j] ⇐ 1
12:     end if
13:   end for
14:   chainCount + +
15: end for
16: Longest chain ⇐ Chains[i] such that Chains[i].length is highest in the
    array of linked lists Chains[]
```

5. Sequence transformation on grids

Although a single invocation of a sequence transformer for a single ancestor–progeny pair takes small number of time steps and hence little time to execute, the number of invocations of sequence transformer for the ancestor–progeny pair to obtain statistics regrading transformations is very large. This is due to the exponential number of possible rules that can be applied at a single step and the different number of time steps that can be needed for transformation from an ancestor to a progeny. This makes our application, of multiple invocations of sequence transformer for different ancestor–progeny pairs to derive statistics regarding the most like rules and number of time steps for the ancestor–progeny pairs, a long-running application suitable for grid computing. Our application can be ever-running since the more the number of invocations of sequence transformer, the higher the quality of the statistics that can be derived. In order to explore the vast space of parameters and to converge on the most likely values for the parameters, we use the distributed resources of a computational grid. This method is similar to the ensemble method used for climate prediction in ClimatePrediction.Net [5].

5.1. Master-worker paradigm

A master-worker paradigm is used for invocations of the sequence transformer, formation of complete chains, and insertions of parameters corresponding to the complete chains into a database. The overall design of our master-worker infrastructure is illustrated in Fig. 8.

The master is responsible for assigning branches to the workers and collecting results from them when they complete their calculations. The master assigns branches to the workers in round-robin fashion. The master, after assigning a branch to a worker, does not wait for the worker to complete its calculations, but proceeds to the next worker. When the first worker completes its calculations, the master is notified of the completion. Thus there is parallelism in the calculations by the workers. The master stores the results from the workers into linked lists corresponding to the branches assigned to the workers. Additionally, the master

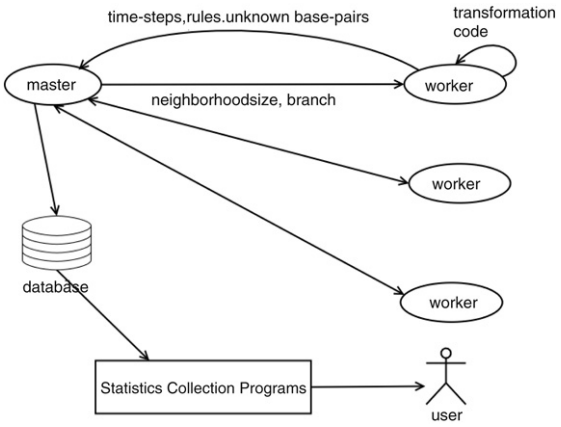


Fig. 8. The Master-worker design.

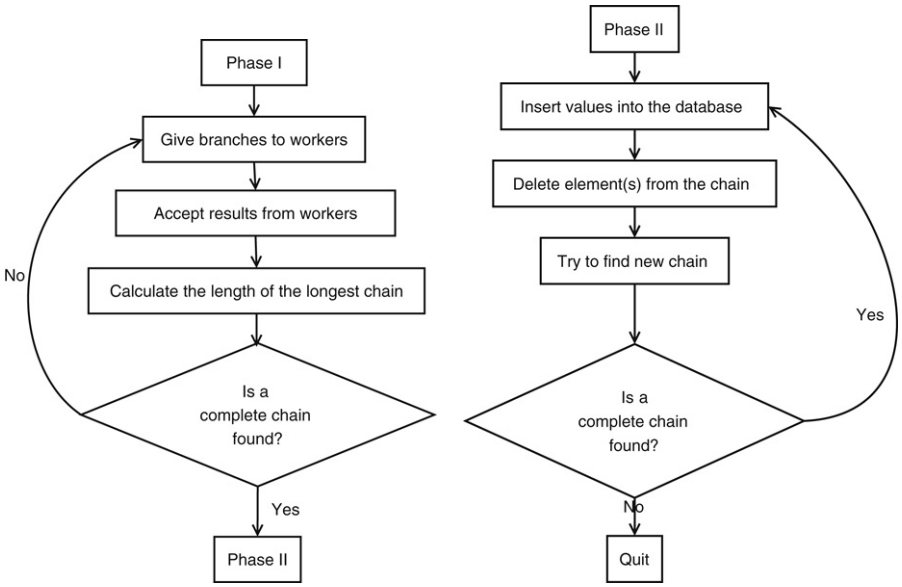
periodically invokes the greedy algorithm for chain formation with the available snapshot of linked list values for various branches. If a complete chain can be formed from the current set of linked list values, the master forks another process to insert the parameter values corresponding to the complete chain into a PostgreSQL [19] database and to possibly form additional chains from the same set of values. Note that the greedy algorithm gives us only one chain from the available set of parameter values, while there may be many complete chains in a given snapshot of linked list values. To obtain additional chains, the forked process deletes an element from the original complete chain and invokes the greedy algorithm to find another complete chain. This procedure is repeated by deleting different sets of elements from the original complete chain. During this process, whenever a complete chain is formed, it is inserted into the database.

A worker takes a branch from the master, fills the unknown base-pairs randomly and invokes the sequence transformer. When the transformation is complete, it sends the results back to the master. The results consist of the number of time steps required for transformations, the rules used during the transformations and the assignment of base-pairs to the unknown portions of the ancestor sequence. The worker then waits for a new branch from the master. The number of worker processes are not related to the number of branches. Hence our framework can make use of any number of available grid resources for the execution of worker processes.

5.2. Phases of execution

The master process operates in one of two phases. In *phase* I, shown in Fig. 9(a), the master continuously gives new branches to the worker processes and collects results from them. The master initially considers all branches for allocations to workers. Once the number of branches in the longest chain exceeds 60% of the total number of branches, the master considers only those branches that are not in the longest chain for allocations to workers. Thus, more resources are used for *difficult* branches for complete chain formation. To avoid potentially large differences between the number of invocations of the sequence transformer for any two branches at this stage, we fix a threshold, *allocation_threshold*, for maximum difference between the maximum and minimum invocations for any two branches in the phylogenetic tree. If this difference exceeds the threshold, the branch with the minimum number of invocations is assigned to the workers until the difference becomes less than the threshold. During phase I, the master also invokes the chain formation algorithm periodically.

Once a complete chain is formed, the master initiates *phase* II. Phase II, shown in Fig. 9(b), involves the insertion of complete chains into the database and the formation of additional complete



(a) Phase I in master. (b) Phase II in master.

Fig. 9. Phases in master.

chains from the same data. Note that for phase II to start, phase I must complete. But, once phase II has been started, the next round of phase I can be started immediately, ensuring pipelined parallelism between phase I and phase II in the master.

5.3. Grid computing techniques

A simple Unix shell script was used to start the master process on the master machine and execute the worker processes on a given set of remote client machines through '*ssh*' commands. Another shell script was run periodically for every 10 min to monitor the status of the worker processes and restart the failed worker processes on the client machines. The shell scripts use Unix *password*-based security mechanisms to establish connections to the remote client machines. For communications between the master and the worker processes, we used Unix TCP/IP socket functions with external data representation (XDR) standard for encoding and decoding data for communication between heterogeneous resources. These communications include transfer of the branch IDs from the master to the workers and the results, namely, the number of time steps and cellular automata rules from the workers to the master. The DNA sequences corresponding to the branches of a phylogenetic tree were pre-staged to the client machines using Unix FTP mechanisms. The pre-staging helped in avoiding the communications of the sequences between the master and the worker processes every time a branch is allocated to a worker. Though we use simple mechanisms for remote process execution and management, monitoring, data transfers and security, it is easy to upgrade the infrastructure to use the grid standard mechanisms of Globus toolkit [8], namely, Grid Resource Allocation and Management (GRAM) [7] for execution management, WebMDS for monitoring and discovery [31], GridFTP [1] for data access and movement, and GSI for security [27].

The grid resources can be highly dynamic in terms of load and availability. Due to the randomness involved in the sequence transformer, the computation times for sequence transformer executions on different branches can differ and can cannot be determined *a priori*. Thus, scheduling strategies that map tasks with larger computations to faster client machines cannot be employed. Hence round-robin scheduling of tasks to the resources was employed at the master. However, the round-robin scheduling can lead to load imbalance among branches due to different rates of progress on different branches. We used load balancing techniques to adapt to the resource and application dynamics in grids. We also employed fault-tolerant techniques for sustaining computations in the presence of resource failures.

5.3.1. Load balancing

During phase I, when the number of branches in the longest chain is less than 60% of the total number of branches, all branches are allocated to the workers for calculations. During this stage, there is a possibility that a particular branch is always assigned to a slow worker hampering the progress for the branch. A threshold, *loadbalance_threshold*, is fixed for maximum allowed difference between the number of invocations of sequence transformer for any two branches. If the difference exceeds the threshold, the branch with the minimum number of invocations is allocated to two workers during each iteration until the difference drops below the threshold. This technique allows uniform progress for all branches irrespective of the different loads on the grid resources where the workers are executing. Theoretically, any number of phase I and phase II processes can be executing at any given point. However, both phase I and phase II processes consume memory on the machine where the master is executing. Hence, phase I and phase II processes must be started after verifying whether the required amount of memory is available at the master machine. We follow a simplistic approach where only one phase I and one phase II processes may be active at any given point of time. This controls the load on the master machine allowing efficient execution of the master.

5.3.2. Fault tolerance

For each worker, the master forks a process that sends the parameters to and receives the results from the worker. The forked process maintains a connection with the worker throughout the calculation of the branch. After the results are sent back by the worker, the forked child process notifies the parent of the arrival of results. Hence, even if the worker fails during its execution, only the forked process gets killed and the master will be able

to continue its execution. Also, the master forks a new process for phase II operations. The forked phase II process does not need to communicate with the main master process. This achieves not only parallelism but also fault tolerance because even if the phase II process fails, it doesn't affect the execution of the main master process. The master process will eventually fork off another phase II process. Finally, even if the master fails after some time, the results obtained so far are still accessible from the database. The user can still collect the statistics for the results from the database irrespective of whether the master is alive. Due to the durability property of databases, we can continue to build upon the previous successful transformations in the case of the master failure.

6. Statistics collection

We have developed various statistics collection programs that retrieve values from the database and give various kinds of statistics. There can be different complete chains with different parameters satisfying the pseudo molecular clock assumption. The statistics collection programs extract collective statistics from all the complete chains. These statistics collection programs can be executed offline by interested users at any point of time. The various statistics that are of interest are:

(1) Timesteps: number of time steps required for transformation. This information can be used to obtain more accurate measures about rates of evolution. We obtain the average and standard deviation values of the time steps for a transformation for a particular branch across different complete chains.
(2) Unknown base-pairs: probabilities of a base-pair assignment to the unknown segments of the ancestor DNA sequences. These probabilities may help in re-building the complete intermediate sequences of the phylogenetic tree.
(3) Rules: various rules used during transformation. These rules may give insights on the impact of neighboring base-pairs on the evolution of DNA sequences. We collect *popular* rules within a particular branch across all the complete chains and for a single complete chain across all branches.
(4) Differential rule analysis: probability of a particular rule being used at a given time step of transformation in a given branch. This analysis may give finer insights into the individual time steps of mutation.
(5) Popularity of transitions: the number of times a particular transition is used for a given branch. This may be useful in determining the exact effects of neighboring base-pairs on mutations.

7. Experiments and results

We conducted experiments and obtained interesting statistics for three HIV genomic segments corresponding to *gag*, *gagpol* and *env* sequences. For brevity, we present results corresponding to only gag sequences. Similar results were obtained for gagpol and env sequences.The results were obtained by executing the statistics collection programs on the PostgreSQL databases formed for the sequences. Since the grid infrastructure that we developed could execute the transformations for long periods of time and produce better statistics, the results presented in this section should be treated as representing good samples and demonstrating the potential of the infrastructure.

The sequences were downloaded from the HIV Sequence database at Los Alamos [21] and were aligned by ClustalW web interface [6]. The lengths of the aligned sequences were 1690, 4494 and 2817 for gag, gagpol and env sequences respectively. These numbers also denote the number of base-pair transitions in a single transformation step for the corresponding sequence. Thus, successful transformations for these sequences with our cellular automata rules, that define only 64 transitions, indicate that a single neighbor-dependent transition defined by a rule in an evolution step can occur at more than one position of the sequence. For each of the three aligned sequence types, a phylogenetic tree was obtained using the Phylip [18] package. The number of branches of the phylogenetic trees were 84, 47 and 46 for gag, gagpol and env, respectively. We utilized a grid infrastructure consisting of 23 machines distributed in 3 countries, as shown in Table 2, for our experiments. These machines operated in non-dedicated modes and were used by local users for different purposes. The worker processes were executed on all the machines. The master process and the PostgreSQL database, where information pertaining to the complete chains of a sequence type are stored by the master, were started on one of the AMD machines in Indian Institute of Science. The results that were obtained correspond to obtaining statistics with 7332, 4759 and 3251 complete chains collected in the PostgreSQL databases for gag, gagpol and env sequences, respectively, after 13, 4 and 3 days, respectively, from the start of the corresponding experiments on the distributed infrastructure. Each complete chain contains all the branches of the phylogenetic tree for a particular sequence type. Each branch of a chain is associated with a set of rules and a certain number of time steps for transformations from the ancestor to progeny sequence of the branch. Different chains may have different sets of rules and different number of time steps for the branch.

Our first interest is to establish if our techniques, based on cellular automata and involving ensemble experiments on grids, can verify the phenomenon of neighbor-dependent mutations in some steps of evolution. Table 3 shows the probabilities

Table 2
The distributed infrastructure

Cluster and location	Number of m/cs	Specifications
Torc cluster, University of Tennessee (UT), USA	8	GNU/Linux 2.6.8, Dual PIII 933 MHz, 512 MB RAM, 40 GB Hard Drive, 100 Mbps Ethernet
DAS-2, Vrije Universiteit, Netherlands	9	GNU/Linux 2.4.21, Dual PIII 996 MHz, 1GB RAM, 20 GB Hard Drive, 100 Mbps Fast Ethernet.
AMD cluster, Indian Institute of Science, India	6	AMD Opteron 246 based 2.21 GHz servers, Fedora Core 4.0, 1 GB RAM, 160 GB Hard Drive, Gigabit Ethernet

Table 3
Differential rule analysis for gag sequences

Branch number (Ancestor–progeny)	Rule used	Time-steps	Prob.
11 (15–16)	CAGGCAAACGCCTGTT ACATTAATTTCGTGCC GTTAAAAGTGTGCGGC TGATGCGCAAATGCCC	42–43	0.95
12 (1–45)	GGGTCAATTTTGGCAT GACCATATTGTCCTCA GATACTAAGTTCATAG AGAAGAACGATTACTT	43–44	0.95
29 (26–27)	CCGAAGTGATTGAAGC GCTTGTTTCTGGCGAT TTTTGTGGTCCACTCA CCTTATTCGCAAAATA	59–61	0.98
32 (16–02_AG)	TCGCAGCGACCGCATA AACATACCGGCTGGCG ATAAGCTGGACTCAAC ACGAGTGCCAAATCTT	65–66	0.92
83 (6–21)	ATATGTGCGGTTACTA TCGGTCTCCGGAGGTG CACTTACCGCCGGATG GCACTAGAGAACTATA	198–199	0.94

The 64 characters in column 2 of the table represent the 64 right-hand sides of the transitions shown in Table 1.

Table 4
Summary of time step information for Gag sequences

Branch number (Ancestor–progeny)	Average number of time steps	Standard deviation
0 (8–9)	15.657665	1.961112
4 (19–14)	31.242908	2.085160
5 (43–44)	32.274139	2.096791
6 (20–21)	34.051281	1.852813
7 (13–24)	35.154255	1.833268
8 (24–36)	36.161758	1.827739
9 (5–4)	37.184532	1.814149
10 (29–30)	38.198307	1.816491
11 (15–16)	39.253136	1.898609
12 (1–45)	40.259411	1.904292
13 (12–13)	41.311649	1.974289
14 (14–18)	42.380253	1.991383
15 (25–26)	43.405754	1.986514
16 (22–8)	44.409302	1.990247
17 (22–15)	45.484451	2.040152

of applications of some cellular automata rules for neighbor-dependent mutations at certain discrete time steps of mutations on some branches of the phylogenetic tree formed from gag sequences. The probability for a rule was obtained by dividing the number of times the rule was applied and the total number of applications of rules for the time step(s). The high probability numbers for about 7000 samples indicate that the corresponding rules for neighbor-dependent mutations are most likely to have been followed in the corresponding steps of evolution. For example, the third entry shows shows that there is 98% probability for the corresponding rule to have been applied during steps 59–61 of mutations in the corresponding branch. If mutations of DNA segments are independent of their neighbors, all the applied rules for neighbor-dependent mutations would have been applied with approximately equal low probabilities. Thus these results show that our cellular automata rules can capture the highly probable neighbor-dependent mutations at some time steps of mutations. Unlike the previous efforts that deal with specific neighbor-dependent transitions [3,16], our work studies the combined effects of a set of neighbor-dependent transitions that define a rule.

Our work also tries to determine the number of time steps taken for mutations from the ancestor to progeny sequences of branches of a phylogenetic tree. The resolution of time steps is useful for analyzing the mutations of existing sequences and can also help in expanding the phylogenetic tree to predict future sequences. Table 4 shows the averages and standard deviations of the number of time steps taken for mutations for some branches of the phylogenetic tree for gag sequences. The averages and standard deviations were obtained over the 7332 complete chains formed for the gag sequences. The low standard deviation values for large number of branches indicate that we are able to determine the most likely number of time steps needed for mutations from the ancestor to progeny sequences of the branches irrespective of the rules applied for the mutations. For example, for branch 10, the most likely number of time steps needed for mutations from the ancestor to the progeny of the branch are in the range 36–40. These fine-level details regarding the number of time steps are not associated with the phylogenetic trees generated by existing packages.

Table 5 shows the resolutions of unknown positions of the intermediate gag sequences. The probability of occurrence of a base-pair in an unknown position of an intermediate sequence, shown in the table, was calculated by dividing the number of resolutions of the position with the base-pair by the total number of resolutions of the position. Entries 1, 3 and 6 of the table show that the corresponding positions of the sequences are most likely to be occupied by purines (A and G) than pyrimidines (C and T). The other entries show that the corresponding positions

Table 5
Resolution of unknown positions for gag sequences

S. no.	Seq. no. (Name)	Position	prob (A) (1)	prob (C) (2)	prob (G) (3)	prob (T) (4)	prob (purines) (1+3)	prob (pyra-midines) (2+4)
1	26 (3)	1390	0.34	0.14	0.40	0.10	0.74	0.25
2	26 (3)	375	0.17	0.24	0.08	0.50	0.25	0.74
3	4 (1)	410	0.43	0.11	0.30	0.15	0.73	0.26
4	9 (14)	411	0.12	0.34	0.13	0.39	0.26	0.73
5	10 (15)	386	0.12	0.41	0.14	0.31	0.26	0.73
6	17 (21)	1251	0.26	0.07	0.46	0.19	0.73	0.26

Columns, 4, 5, 6, and 7 show the probabilities of occurrence of the corresponding base-pairs in an unknown position of an intermediate sequence.

Table 6
Usefulness of large number of runs

Branch	Average number of time steps	Standard deviation
Part 1. On day X. Number of complete chains = 1347		
0	14.625093	3.565527
1	20.830734	3.153974
Part 2. On Day (X + 10). Number of complete chains = 7607		
0	15.832522	1.890947
1	22.199816	1.708907

of the sequences are most likely to be occupied by pyrimidines than purines. Resolution of the intermediate sequences leads to completeness and improvement in quality of the associated phylogenetic tree. This in turn can enhance the studies that utilize these phylogenetic trees for better predictions of future sequences.

In order to show that our long-running computations can have potential long term benefits in resolving uncertainties associated with mutations, we conducted experiments with gag sequences on the 6 AMD machines in India. We then observed the average number of time steps in mutations of 2 branches of the phylogenetic tree at 2 different periods of time separated by 10 days. Table 6 shows the results corresponding to the 2 branches. The first part of the table shows the results obtained on a particular day when 1347 complete chains were collected in the PostgreSQL database and the second part shows the results obtained 10 days later when 7607 complete chains were collected. The lower standard deviation values for the results in the second part of the table show that the average number of time steps converges with increasing number of executions. Thus, our work can give more definite findings regarding mutations as more time is spent by the application on grid resources. Thus, our application regarding DNA sequence transformations is a natural grid application that can run continuously and make use of more grid resources, when available, to yield continuous better results.

8. Discussion

The primary focus of our current work has been exploring different ways of convergence to progeny sequence from an ancestor sequence using cellular automata rules. Our techniques, namely, selective application of rules and dynamic formation of rules, follow greedy approaches to attain convergence. In order to model evolutions more realistically, we plan to follow only selective application of rules and probabilistically freeze a position in the current sequence once the base-pair at the position matches with the corresponding base-pair in the progeny sequence.

Currently, we assume that a single rule for neighbor-dependent mutations is applied for the entire sequence at a single time step. In future, we plan to split a sequence into windows of different sizes and apply different rules for different windows in a time step. This will help in faster convergence especially for larger sequences

where same neighborhoods can lead to different transitions of a base-pair in different regions of the sequence. We also plan to focus more on popularity of individual transitions that define a rule. While mutations in a time step will be based on cellular automata rules, the analysis of mutations will be on transitions. This will not only help in considerably reducing the storage complexity associated with the rules, but can also lead to more definite statistics regarding neighbor-dependent mutations.

In this work, we had considered only the effects of a left and a right neighbor on transitions of a base-pair. We plan to explore larger neighborhood sizes. However, for large neighborhood sizes and small sequences like HIV, the probability of finding a base-pair at different positions of a sequence with same flanking neighbors is low. Hence, cellular automata rules, that imply same transitions for a base-pair when flanked with same neighbors, will not be applicable for larger neighborhood sizes. Thus, mutation analysis should be performed at the level of transitions. Considering very large neighborhood sizes also necessitates management and analysis of many hundred thousand transitions resulting in significant increase in storage complexity. We plan to restrict our work to analyze the effects of maximum of three flanking neighbors on either side of a base-pair.

We also plan to consider larger sequences similar to those found in human genomes [12]. Though the number of time steps for convergence does not increase with the increase in length of the sequence, the time taken for processing a single time step will increase. We plan to employ finer-level parallelization scheme where the sequence transformer will be parallelized by splitting the sequence into multiple regions and dividing the regions across processors. We plan to employ a 2-level parallelization by dividing the grid resources into several clusters. While different sequence transformer instances will be executed simultaneously at different clusters, parallelization of a sequence transformer instance will be attained within a cluster of machines.

9. Conclusions

In this work, we have developed techniques based on cellular automata to analyze the rules for neighborhood-based mutations on branches of a phylogenetic tree. Fine-grained analysis of rules at different time steps of mutations, and resolving the number of time steps of mutations and incomplete intermediate sequences associated with phylogenetic trees are the primary contributions of our work to evolution studies. The key elements of the contributions are a novel approach to studying mutations by exploring different rules for transformations from an ancestor to a progeny sequence, building a master-worker application for simultaneous exploration of different number of rules, development of the application as an ever-running application suitable for grid computing, and techniques for coordination, maintenance and statistical analysis of large-scale ensemble explorations on grids. Based on the results collected for three HIV sequences, we have verified the phenomenon of neighbor-dependent mutations, found that certain combinations of these mutations occur with greater than 90% probabilities, determined the average number of time steps of mutations over large number of explorations with less than 2 standard deviations, and also resolved unknown positions of incomplete intermediate sequences associated with phylogenetic trees. We were also able to show that grids are very suitable for our sequence transformation application since the ever-expanding grid resources can be used for increasing the exploration space and help in convergence of findings regarding mutations.

10. Future work

Apart from dealing with various research challenges covered in Section 8, we also plan to develop robust scheduling mechanisms to map the individual sequence transformations to the processors of a grid. Existing scheduling techniques that need expected time to completion will not be adequate because the number of time steps needed for sequence transformations cannot be determined *a priori*. Considering very large neighborhood sizes and studying varying impacts of different neighborhoods are other challenging research problems that may be considered in future.

References

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, The globus striped gridFTP framework and server, in: Proceedings of Super Computing 2005, SC05, 2005.
[2] P. Arndt, C. Burge, T. Hwa, DNA sequence evolution with neighbor-dependent mutation, Journal of Computational Biology 10 (2003) 313–322.
[3] M. Bulmer, Neighboring base effects on substitution rates in pseudogenes, Molecular Biology and Evolution 3 (4) (1986) 322–329.
[4] C. Burks, D. Farmer, Towards modeling dna sequences as automata, Physica D: Nonlinear Phenomena 10 (1–2) (1984) 157–167.
[5] Climateprediction.net, http://www.climateprediction.net.
[6] ClustalW, http://www.ebi.ac.uk/clustalw.
[7] K. Czajkowski, I. Foster, C. Kesselman, Agreement-based resource management, Proceedings of the IEEE 93 (3) (2005) 631–643.
[8] I. Foster, Globus toolkit version 4: Software for service-oriented systems, in: IFIP International Conference on Network and Parallel Computing, in: LNCS, vol. 3779, Springer-Verlag, 2006.
[9] N. Ganguly, B. Sikdar, A. Deutsch, G. Canright, P. Chaudhuri, A survey on cellular automata, Tech. rep., Centre for high performance computing, Dresden University of Technology (December 2003).
[10] Z. Hao, S. Yan, R. Gunaretnam, A. Mondry, P. Dhar, Facilitating arrhythmia simulation: The method of quantitative cellular automata modeling and parallel running, Biomedical Engineering Online 3 (29).
[11] S. Hess, D. Jonathan, R. Blake, Wide variations in neighbor-dependent substitution rates, Journal of Molecular Biology 236 (1994) 1022–1033.
[12] Human genome project information, http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml.
[13] B. Korber, M. Muldoon, J. Theiler, F. Gao, R. Gupta, A. Lapedes, B.H. Hahn, S. Wolinsky, T. Bhattacharya, Timing the ancestor of the HIV-1 pandemic strains, Science 288 (5472) (2000) 1789–1796.
[14] K. Laurio, F. Linakera, A. Narayanana, Regular biosequence pattern matching with cellular automata, Information Sciences 146 (1–4) (2002) 89–101.
[15] Mcell, http://www.mcell.cnl.salk.edu.
[16] B. Morton, I. Bi, M. McMullen, B. Gaut, Variation in mutation dynamics across the maize genome as a function of regional and flanking base composition, Genetics 172 (1) (2006) 569–577.
[17] G. Olsen, H. Matsuda, R. Hagstrom, R. Overbeek, FastDNAmL: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood, Computer Applications in the Biosciences 10 (1) (1994) 41–48.
[18] Phylip Package, http://evolution.genetics.washington.edu/phylip.html.
[19] PostgreSQL, http://www.postgresql.org.
[20] H.-P. Schwefel, Deep insight from simple models of evolution, BioSystems 64 (1) (2002) 189–198.
[21] HIV Sequence Database, http://hiv.lanl.gov.
[22] SETI@Home, http://setiathome.berkeley.edu.
[23] A. Siepel, D. Haussler, Phylogenetic estimation of context-dependent substitution rates by maximum likelihood, Molecular Biology and Evolution 21 (3) (2004) 468–488.
[24] G. Sirakoulis, I. Karafyllidis, C. Mizas, V. Mardiris, A. Thanailakis, P. Tsalides, A cellular automaton model for the study of DNA sequence evolution, Computers in Biology and Medicine 33 (5) (2003) 439–453.
[25] C. Stewart, D. Hart, M. Aumuller, R. Keller, M. Muller, H. Li, R. Repasky, R. Sheppard, D. Berry, M. Hess, U. Wossner, J. Colbourne, A global grid for analysis of arthropod evolution, in: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, 2004.
[26] Understanding Evolution, http://evolution.berkeley.edu.
[27] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke, Security for grid services, in: HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, 2003.
[28] D. Wishart, R. Yang, D. Arndt, P. Tang, J. Cruz, Dynamic cellular automata: An alternative approach to cellular simulation, Silico Biol. 5 (2) (2005) 139–161.
[29] S. Wolfram, A New Kind of Science, Wolfram Media, Inc, 2002.
[30] P. Woolf, J. Linderman, Self organization of membrane proteins via dimerization, Biophysical Chemistry 104 (1) (2003) 217–227.

[31] X. Zhang, J. Freschl, J. Schopf, A performance study of monitoring and information services for distributed systems, in: HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, 2003.

Yadnyesh Joshi is a Software Design Engineer at Microsoft. He obtained his B.E. degree in the Department of Computer Engineering at K. J. Somaiya College of Engineering, India in 2005 and received his Masters degree at Supercomputer Education and Research Centre, Indian Institute of Science in 2007.

Sathish Vadhiyar is an Assistant Professor in Supercomputer Education and Research Centre, Indian Institute of Science. He obtained his B.E. degree in the Department of Computer Science and Engineering at Thiagarajar College of Engineering, India in 1997 and received his Masters degree in Computer Science at Clemson University, USA in 1999. He graduated with a Ph. D. in the Computer Science Department at University of Tennessee, USA in 2003. His research areas are in parallel and grid computing with primary focus on performance modeling of parallel applications, scheduling and rescheduling methodologies for grid systems, and grid applications. Dr. Vadhiyar is a member of IEEE and has published papers in peer-reviewed journals and conferences. He was a tutorial chair and session chair of escience 2007 and served on the program committees of conferences related to parallel and grid computing including IPDPS, CCGrid, eScience and HiPC.