Developing and Applying Heterogeneous Phylogenetic Models with XRate

Oscar Westesson, Ian Holmes*

Department of Bioengineering, University of California, Berkeley, California, United States of America

Abstract

Modeling sequence evolution on phylogenetic trees is a useful technique in computational biology. Especially powerful are models which take account of the heterogeneous nature of sequence evolution according to the "grammar" of the encoded gene features. However, beyond a modest level of model complexity, manual coding of models becomes prohibitively labor-intensive. We demonstrate, via a set of case studies, the new built-in model-prototyping capabilities of XRate (macros and Scheme extensions). These features allow rapid implementation of phylogenetic models which would have previously been far more labor-intensive. XRate 's new capabilities for lineage-specific models, ancestral sequence reconstruction, and improved annotation output are also discussed. XRate 's flexible model-specification capabilities and computational efficiency make it well-suited to developing and prototyping phylogenetic grammar models. XRate is available as part of the DART software package: http://biowiki.org/DART.

Citation: Westesson O, Holmes I (2012) Developing and Applying Heterogeneous Phylogenetic Models with XRate. PLoS ONE 7(6): e36898. doi:10.1371/journal.pone.0036898

Editor: Christos A. Ouzounis, The Centre for Research and Technology, Hellas, Greece

Received October 18, 2011; **Accepted** April 9, 2012; **Published** June 5, 2012

Funding: This research was funded from an NIH Research Project Grant Program (R01). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

* E-mail: ihh@berkeley.edu

Introduction

Phylogenetics, the modeling of evolution on trees, is an extremely powerful tool in computational biology. The better we can model a system, the more can learn from it, and vice-versa. Especially attractive, given the plethora of available sequence data, is modeling sequence evolution at the molecular level. Models describing the evolution of a single nucleotide began simply (e.g. JC69 [1]), later evolving to capture such biological features as transition/transversion bias (e.g. K80 [2]) and unequal base frequencies (e.g. HKY85 [3]). Felsenstein's "pruning" algorithm allows combining these models with phylogenetic trees to compute the likelihood of multiple sequences [4].

As powerful as phylogenetic models are for explaining the evolutionary depth of a sequence alignment, they are even more powerful when combined with a model for the feature structure: the partition of the alignment into regions, each evolving under a particular model. The phylogenetic grammar, or "phylo-grammar", is one such class of models. Combining hidden Markov models (and, more generally, stochastic grammars) and phylogenetic substitution models provides computational modelers with a rich set of comparative tools to analyze multiple sequence alignments (MSAs): gene prediction, homology detection, finding structured RNA, and detecting changes in selective pressure have all been approached with this general framework [5–8]. Readers unfamiliar with phylo-grammars may benefit from relevant descriptions and links available here: http://biowiki.org/PhyloGrammars or the original paper describing XRate [9]. Also, a collection of animations depicting various evolutionary models at work (generating multiple alignments or evolving sequences) has been compiled here: http://biowiki.org/PhyloFilm.

While the mathematics of sequence modeling is straightforward, manual implementation can quickly become the limiting factor in iterative development of a computational pipeline. To streamline this step, general modeling platforms have been developed. For instance, Exonerate allows users to specify a wide variety of common substitution and gap models when aligning pairs of sequences [10]. Dynamite uses a specification file to generate code for dynamic programming routines [11]. HMMoC is a similar model compiler sufficiently general to work with arbitrary HMMs [12]. The BEAST program allows users to choose from a wide range of phylogenetic substitution models while also sampling over trees [13]. The first three of these are non-phylogenetic, only able to model related pairs of sequences. Dynamite and HMMoC are unique in that they allow definition of arbitrary models via specification files, whereas users of BEAST and Exonerate are limited to the range of models which have been hard-coded in the respective programs.

Defining models' structure manually can be limiting as models grow in size and/or complexity. For instance, a Nielsen-Yang model incorporating both selection and transition/transversion bias has nearly 4000 entries - far too many for a user to manually specify [14]. Such a large matrix requires specific model-generating code to be written and integrated with the program in use - not always possible or practical for the user depending on the program's implementation.

XRate is a phylogenetic modeling program that implements the key parameterization and inference algorithms given two ingredients: a user-specified phylo-grammar, and a multiple sequence alignment. (A phylogeny can optionally be specified by the user, or it can be inferred by the program.) XRate 's models describe the parametric structure of substitution rate matrices, along with grammatical rules governing which rate matrices can account for which alignment columns. This essentially amounts to partitioning the alignment (e.g. marking up exon boundaries and reading frames) and factoring in the transitions between the different types of region.

Parameter estimation and decoding (alignment annotation) algorithms are built in, allowing fast model prototyping and fitting. Model training (estimating the rate and probability parameters of the grammar) is done via a form of the Expectation Maximization (EM) algorithm, described in more detail in the original XRate paper [9]. Most recently, XRate allows programmatic model construction via its macros and Scheme extensions. XRate 's built-in macro language allows large, repetitive grammars to be compactly represented, and also enables the model structure to depend on aspects of the data, such as the tree or alignment. Scheme extensions take this even further, interfacing XRate to a full-featured functional scripting language, allowing complex XRate -oriented workflows to be written as Scheme programs.

In this paper we demonstrate XRate 's new model-specification tools via a set of progressively more complex examples, concluding with XDecoder, a phylo-grammar modeling RNA secondary structure overlapping protein-coding regions. We also describe additional improvements to XRate since its initial publication, namely ancestral sequence reconstruction, GFF/WIG output, and hybrid substitution models. Finally, we show how XRate's features are exposed as function extensions in a dialect of the Scheme programming language, typifying a Functional Programming (FP) style of model development and inference for phylogenetic sequence analysis. Terminology relevant to modeling with XRate are defined in detail in Text S3. We also provide an online tutorial for making nontrivial modifications to existing grammars, going step-by-step from a Jukes-Cantor model to an autocorrelated Gamma-distributed rates phylo-HMM: http://biowiki.org/XrateTutorial.

Methods

The XRate generative model

A phylo-grammar generates an alignment in two steps: nonterminal transformations and token evolution. The sequence of nonterminal transformations comprises the "grammar" portion of a phylo-grammar, and the "phylo" portion refers to the evolution of tokens along a phylogeny. First, transformation rules are repeatedly applied, beginning with the START nonterminal, until only a series of pseudoterminals remains. From each group of pseudoterminals (a group may be a single column, two "paired" columns in an RNA structure, or a codon triplet of columns), a tuple of tokens is sampled from the initial distribution of the chain corresponding to the pseudoterminal. These tokens then evolve down the phylogenetic tree according to the mutation rules of the chain, resulting in the observed alignment columns.

If the nonterminal transformations contain no bifurcations and all emissions occur on the same side of the nonterminal, the grammar is a phylogenetic hidden Markov model (phylo-HMM), a special subclass of phylo-grammars. Otherwise, it is a phylogenetic stochastic context-free grammar (phylo-SCFG), the most general class of models implemented by XRate. This distinction, along with other related technical terms, are described in greater detail in Text S3, the Glossary of XRate model terminology.

The generality of XRate requires a slight tradeoff against speed. Since the low-level code implementing core operations is shared among the set of possible models, XRate will generally be slower than programs with source code optimized for a narrower range of models. Computing the Felsenstein likelihood under the HKY85 [3] model of a 5-taxon, 1 Mb alignment, XRate required 1.25 minutes of CPU time and 116 MB RAM, while PAML required 9 seconds of CPU time and 19 MB RAM for the same operation. Running PFOLD [15] on a 5-taxon, 1 KB alignment required 11 seconds and 164 MB RAM, and running XRate on the same alignment with a comparable grammar required 25 seconds and 62 MB RAM. All programs were run with default settings on a 3.4 GHz Intel i7 processor. Model-fitting also takes longer with XRate: a previous work found that XRate's parameter estimation routines were approximately 130 times slower than those in PAML [16].

In an attempt to improve XRate's performance, we tried using Beagle, a library that provides CPU and accelerated parallel GPU implementations of Felsenstein's algorithm along with related matrix operations [17]. We have, however, been so far unable to generate significant performance gains by this method.

Despite these caveats, XRate has proved to be fast enough for genome-scale applications, such as a screen of *Drosophila* whole-genome alignments [18]. Furthermore, it implements a significantly broader range of models than the above-cited tools.

XRate inputs, outputs and operations

The formulation of the XRate model presented in the previous section is generative: that is, it describes the generation of data on a tree. In practice, the main reason for doing this is to generate simulation data for benchmarking purposes. This is possible using the tool simgram [19], which is provided with XRate as part of the DART package.

Most common use cases for generative models involve not simulation, but inference: that is, reconstructing aspects of the generative process (sequence of nonterminal transformations, token mutations, or grammar parameters) given observed sequence data (in the form of a multiple sequence alignment). Using a phylo-grammar, a set of aligned sequences, and a phylogeny relating these sequences (optionally inferred by XRate), XRate implements the relevant parameterization and inference algorithms, allowing researchers to analyze sequence data without having to implement their own models.

Sequences are read and written in Stockholm format [20] (converters to and from common formats are included with DART). This format allows for the option of embedding a tree in Newick format [21] (via the # = GF NH tag) and annotations in GFF format [22]. By construction, Newick format necessarily specifies a rooted tree, rather than an unrooted one. However, the root placement is only relevant for time-irreversible models; when using time-reversible models, the placement of the root is arbitrary and can safely be ignored. Given these input ingredients, a call to XRate proceeds in the following order (more detail is provided at http://biowiki.org/XRATE and http://biowiki.org/XrateFormat):

1. The Stockholm file and grammar alphabet are parsed (as macros may depend on these).
2. Any grammar macros are expanded, followed by Scheme functions.
3. If requested, or a tree was not provided in the input data, one is estimated using neighbor-joining [23]. As noted above, this is a rooted tree, but the root placement is arbitrary if a time-reversible model is used.
4. Grammar parameters are estimated (if requested).
5. Alignment is annotated (if requested).
6. Ancestral sequences are reconstructed (if requested).

After the analysis is complete, the alignment (along with an embedded tree) is printed to the output stream along with ancestral sequences (if requested) as well as any #=GC and #=GR column annotations. GFF and WIG annotations are sent to standard output by default, but these can be directed to separate files by way of the -gff and -wig options, respectively.

Results and Discussion

The XRate format macro language for phylo-grammar specification: case studies

The following sections describe case studies of repetitively-structured models which motivate the need for grammar-generating code. Historically, we have attempted several solutions to the case studies described. We first briefly review the factors that influenced our eventual choice of Scheme as a macro language.

XRate was preceded by Searls' Prolog-based automata [24] and Birney's Dynamite parser-generator [11], and roughly contemporaneous with Slater's Exonerate [10] and Lunter's HMMoC [12]. In early versions of XRate (circa 2004), and in Exonerate, the only way for the user to specify their own phylo-grammar models was to write C/C++ code that would compile directly against the program's internal libraries. This kind of compilation step significantly slows model prototyping, and impedes re-use of model parameters.

Current versions of XRate, along with Dynamite and HMMoC, understand a machine-readable grammar format. In the case of XRate, this format is based on Lisp S-expressions. In such formats (as the case studies illustrate) the need arises for code that generates repetitively-structured grammar files. It is often convenient, and sometimes sufficient, to write such grammar-generating code in an external language: for example, we have written Perl, Python and C++ libraries to generate XRate grammar files [9,16]. However, this approach still has the disadvantage (from a programmer's or model developer's perspective) that (a) code to generate real grammars tends to require an ungainly mix of grammar-related S-expression constants embedded in Perl/Python/C++ code, and (b) the requirement for an explicit model-generation step can delay prototyping and evaluation of new phylo-grammar models.

XRate's macro language provides an alternate way to generate repetitive models within XRate, without having to resort to external code-generating scripts. This allows the model-specifying code to remain compact, readable, and easy to edit. As we report in this manuscript, the XRate grammar format now also natively includes a Scheme-based scripting language that can be embedded directly within grammar files, whose syntax blends seamlessly with the S-expression format used by XRate and whose functional nature fits XRate's problem domain. We provide here examples of common phylogenetic models which make use of various macro features, and refer the reader to the online documentation for a complete introduction to XRate's macro features: http://biowiki.org/XrateMacros. All of the code snippets presented here are available as minimal complete grammars in Text S1. The full, trained grammars corresponding to those presented here are available as part of DART. This correspondence is described here: http://biowiki.org/XratePaper2011

A repetitively-structured HMM specified using simple macros. Probabilistic models for the evolution of biological sequences tend to contain repetitive structure. Sometimes, this structure arises as a reflection of symmetries in the phylo-grammar; other times, it arises due to structure in the data, such as the tree or the alignment. While small repetitive models can be written manually, developing richer evolutionary models and grammars often demands writing code to model the underlying structure.

Markov chain symmetry. The most familiar source of repetition derives from the substitution model's structure: different substitutions share parameters based on prior knowledge or biological intuition. Perhaps most repetitive is the Jukes-Cantor model for DNA. The matrix entries Q_{ij} denote the rate of substitution from i to j:

$$Q^{JC} = \left\{ \begin{array}{ccccc} & A & C & G & T \\ A & * & u & u & u \\ C & u & * & u & u \\ G & u & u & * & u \\ T & u & u & u & * \\ \end{array} \right\}$$

Here u is an arbitrary positive rate parameter. The $*$ character denotes the negative sum of the remaining row entries (here equal to $-3u$ in every case). The parameter u is typically set to $1/3$ in order that the stochastic process performs, on average, one substitution event per unit of time.

This matrix can be specified in XRate with two nested loops over alphabet tokens. Each loop over alphabet tokens has the form (&foreach-token X expression…) where expression… is a construct to be expanded for each alphabet token X. Here, expression sets the substitution rate between each pair of source and destination tokens (except for the case when the source and destination tokens are identical, for which case we simply generate an empty list, (), which will be ignored by the XRate grammar parser). We do not explicitly need to write the negative values of the on-diagonal matrix elements (labeled $*$ in the above description of the matrix); XRate will figure these out for itself. To check whether source and destination tokens are equal in the loop, we use a conditional &if statement, which has the form (&if (condition) (expansion-if-true) (expansion-if-false)). The condition is implemented using the &eq macro, which tests if its two arguments are equal. Putting all these together, the nested loops look like this:

```
(&foreach-token tok1
   (&foreach-token tok2
      (&if (&eq tok1 tok2)
         () ;; If tok1 = = tok2, expand to an empty list (ignored by parser)
         (mutate (from (tok1)) (to (tok2)) (rate u)))))
```

While this illustrates XRate's looping and conditional capabilities, such a simple model would almost be easier to code by hand. For a slightly more complex application, we turn to the model of Pupko *et al* in their 2008 work. In their RASER program the authors used a chain augmented with a latent variable indicating "slow" or "fast" substitution. Reconstructing ancestral sequences on an HIV phylogeny allowed them to infer locations of transitions between slow and fast modes - indicating a possible gain or loss of selective pressure [25]. The chain shown below, Q^{RASER}, shows a simplified version of their model: substitutions *within* rate classes occur according to a JC69 model scaled by rate parameters s and f (slow and fast, respectively), and transitions *between* rate classes occur with rates r_{sf} and r_{fs} (slow → fast and fast → slow, respectively).

$$Q^{RASER} = \left\{ \begin{array}{c|cccc|cccc} & A_s & C_s & G_s & T_s & A_f & C_f & G_f & T_f \\ A_s & * & us & us & us & r_{sf} & 0 & 0 & 0 \\ C_s & us & * & us & us & 0 & r_{sf} & 0 & 0 \\ G_s & us & us & * & us & 0 & 0 & r_{sf} & 0 \\ T_s & us & us & us & * & 0 & 0 & 0 & r_{sf} \\ \hline A_f & r_{fs} & 0 & 0 & 0 & * & uf & uf & uf \\ C_f & 0 & r_{fs} & 0 & 0 & uf & * & uf & uf \\ G_f & 0 & 0 & r_{fs} & 0 & uf & uf & * & uf \\ T_f & 0 & 0 & 0 & r_{fs} & uf & uf & uf & * \end{array} \right\}$$

While this chain contains four times as many rates as the basic JC69 model, there are only five parameters: u,s,f,r_{sf},r_{fs} since the model contains repetition via its symmetry. While manual implementation is possible, the model can be expressed in just a few lines of XRate macro code. Further, additional "modes" of substitution (corresponding to additional quadrants in the matrix above) can be added by editing the first two lines of the following code.

XRate represents latent variable chains as tuples of the form (state class), where state is a particular state of the Markov chain and class is the value of a hidden variable. In this case, standard DNA characters are augmented with a latent variable indicating substitution rate class: A_f indicates an **A** which evolves "fast." The following syntax is used to declare a latent variable chain (in this case, this variable may take values s or f), with the row tag specifying CLASS as the Stockholm #=GR identifier for per-sequence, per-column annotations:

```
(hidden-class (row CLASS) (label (s f)))
```

Combining loops, conditionals, hidden classes, and the (&cat LIST) function (which concatenates the elements of LIST), we get the following XRate code for the RASER chain:

```
(rate (s 0.1) (f 2.0) (r_sf 0.01) (r_fs 0.01) (u 1.0))
(chain
  (hidden-class (row CLASS) (label (s f)))
  (terminal RASER)
  (&foreach class1 (s f)
    (&foreach class2 (s f)
      (&foreach-token tok1
        (&foreach-token tok2
          (&if (&eq class1 class2)
            (&if (&eq tok1 tok2)
              () ;; if class1==class2 && tok1==tok2, expand to empty list (will be ignored)
              ;; The following line handles the case (class1==class2 && tok1!=tok2)
              (mutate (from (tok1 class1)) (to (tok2 class2)) (rate u class1)))
            (&if (&eq tok1 tok2)
            ;; The following line handles the case (class1!=class2 && tok1==tok2)
            (mutate (from (tok1 class1)) (to (tok2 class2)) (rate (&cat r_ class1 class2)))
            ())))))))) ;; if class1!=class2 && tok1!=tok2, expand to empty list (ignored)
```

Phylo-HMM-induced repetition. The previous examples both involved specifying the Markov chain component of a phylo-grammar. Coupled with a trivial top-level grammar (a START state and an EMIT state which emits the chain via the EMIT* pseudoterminal), these models describe an alignment where each column's characters evolve according to the same substitution model. A common extension to this is using sequences of hidden states which generate alignment columns according to different substitution models. These "phylo-grammars" (which can include phylo-SCFGs and the more restricted phylo-HMMs) allow modelers to describe and/or detect alignment regions exhibiting different evolutionary patterns. Phylo-HMMs model left-to-right correlations between alignment columns, and phylo-SCFGs are capable of modeling nested correlations (such as "paired" columns in an RNA secondary structure). Readers unfamiliar with phylo-grammars may benefit from relevant descriptions and links available here: http://biowiki.org/PhyloGrammars, animations available here: http://biowiki.org/PhyloFilm, and the original paper describing XRate [9].

We outline here a phylo-HMM that is simple to describe, but would take a substantial amount of code to implement without XRate 's macro language. The model is based on PhastCons, a program by Siepel *et al* which uses an HMM whose three states (or, in XRate terminology, nonterminals) use substitution models differing only by rate multipliers [26]. This model, depicted schematically in Figure 1, can be used to detect alignment regions evolving at different rates. If the rates of each hidden state correspond to quantiles of the Gamma distribution, then summing over hidden states of this model is equivalent to the commonly-used Gamma model of rate heterogeneity. We provide this grammar in Text S1, which is essentially identical to the PhastCons grammar with *n* states except for its invocation of a Scheme function returning the *n* Gamma-derived rates for a given shape parameter. We can define such a model in XRate easily due to the symmetric structure: all three nonterminals have similar underlying substitution models (varying only by a multiplier) and also similar probabilities of making transitions to other nonterminals via grammar transformation rules.

The grammar will have nonterminals named "1", "2"…up to numNonTerms, each one associated with a rate parameter (r_1, r_2…) and substitution chain (chain_1, chain_2…). To express this grammar in XRate macro code, we'll need to declare each of these nonterminals, the production rules which govern transitions between them, rate parameters, and the nonterminal-associated substitution chains. (For a fully-functional grammar, an alphabet is also needed; these are omitted in code snippets included in the main text, but the corresponding grammars in Text S1 contain alphabets.)

First, define how many nonterminals the model will have: adding more nonterminals to the model later on can be done simply by adjusting this variable. We define a SEED value to initialize the rate parameters (this is not a random number seed, but rather an initial guess at the parameter value necessary for the EM algorithm to begin), which is done inside a foreach-integer loop using the numNonterms variable. The (foreach-integer X (1 K) expression) expands expression for all values of X from 1 to K. In this case, we define a rate parameter for each of our nonterminals 1.. K.

```
(&define numNonterms 3)
(&define SEED 0.001)
(&foreach-integer nonterminal (1 numNonterms)
  (rate ((&cat r_ nonterminal) SEED)))
```

Next, define a Markov chain for each nonterminal: all make use of the same underlying substitution model (e.g. JC69 [1], HKY85 [3]) whose entries are stored as Q_a_b for the transition rate between characters a and b. This "underlying" chain must be

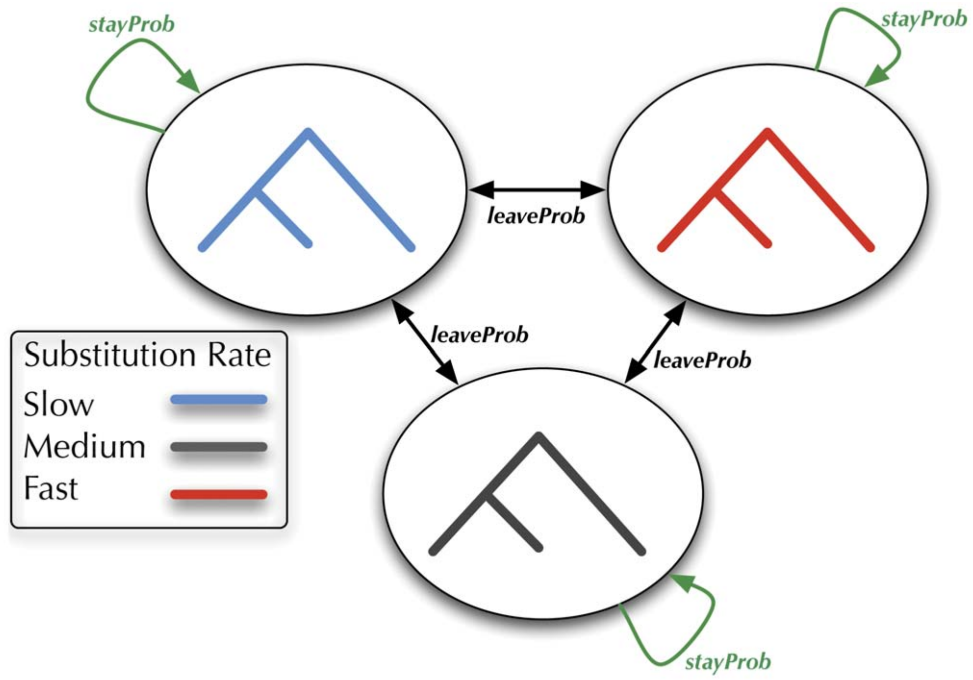

Figure 1. The model used by PhastCons, a 3-nonterminal HMM with rate multipliers, is compactly expressed by XRate's macro language. Different nonterminal have different evolutionary rates, but they all share the same underlying substitution model. Transition probabilities are shared: a transition between nonterminals happens with probability *leaveProb*, and self-transitions happen with probability *stayProb*. This model (with any number of nonterminals) can be expressed in XRate's macro language in approximately 20 lines of code.
doi:10.1371/journal.pone.0036898.g001

defined elsewhere - either in an included file (using the (&include) directive), or directly in the grammar file. For instance, we could re-use the JC69 chain, declaring rate parameters for later use:

```
(&foreach-token tok1
    (&foreach-token tok2
        (&if (&eq tok1 tok2)
            () ;; If tok1 = = tok2, expand to an empty list (ignored by parser)
            (rate (&cat Q_ tok1 _ tok2) u))))
```

Each nonterminal has an associated substitution model which is Q_a_b scaled by a different rate multiplier r_nonterminal. Using an integer loop, we create a chain for each nonterminal using the rate parameters we defined in the two previous code snippets:

```
(&foreach-integer nonterminal (1 numNonterms)
    (chain
        (terminal (&cat chain_ nonterminal))
        (&foreach-token tok1
            (&foreach-token tok2
                (&if (&eq tok1 tok2)
                    ()
                    (mutate (from (tok1)) (to (tok2))
                    (rate (&cat Q_ tok1 _ tok2) (&cat r_ nonterminal))))))))
```

Next, define the production rules which govern the nonterminal transitions. For simplicity of presentation (but not required), we assume here that transitions between nonterminals all occur with probability proportional to leaveProb, and all self-transitions have probability stayProb.

The pgroup declaration defines a probability distribution over a finite outcome space, with the parameters declared therein normalized to unity during parameter estimation. In this grammar we declare stayProb and leaveProb within a pgroup since they describe the two outcomes at each step of creating the alignment: staying at the current nonterminal or moving to a different one.

```
(pgroup (stayProb 0.9) (leaveProb 0.1))
(&foreach-integer nonterm1 (1 numNonterms)
    ;; Each nonterminal has a transition from start
    (transform (from (start)) (to (nonterm1)) (prob (&/1 numNonterms)))
    ;; Each nonterminal can transition to end - we assign this prob 1
    ;; since the alignment length directs when this transition occurs
    (transform (from (nonterm1)) (to ()) (prob 1))
    (&foreach-integer nonterm2 (1 numNonterms)
        (&if (&eq nonterm1 nonterm2))
            ;; If nonterm1 = = nonterm2, this is a self-transition
            (transform (from (nonterm1)) (to (nonterm2)) (prob stayProb))
            ;; Otherwise, this is an inter-nonterminal transition
            ;; with probability changeProb/(numNonterms - 1)
            (transform (from (nonterm1)) (to (nonterm2))
            (prob (&/changeProb (&- numNonterms 1))))))
```

Lastly, associate each nonterminal with its specially-designed Markov chain for emitted alignment columns:

```
(&foreach-integer nonterminal (1 numNonterms)
    (transform (from (nonterminal))(to ((&cat chain_ nonterminal) (&cat nonterminal *))))
    (transform (from ((&cat nonterminal *))) (to (nonterminal))))
```

Data-induced repetition. Models whose symmetric structure depends on the input data are less common in phylogenetic analysis, perhaps because normally their implementation requires creating a new model for each new dataset to be analyzed. XRate allows the user to create models based on different parts of the input data, namely the tree and the alignment, "on the fly" via its macro language. This is accomplished by making use of the tree iterators (e.g. &BRANCHES, &NODES, and &LEAVES) and alignment data (e.g. &COLUMNS) to create nonterminals and/or terminal chains associated with these parts of the input data.

In their program DLESS, Haussler and colleagues used such an approach in a tree-dependent model to detect lineage-specific selection. Their model used a phylo-HMM with different nonterminals for each tree node, with the substitution rate below this node scaled to reflect gain or loss of functional elements [26]. We show a simplified form of their model as a schematic in Figure 2, with blue colored branches representing a slowed evolutionary rate.

Using XRate's macros we can express this model in a compact way just as was done with the PhastCons model. Since both models use a set of nonterminals with their own scaled substitution models, we need simply to replace the integer-based loop (&foreach-integer nonterminal (1 numNonterms) expression) with the tree-based loop (&foreach-node state expression) to create a nonterminal for each node in the tree. Then, define each node-specific chain as a *hybrid chain*, such that the chain associated with tree node *n* has all the branches below node *n* scaled to reflect heightened selective pressure. Hybrid chains, substitution processes which vary across the tree, are discussed briefly in the section on "Recent enhancements to XRate", and the details of their specification is thoroughly covered in the XRate format documentation, available here: http://biowiki.org/XrateFormat. A minimal working form of the DLESS-style grammar included in Text S1.

A repetitively-structured codon model specified using Scheme functions. While XRate's macro language is very flexible, there are some relatively common models that are difficult to express within the language's constraints. For example, a Nielsen-Yang codon matrix incorporating transition bias and selection has nearly 4,000 entries whose rates are determined by the following criteria:

$$Q^{NY}_{ij}=\begin{cases}0 & \text{if } i \text{ and } j \text{ differ at more than one position}\\ \pi_j & \text{if } i \text{ and } j \text{ differ by a synonymous transversion}\\ \kappa\pi_j & \text{if } i \text{ and } j \text{ differ by a synonymous transition}\\ \omega\pi_j & \text{if } i \text{ and } j \text{ differ by a nonsynonymous transversion}\\ \omega\kappa\pi_j & \text{if } i \text{ and } j \text{ differ by a nonsynonymous transition}\end{cases}$$

This sort of Markov chain is difficult to express in XRate's macro language since its entries are determined by aspects of the codons (synonymous changes and transitions/transversions) which in turn depend on knowledge of the properties of nucleotides and codons that would have to be hard-coded directly into the loops and conditionals afforded by XRate's macros. The conditions on the right side of the above equation are better framed as values returned from a function: given a pair of codons, the function returns the "type" of difference between them, which in turn determines the rate of substitution between the two codons.

Scheme extensions. It is this sort of situation which motivates extensions to XRate that are more general-purpose than the simple macros described up to this point. There are several valid choices for the programming language that can be used to implement such extensions. For example, a chain such as Q^{NY} can be generated fairly easily by way of a Perl or Python script tailored to generate XRate grammar code. While this is a convenient scripting mechanism for many users (and is perfectly possible with XRate), it tends to lead to an awkward mix of code and embedded data (i.e. snippets of grammar-formatting text). This obscures both the generating script and the final generated grammar file (the former due to the code/data mix, and the latter due to sheer size).

Another choice of programming language for implementing XRate extensions, which suffers slightly less from these limitations, is Scheme. As XRate 's macro language is based on Lisp (the parent language to Scheme), the syntaxes are very similar, so the "extension" blends naturally with the surrounding XRate grammar file. Scheme is inherently functional and is also "safe" (in that it has garbage collection). Lastly, data and code have equivalent formats in Scheme, enabling the sort of code/data mingling outlined above.

To implement the Q^{NY} chain in XRate, we can use the XRate Scheme standard library (found in dart/scheme/xrate-stdlib.scm). This standard library implements all the necessary functions to define the Nielsen-Yang model, with the genetic code implemented as a Scheme association list (facilitating easy substitution of alternate genetic codes, such as the mitochondrial code) as well as a wrapper function to initialize the entire model.

Without stepping through every detail of the Scheme implementation of the Nielsen-Yang model in the XRate standard library, we will simply note that this implementation (the Nielsen-Yang model on a DNA alphabet) is available via the following XRate code (the include path to dart/scheme is searched by default by the Scheme function load-from-path):

```
(&scheme
  (load-from-path "xrate-stdlib.scm")
  xrate-dna-alphabet
  (xrate-NY-grammar))
```

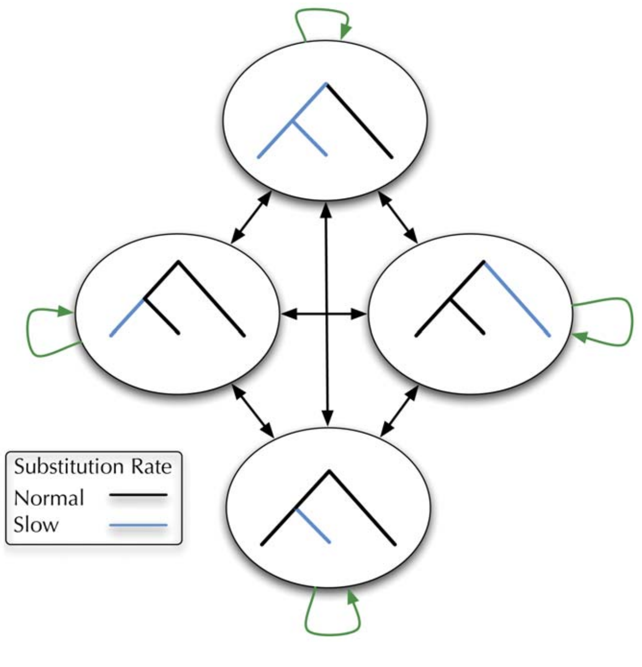

Figure 2. A schematic of a DLESS-style phylo-HMM: each node of the tree has its own nonterminal, such that the node-rooted subtree evolves at a slower rate than the rest of the tree. Inferring the pattern of hidden nonterminals generating an alignment allows for detecting regions of lineage-specific selection. Expressing this model compactly in XRate 's macro language allows it to be used with any input tree without having to write data-specific code or use external model-generating scripts.
doi:10.1371/journal.pone.0036898.g002

Note that xrate-dna-alphabet is a simple variable, but xrate-NY-grammar is a function and is therefore wrapped in parentheses (as per the syntax of calling a function in Scheme). The reason that xrate-NY-grammar is a function is so that the user can optionally redefine the genetic code, which (as noted above) is stored as a Scheme association list, in the variable codon-translation-table (the standard library code can be examined for details).

A macro-heavy grammar for RNA structures in protein-coding exons. As a final example of the possibilities that XRate's new model-specification features enable, we present a new grammar for predicting RNA structures which overlap protein-coding regions. XDecoder is based closely on the RNADecoder grammar first developed by Pederson and colleagues [27]. This grammar is designed to detect phylogenetic evidence of conserved RNA structures, while also incorporating the evolutionary signals brought on by selection at the amino-acid level. In eukaryotes, RNA structure overlapping protein coding sequence is not yet well-known, but in viral genomes this is a common phenomenon due to constraints on genome size acting on many virus families. XDecoder is available as an XRate grammar, linked here: http://biowiki.org/XratePaper2011.

Motivation for implementation. Our endeavor to re-implement the RNADecoder grammar was based both on practical and methodological reasons. The original RNADecoder code is no longer maintained, but performs well on published viral datasets [28]. Running RNADecoder on an alignment of full viral genomes is quite involved: the alignment must first be split up into appropriately-sized chunks (300 columns), converted to COL format [29], and linked to a tree in a special XML file which directs the analysis. The grammar and its parameters, also stored in an XML format, are difficult to read and interpret. RNADecoder attains remarkably higher specificity in genome-wide scans as compared to protein-naive prediction programs like PFOLD [15] or MFOLD [30].

Using XDecoder. We developed our own variant of the RNADecoder model as an XRate grammar, called XDecoder. This would have been a protracted task without XRate's macro capabilities: the expanded grammar is nearly 4,000 lines of code. Using XRate's macros, the main grammar (excluding the pre-estimated dinucleotide Markov chain) is only 100 lines of macro code. Starting with an alignment of full-length *poliovirus* genomes, annotated with reading frames, an analysis can be run with a single simple command:

```
xrate -g XDecoder.eg −l 300 -wig polio.wig polio.stk > polio_annotated.stk
```

This runs XRate with the XDecoder grammar on the Stockholm-format alignment polio.stk, allowing no more than 300 positions between paired columns, creating the wiggle file polio.wig, annotating the original alignment with maximum likelihood secondary structure and rate class indicators, and writing the annotated alignment to the the file polio_annotated.stk.

Each analysis with RNADecoder requires an XML file to coordinate the alignment and tree as well as direct parts of the analysis (training and annotation). XRate reads Stockholm format alignments which natively allows for alignment-tree association, enabling simple batch processing of many alignments. The grammar can be run on arbitrarily long alignments, provided a suitable maximum pair length is specified via the −l N argument. This prevents XRate from considering any pairing whose columns are more than N positions apart, effectively limiting both the memory usage and runtime.

Training the grammar's parameters, which may be necessary for running the grammar on significantly different datasets, is also accomplished with a single command:

```
xrate -g XDecoder.eg −l 300 -t XDecoder.trained.eg polio.stk
```

The results of an analysis using XDecoder are shown in Figure 3, together with gene and RNA structure annotations. Also shown are three related analyses (all done using XRate grammars): PhastCons conservation, coding potential, and pairing probabilities computed using PFOLD. These three separate analyses reflect the signals that XDecoder must tease apart in order to reliably predict RNA structures. DNA-level conservation could be due to protein-coding constraints, regional rate variation, pressure to maintain a particular RNA structure, or a combination of all three. Using codon-position rate multipliers, multiple rate classes, and a secondary structure model, XDecoder unifies all of these signals in a single phylogenetic model, resulting in the highly-specific predictions shown at the top of Figure 3. The full JBrowse instance is provided as a demo at http://jbrowse.org/poliovirus-xrate-demo-by-oscar-westesson/.

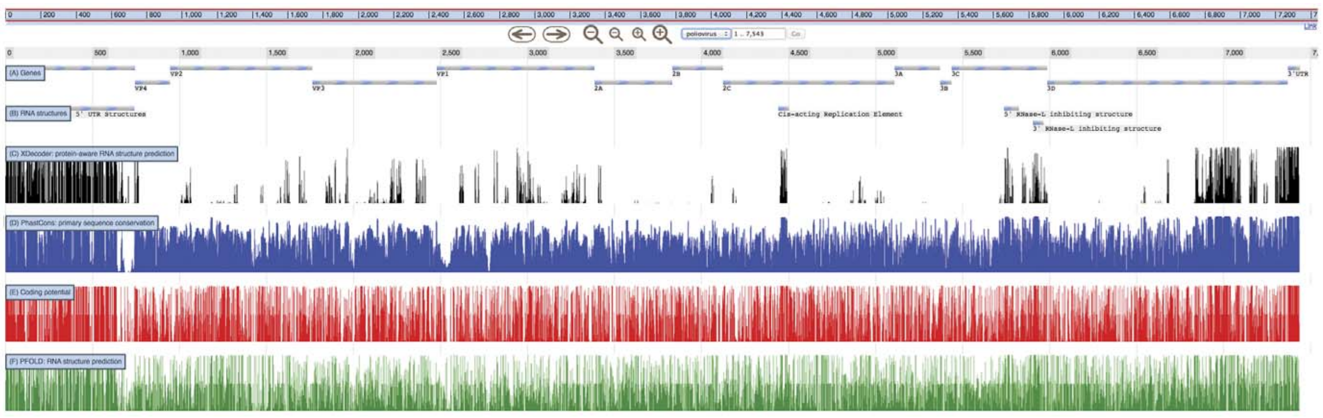
(A) Genes
VP4
VP2
VP3
VP1
2A
2B
2C
3A
3B
3C
3D
3' UTR
(B) RNA structures
5' UTR structures
Cis-acting Replication Element
3' RNase-L inhibiting structure
3' RNase-L inhibiting structure
(C) XDecoder: protein-aware RNA structure prediction
(D) PhastCons: primary sequence conservation
(E) Coding potential
(F) PFOLD: RNA structure prediction

Figure 3. Data from several XRate analyses, shown alongside genes (A) and known RNA structures (B) in *poliovirus*. XDecoder (**C**) recovers all known structures with high posterior probability and predicts a promising target for experimental probing (region 6800–7100). XDecoder was run on an alignment of 27 *poliovirus* sequences with the results visualized as a track in JBrowse [32] via a wiggle file. Alongside XDecoder probabilities are the three signals which XDecoder aims to disentangle: (**D**) conservation, (**E**) coding potential, and (**F**) RNA structure. Paradoxically, the CRE and RNase-L inhibition elements show both conservation and coding sequence preservation, whereas PFOLD's predictions show only a slight increase in probability density around the known structures. XDecoder is the only grammar which returns predictions of reasonable specificity. The full JBrowse instance is included as Text S 2.
doi:10.1371/journal.pone.0036898.g003

Recent enhancements to XRate

Lineage-specific models. All Markov chains in phylo-grammars describe the evolution of characters starting at the root and ending at the tips of the tree. In lineage-specific models, or *hybrid chains* in XRate terminology, the requirement that all branches share the same substitution process is relaxed. Phylogenetic analysis is often used to detect a departure from a "null model" representing some typical evolutionary pattern. Standard applications of HMMs and SCFGs focus on modeling this departure on the alignment level, enabling different columns of the alignment to show different patterns of evolution. Using hybrid chains, users can explicitly model differences in evolution across parts of the tree. By combining a hybrid chain with grammar nonterminals, this could be used to detect alignment regions (i.e. subsets of the set of all sites) which display unusually high (or low) mutation rates in a particular part of the tree, such as in the DLESS model described in the section on "Data-induced repetition". The details of specifying such models are contained within the XRate format documentation, at http://biowiki.org/XrateFormat.

Ancestral sequence reconstruction. A phylo-grammar is a generative model: it generates a hidden parse tree, then further generates observed data conditional on that parse tree. The observed data here is an alignment of sequences; the hidden parse tree describes which alignment columns are to be generated by the evolutionary models associated with which grammar nonterminals. Inference involves reversing the generative process: reconstructing the hidden parse structure and evolutionary trajectories that explain the alignment.

The original version of XRate was focused on reconstructing the parse tree, for the purposes of annotating hidden structures such as gene boundaries or conserved regions. A newly-implemented feature in XRate allows an additional feature: reconstruction of ancestral sequences. This functionality is already implicit in the phylogenetic model: no additional modification to the grammar is necessary to enable reconstruction. The user can ask XRate to return the most probable ancestral sequence at each internal node, or the entire posterior distribution over such sequences, via the -ar and -arpp command-line options. Since XRate does marginal state reconstruction, the character with the highest posterior probability returned by the -arpp option will always correspond to the single character returned by the -ar option. Ancestral sequence reconstruction can be used to answer paleogenetic questions: what did the sequence of the ancestor to all of clade X look like? Similarly, evolutionary events such as particular substitutions or the gain or loss of function (also called trait evolution) can be pinpointed to particular branches.

Direct output of GFF and Wiggle annotations. XRate allows parse annotations to be written out directly in common bioinformatics file formats: GFF (a format for specifying coordinates of genomic features) [22] and WIG (a per-base format for quantitative data) [31].

This allows a direct link between XRate and visualization tools such as JBrowse [32], GBrowse [33], the UCSC Genome Browser [34],and Galaxy [35], allowing the results of different analyses to be displayed next to one another and/or processed in a unified framework.

GFF: Discrete genomic features. GFF is a format oriented towards storing genomic features using 9 tab-delimited fields: each line represents a separate feature, with each field storing a particular aspect of the feature (e.g. identifier, start, end, etc). With XRate, a common application is using GFF to annotate an alignment with features corresponding to grammar nonterminals. For instance, using a gene-prediction grammar one could store the predicted start and end points of genes together with a confidence measure. Similarly, predicted RNA base pairs could be represented in GFF as one feature per pair, with start and end positions indicating the paired positions.

WIG: Quantitative values for each column(s). Wiggle format stores a quantitative value for a single or group of positions. This can be especially useful to summarize a large number of possibilities as a single representative value. For instance, when predicting regions of structured RNA, XRate may sum over many thousands of possible structures. We can summarize the model's results with the posterior probability that each column is involved in a base-pairing interaction.

The Dart Scheme (Darts) interpreter

Another way to use XRate, instead of running it from the command line, is to call it from the Scheme interpreter (included in DART). The compiled interpreter executable is named "darts" (for "DART Scheme"). This offers a simple yet powerful way to create parameter-fitting and genome annotation workflows. For example, a user could train a grammar on a set of alignments, then use the resulting grammar to annotate a set of test alignments.

Darts, in common with the Scheme interpreter used in XRate grammars, is implemented using Guile (GNU's Ubiquitous Intelligent Language for Extension: http://www.gnu.org/software/guile/guile.html). Certain commonly-encountered bioinformatics objects, serializable via standard file formats and implemented as C++ classes within XRate, are exposed using Guile's "small object" (smob) mechanism. Currently, these types include Newick-format trees and Stockholm-format alignments. API calls are provided to construct these "smobs" by parsing strings (or files) in the appropriate format. The smobs may then be passed directly as parameters to XRate API calls, or may be "unpacked" into Scheme data structures for individual element access. Guile encourages sparing use of smobs; consequently, smobs are used within Darts exclusively to implement bioinformatic objects that already have a broadly-used file format (Stockholm alignments and Newick trees). In contrast, formats that are newly-introduced by XRate (grammars, alphabets and so forth) are all based on S-expressions, and so may be represented directly as native Scheme data structures.

The functions listed in Text S2 provide an interface between Scheme and XRate . Together with the functions in the XRate -scheme standard library and Scheme's native functional scripting abilities, a broad array of models and/or workflows are possible. For instance, one could estimate several sets of parameters for Nielsen-Yang models using groups of alignments, and then embed each one in a PhastCons-style phylo-HMM, finally using this model to annotate a set of alignments. While this and other workflows could be accomplished in an external framework (e.g. Make, Galaxy [35]), Darts provides an alternate way to script XRate tasks using the same language that is used to construct the grammars.

Supporting Information

Text S1 contains example grammars referred to in the text, as well as small and large test Stockholm alignments. The alignment of *poliovirus* genomes along with the grammars used to produce Figure 3 are also included along with a Makefile indicating how the data was analyzed. Typing make help in the directory containing the Makefile will display the demonstrations available to users.
(ZIP)

Text S2 contains tables describing the Scheme- XRate functions available in Darts.
(PDF)

Text S3 contains a glossary of XRate terminology.
(PDF)

Author Contributions

Conceived and designed the experiments: OW IH. Performed the experiments: OW IH. Analyzed the data: OW. Contributed reagents/materials/analysis tools: OW IH. Wrote the paper: OW IH.

References

1. Jukes TH, Cantor C (1969) Evolution of protein molecules. In: Mammalian Protein Metabolism. New York: Academic Press. pp 21–132.
2. Kimura M (1980) A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. Journal of Molecular Evolution 16: 111–120.
3. Hasegawa M, Kishino H, Yano T (1985) Dating the human-ape splitting by a molecular clock of mitochondrial DNA. Journal of Molecular Evolution 22: 160–174.
4. Felsenstein J (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. Journal of Molecular Evolution 17: 368–376.
5. Meyer IM, Durbin R (2004) Gene structure conservation aids similarity based gene prediction. Nucleic Acids Research 32: 776–783.
6. Eddy SR (1998) Profile hidden Markov models. Bioinformatics 14: 755–763.
7. Knudsen B, Hein J (1999) RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. Bioinformatics 15: 446–454.
8. Garber M, Guttman M, Clamp M, Zody M, Friedman N, et al. (2009) Identifying novel constrained elements by exploiting biased substitution patterns. Bioinformatics.
9. Klosterman PS, Uzilov AV, Bendana YR, Bradley RK, Chao S, et al. (2006) XRate: a fast prototyping, training and annotation tool for phylo-grammars. BMC Bioinformatics 7.
10. Slater GSC, Birney E (2005) Automated generation of heuristics for biological sequence comparison. BMC Bioinformatics 6: 31.
11. Birney E, Durbin R (1997) Dynamite: a exible code generating language for dynamic programming methods used in sequence comparison. In: Gaasterland T, Karp P, Karplus K, Ouzounis C, Sander C, et al., editor. Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology. Menlo Park, CA: AAAI Press. pp 56–64.
12. Lunter G (2007) HMMoC–a compiler for hidden Markov models. Bioinformatics 23: 2485–2487.
13. Drummond AJ, Rambaut A (2007) BEAST: Bayesian evolutionary analysis by sampling trees. BMC Evolutionary Biology 7.
14. Yang Z, Nielsen R, Goldman N, Pedersen AM (2000) Codon-substitution models for heterogeneous selection pressure at amino acid sites. Genetics 155: 432–449.
15. Knudsen B, Hein J (2003) Pfold: RNA secondary structure prediction using stochastic context-free grammars. Nucleic Acids Research 31: 3423–3428.
16. Heger A, Ponting CP, Holmes I (2009) Accurate estimation of gene evolutionary rates using XRATE, with an application to transmembrane proteins. Molecular Biology and Evolution 26: 1715–1721.
17. Ayres D, Darling A, Zwickl D, Beerli P, Holder M, et al. (2011) Beagle: an application programming interface and high-performance computing library for statistical phylogenetics. Systematic Biology.
18. Bradley RK, Uzilov AV, Skinner ME, Bendana YR, Barquist L, et al. (2009) Evolutionary modeling and prediction of non-coding RNAs in Drosophila. PLoS ONE 4: e6478.
19. Varadarajan A, Bradley RK, Holmes I (2008) Tools for simulating evolution of aligned genomic regions with integrated parameter estimation. Genome Biology 9.
20. The Stockholm _le format. Available: http://sonnhammer.sbc.su.se/Stockholm.html. Accessed 4 May 2012.
21. The Newick file format. Available: http://evolution.genetics.washington.edu/phylip/newicktree.html. Accessed 4 May 2012.
22. GFF: an exchange format for gene-finding features. Available: http://www.sanger.ac.uk/resources/software/gff/. Accessed 4 May 2012.
23. Saitou N, Nei M (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular Biology and Evolution 4: 406–425.
24. Searls DB, Murphy KP (1995) Automata-theoretic models of mutation and alignment. In: Rawlings C, Clark D, Altman R, Hunter L, Lengauer T, et al., editor. Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology. Menlo Park, CA: AAAI Press. pp 341–349.
25. Penn O, Stern A, Rubinstein ND, Dutheil J, Bacharach E, et al. (2008) Evolutionary modeling of rate shifts reveals specificity determinants in hiv-1 subtypes. PLoS Computational Biology 4: e1000214.
26. Siepel A, Bejerano G, Pedersen JS, Hinrichs AS, Hou M, et al. (2005) Evolutionarily conserved elements in vertebrate, insect, worm, and yeast genomes. Genome Research 15: 1034–1050.
27. Pedersen JS, Meyer IM, Forsberg R, Simmonds P, Hein J (2004) A comparative method for finding and folding RNA secondary structures within protein-coding regions. Nucleic Acids Research 32: 4925–4923.
28. Watts J, Dang K, Gorelick R, Leonard C, Bess J, et al. (2009) Architecture and secondary structure of an entire hiv-1 rna genome. Nature.
29. Col format. Available: http://colformat.kvl.dk/. Accessed 4 May 2012.
30. Zuker M (1989) Computer prediction of RNA structure. Methods in Enzymology 180: 262–288.
31. Wiggle track format. https://cgwb.nci.nih.gov/goldenPath/help/wiggle.html. Accessed : 4 May 2012.
32. Skinner ME, Uzilov AV, Stein LD, Mungall CJ, Holmes IH (2009) JBrowse: a next-generation genome browser. Genome Res 19: 1630–1638.
33. Stein L, Mungall C, Shu S, Caudy M, Mangone M, et al. (2002) The generic genome browser: a building block for a model organism system database. Genome Research 12: 1599–1610.
34. Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, et al. (2003) The human genome browser at UCSC. Genome Research 12: 996–1006.
35. Goecks J, Nekrutenko A, Taylor J, Afgan E, Ananda G, et al. (2010) Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol 11: R86.