# Biological Sequence Analysis using Probabilistic Logic Generalizations of Hidden Markov Models

PhD Thesis

**Søren Mørk**
DEPARTMENT OF SCIENCE, SYSTEMS AND MODELS
ROSKILDE UNIVERSITY

Academic Supervisor:

**Ole Skovgaard**

Submitted:

01/10/2012

# Preface

This thesis concerns biological sequence analysis using probabilistic logic generalizations of hidden Markov models (HMMs), and has been submitted according to the requirements for obtaining the PhD degree at Roskilde University, Denmark.

My PhD project was part of *the Lost Project*, a general exploration of the combination of logic programming and bioinformatics [1], funded by a NABIIT grant from the Danish Strategic Research Council [09-061842/DSF].

The work has been carried out at the Department of Science, Systems and Models, Roskilde University, the Department of Communication, Business and Information Technologies, Roskilde University and at the Department of Bioengineering, University of California - Berkeley.

The thesis consists of a core of three papers and an introduction. The papers describe probabilistic logic hidden Markov models applied to gene finding, genome annotation, and RNA/mRNA structure prediction.

The introduction starts with a short motivational background section on the importance of: i: The sequential composition of DNA, RNA and proteins. ii: The overlapping biological forms and functions imposed on genomes by the central dogma of molecular biology. iii: A generalized approach to probabilistic model development dictated by this overlap. iv: The use of (prokaryotic) gene finding as a starting point to develop and test principles for constructing models for biological sequence analysis.

Hereafter, a concise description of the probabilistic logic programming language and machine learning system PRISM is given. This is followed by a brief introduction to HMMs and how they can be implemented in PRISM. The last section of the overview details how HMMs implemented in PRISM can be extended, and integrated, in various ways to be used for modeling biological sequences and performing fundamental biological sequence analysis tasks.

The introduction ends with summaries of the papers included in the thesis, some of the perspectives of the present work, and finally, a general conclusion on the project.

I have attempted to provide an introduction that is more comprehensible than comprehensive. I assume that readers are somewhat familiar with computational biology and bioinformatics, including basic concepts of molecular biology and machine learning and general notions of probabilistic modeling and mathematical notation. A good starting point for a more thorough introduction to probabilistic logic programming is DeRaedt *et al.* 2008 [2]. For a brilliant introduction to biological sequence analysis I highly recommend Durbin *et al.* 1998 [3].

The PRISM source code for all models used in the papers and accessory scripts are available at: http://github.com/somork/

The PRISM source code of some examples of model types mentioned in the introduction that have not yet been included in manuscripts are provided in the Appendix.

I started my PhD project as a molecular evolutionary biologist with a deep interest in theoretical biology, and only some knowledge of bioinformatics. The PhD project offered to me by Ole Skovgaard and Henning Christiansen has been an extremely fortunate opportunity for me to immerse myself in computational biology in general, and biological sequence analysis in particular. I am very thankful for this. As a biologist entering into the cross-field of biology and computer science, that the field of bioinformatics is, my attention has primarily been devoted towards the elements from computer science necessary for operating in this field. Doing this, I have come to deeply cherish the fascinating world of computer science, both theoretical and applied, that probabilistic logic programming has been a (relatively unconventional) route to.

I would like to thank the following:

Anders Albrechtsen, Nicos Angelopolous, Lars Barquist, Henning Christiansen, James Cussens, Alexei Drummond, Jan Gorodkin, Rikke Hansen, Christian Theil Have, Jakob Hull Havgaard, Jenny Hedman, Jotun Hein, Ian Holmes, Manfred Jaeger, Yoshitaka Kameya, Bjarne Knudsen, Michael Knudsen, Anders Krogh, Erlend Kristiansen, Ole Torp Lassen, Ida Moltke, Tobias Mourier, Kasper Munch, Chris Mungal, Thomas Mørk, Jakob Skou Pedersen, Tune Pers, Christa Persson, Matthieu Petit, Taisuke Sato, Mitch Skinner, Ole Skovgaard, Marc Suchard, Bjarne Vincents, Oscar Westesson, Eske Willerslev, Rasmus Grønfeldt Winther, and Sine Zambach,

I would also like to thank my wife Marianne and our children Simon, Alva, Frida and Siv.

# Summary

Genomes are templates for the sequential synthesis of DNA, RNA and proteins. Modeling the resultant overlap of biological form and function requires developments of powerful sub-models and principled model integration. The ease of which this is achieved is greatly increased by the use of a generalized prototyping environment.

Here, I describe a range of probabilistic sequence models based on extensions of hidden Markov models (HMMs), and combinations hereof, that can be used for a number of basic tasks of biological sequence analysis. The HMM variants presented here are implemented in the probabilistic logic programming language and machine learning system PRISM. PRISM provides the ability to produce, combine, test and compare different probabilistic sequence models, using principled approaches, within a single unified framework.

The repertoire of models presented include: (i): Models of prokaryotic protein coding genes based on model types such as mixed memory HMMs. (ii): Models of gene sequence composition for genome annotation based on delete HMMs. (iii): Models of RNA secondary structures based on stochastic context free grammar (SCFG)-like HMMs. (iv): Models for homology assignment based on multi-sequence HMMs and multi-sequence SCFG-like HMMs. (v): Models of overlapping genomic structures based on factorial HMMs, factorial SCFG-like HMMs or combined factorial HMM/SCFG-like HMMs.

The combined work demonstrates the feasibility of using probabilistic logic programming for biological sequence analysis, general principles for improving genome annotation and gene structure prediction, and potential for application on additional tasks within biological sequence analysis.

# Dansk Resume

Genomer fungerer som skabeloner for den sekventielle syntese af både DNA, RNA og proteiner. For at kunne modelere det deraf følgende overlap af biologisk form og funktion kræves det, at der både udvikles effektive modeller af de forskellige typer funktioner, og effektive måder at integrere sådanne modeller på. Udviklingen af sdanne modeller kan gres vsentligt nemmere ved at benytte prototypning af modelstrukturer i et generaliseret udviklingsmilj.

I denne afhandling beskriver jeg resultater fra arbejdet med udviklingen af en række probabilistiske sekvensmodeller, med udgangspunkt i forskellige typer hidden Markov modeller (HMM'er) og kombinationer heraf.

HMM varianterne er alle implementeret i det probabilistisk-logiske programmeringsprog og maskinlæringssystem PRISM. Dette er gjort for systematiskt at kunne producere, kombinere, teste og sammenligne forskellige probabilistiske sekvensmodeller.

Samlingen af modeller inkluderer: (i): Modeller af prokaryote proteinkodende gener baseret på bl.a. mixed memory HMM'er. (ii): Modeller af den sekventielle rækkefølge af gener i et genom, baseret p delete HMM'er. (iii): Modeller af RNA sekundære strukturer baseret på stokastisk kontext frie grammatik (SCFG)- lignende HMM'er. (iv): Modeller til alignment, baseret på multi-sekvens HMM'er og multi-sekvens SCFG-lignende HMM'er. (v): Modeller af overlappende genomiske strukturer baseret på faktorielle HMM'er, faktorielle SCFG-lignede HMM'er og kombinerede faktorielle HMM/SCFG-ligende HMM'er.

Det samlede arbejde demonstrerer brugbarheden af probabilistisk logikprogrammering til biologisk sekvensanalyse, generelle principper til at forbedre genom-annotering og genstruktur modelering, samt potentiale til yderligere anvendelse indenfor biologisk sekvensanalyse.

# List of Papers

## Paper I

Mørk S and Holmes I (2012) Evaluating Bacterial Gene-Finding HMM Structures as Probabilistic Logic Programs. *Bioinformatics*. 28: 636-642

## Paper II

Have CT and Mørk S. A Probabilistic Genome-Wide Reading Frame Sequence Model. (*in review - PLOS One*)

## Paper III

Mørk S, Have CT and Havgaard JH. Prototyping RNA Models as Probabilistic Logic Programs. (*in prep.*)

First authorships underlined.

# Contents

# Introduction

## Biological Sequence Analysis

The elucidation of the sequential structure of DNA [4] was the most important biological discovery of the 20th century.

The sequential structure of DNA is the key to understanding how cells maintain and regulate the processes that keep them alive. Through the biologically essential processes of replication (DNA $\rightarrow$ DNA), transcription (DNA $\rightarrow$ RNA) and translation (RNA $\rightarrow$ Protein), DNA sequences serve as the template for the sequential synthesis of DNA, RNA and proteins alike, the central dogma of molecular biology [5].

The directional flow of information, in collection with natural selection acting on the rate of surviving replicons, imposes constraints on the composition of DNA sequences both in terms of the functions they encode, as well as in terms of the sequential composition of their ancestors (Figure 1).

Discerning the different sources, and complicated interactions, of the constraints that govern the sequential composition of the DNA nucleotides; A(denosine), C(ytosine), G(uanine) and T(hymidine), that constitutes genomes, is one of the major goals of biology.

Within recent decades enormous amounts of DNA sequence data have become publicly available. As per January 2012, GenBank, the primary deposit of genome sequence data, contained more than one hundred million sequences, comprising more than one hundred billion nucleotides (http://www.ncbi.nlm.nih.gov/genbank/), including thousands of complete whole genome sequences from vira, prokaryotes and eukaryotes.

Probabilistic sequence models serve as a rigorous basis for sensible automation of important computational tasks involving such sequence data [3], and are paramount for obtaining quantitative descriptions and predictions of sequential form and function.

The current biological sequence analysis model plethora ranges from relatively simple models of gene structures, that can be used for single sequence gene finding [6], over more complicated models for alignment[7] and phylogenetic inferences[8], ancestral reconstruction [9], to very complicated models for the simultaneous execution of multiple such tasks [10, 11]. The basic biological sequence analysis tasks of structure prediction, alignment and evolutionary inferences are inherently related due to the constraints imposed by the central dogma (Figure 2).

In the following, I describe how using a unified framework for developing, integrating and testing models, the probabilistic logic programming language and machine learning system PRISM [12], can be used for developing powerful probabilistic sequence models for biological sequence analysis.

The strategy applied is to start from simple models, make generalizations of them and proceed with incremental changes according to general model types and features needed to perform the tasks at hand.

Hidden Markov model (HMM) variants for gene finding have been used as a starting point for the following reasons:

- The single sequence HMMs are very simple models, yet powerful enough to be applicable for biological sequence analysis problems such as gene finding.
- Due to the linearity of the process of protein translation, and the important short distant interactions imposed by codon structure, gene finding of protein coding genes have traditionally been based on a limited number of HMM variants.
- Prokaryotic genes have a relatively simple structure that is easier to capture with HMMs than that of the truncated genes of eukaryotes, or the very compact viral genes that frequently are overlapping other genes or harboring structural components.
- Gene finding is a relatively simple binary classification task. This enables evaluation of model performance through relative simple statistical means such

as sensitivity/specificity measures or receiver operator characteristics (ROC) curves. Such simple measures are less easily applied to more complicated biological sequence analysis tasks such as alignment or phylogenetic inferences.

- Gene finding is experimentally verifiable. It is possible (though quite laborious) to establish a reliable golden standard for gene finding experimentally, making the classification task much more reliable. This is not possible for alignment and phylogenetic analysis, that have to rely rely on internal consistency due to the time-scales involved.
- Well performing single sequence HMMs can serve as the template from which more elaborate models can be developed in order to improve the more difficult tasks. Such models include models for non-coding RNA finding and structure prediction, as well as models that can be used for alignment and incorporating sequence signals into phylogenetic analysis.

Bacterial genomes have relatively simple structure and moderate sizes, with mostly text-book gene structures.

One notable example, the genome of the wild-type K-12 *E. coli* bacteria, is a circular double-stranded piece of DNA consisting of 4,639,675 base-pairs that was first sequenced in 1997 [13]. The current curated GenBank annotation (RefSeq NC000913.2) of the *E. coli* K-12 genome contains 4493 genes. Of these, 155 are non-coding RNA genes (transfer RNAs (tRNAs), ribosomal RNAs (rRNAs) used in translation, as well as various small RNAs with other functions). 4144 are protein coding genes. Despite the fact that this intensely studied model organism has the perhaps best annotated genome of the currently more than 1000 complete prokaryotic genomes, only roughly half (2413) of the protein coding genes have been experimentally verified [14]. The remaining 1731 are potential gene candidates suggested by state-of-the-art prokaryotic gene finding programs based on probabilistic sequence models.

The contiguous protein coding sequences of bacterial genes makes them much easier to predict computationally than the genes of eukaryotes that have large noncoding fractions of their gene sequences (introns).

Significant improvements in bacterial gene finding methodology have made it an almost solved problem. This makes it hard to come up with significant improvements, but provides an ideal benchmark of sequence models that can form the basis of other types of bioinformatic tasks that can potentially use the sequential signal (*e.g.* alignment).

The earliest attempts of computational gene finding were based on k-mer statistics [15, 16]. A significant breakthrough came with the HMM based models [6], that has since been modified in terms of the model types used (*e.g.* Glimmer [17], and Genemark [18]), significance calculations (Easygene [19], lenght modeling (Agene [20]), the incorporation of additional signals such as Ribosome Binding Sites (e.g. prodigal [21]) and the utilization of comparative approaches (*e.g.* twin-scan [22] and N-scan [23]) and combiners (*e.g.* JIGSAW [24]).

Especially the introduction of comparative approaches, that use conserved sequence signals and synteny have lifted the prediction accuracy to almost 100%.

Naturally, this has left most recent developments within gene finding with a focus on eukaryotic gene finding, and with the use of comparative approaches (thanks to the large amounts of whole genome sequences available for a large number of organisms).

However, the availability of a fast prototyping environment for probabilistic sequence models (PRISM) is a good opportunity to explore whether there are alternative models structures that model the sequence component more efficiently.

The application of a combination of logic programming, computational linguistics and stochastic automata on problems in biological sequence analysis is not a recent development (*e.g.* [25, 26, 27, 28]).

The rapid recent growth in probabilistic logic methods has seen numerous applications on problems in expert systems, data-linguistics, robotics, and has also been applied to bioinformatic problems - mostly with a strong emphasis on the computer science.

Notable recent examples of probabilistic logic methods applied on bioinformatics include using constrained hidden Markov models for haplotyping [29] and gene finder performance evaluation through the generation of artificial datasets [30].

The use of probabilistic logic programming for biological sequence analysis is still in its infancy. Paper 1 was the first publication of probabilistic logic programming for biological sequence analysis in a dedicated bioinformatic journal.

As a prototyping environment, probabilistic logic programming brings speed to the development process. The generalized nature of the approach is free from the implementational fine-tunings of the state-of-the-art approaches available for biologists for bioinformatic tasks. This is a strong advantage in terms of prototyping, and providing novel approaches. However, it also means that methods based on probabilistic logic programming is not likely to be consumer ready state-of-the-art products. Nevertheless I sincerely believe that probabilistic logic programming will increase in use in bioinformatics and computational biology, and that proper state-of-the-art material will eventually evolve.

The overall approach for model development described in this thesis, based on probabilistic logic hidden Markov models, is conceptually related to the work in Christiansen *et al.* 2011 [31], but takes an approach more closely related to the spirit of Durbin *et al.* 1998 [3] in providing a model based unified angle to a range of subjects within biological sequence analysis.

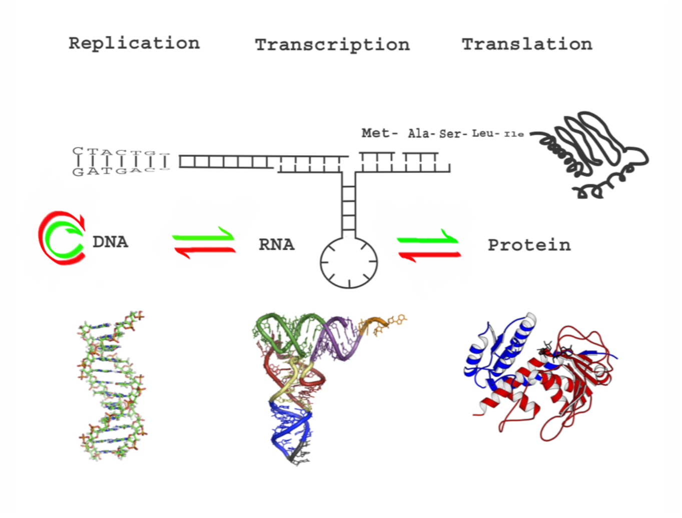
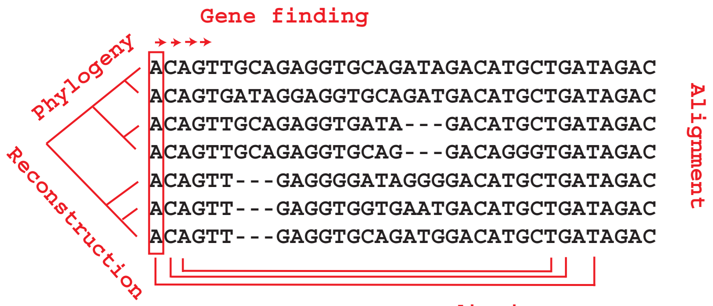


Figure 1: The central dogma of molecular biology relates the sequential information transfers between DNA, RNA and protein (green arrows), and the corresponding constraints that the biological form and functions embedded herein places on the sequential composition of DNA (red arrows).



Gene finding
Phylogeny
Reconstruction
ACAGTTGCAGAGGTGCAGATAGACATGCTGATAGAC
ACAGTGATAGGAGGTGCAGATGACATGCTGATAGAC
ACAGTTGCAGAGGTGATA---GACATGCTGATAGAC
ACAGTTGCAGAGGTGCAG---GACAGGGTGATAGAC
ACAGTT---GAGGGGATAGGGGACATGCTGATAGAC
ACAGTT---GAGGTGGTGAATGACATGCTGATAGAC
ACAGTT---GAGGTGCAGATGGACATGCTGATAGAC
Alignment
RNA structure prediction

Figure 2: An interdependence of the basic biological sequence analysis tasks is dictated by the interactions of overlapping biological form and function that follows from the central dogma of molecular biology. This manifests itself through the constraints imposed on DNA sequences in different dimensions. Naturally, modeling the sequence composition of DNA requires model integration of the different types of constraints imposed by these relationships.

# PRISM

PRISM was introduced in [32] by Taisuke Sato and Yoshitaka Kameya and is publicly available at http://sato-www.cs.titech.ac.jp/prism.

PRISM is particularly well suited for developing models for biological sequence analysis, due to the high expressive power of the PRISM code, the resultant ease of extending and combining model types, the subsumption of model types that have traditionally been used for biological sequence analysis, and the generic nature of the built-in machine learning capabilities.

PRISM is an extension of the logic programming language B-Prolog, augmented with predicates that define random variables, probabilistic inferences and machine learning routines. The built-in routines include predicates for sampling (*sample*), for calculating the joint probability of data and a model (*prob*), for calculating the most probable state of unobserved model variables that comprise an explanation of data and its associated probability (*viterbi*). PRISM also has routines for inferring maximum likelihood or maximum a posteriori estimates of model parameters through an Expectation-Maximization (EM) [33] algorithm or through a Variational Bayes Expectation-Maximization (VB-EM) algorithm [34] (*learn*).

The types of probabilistic models that can be represented as PRISM programs include HMMs, stochastic context free grammars (SCFGs) and Discrete Bayesian Networks [12]. Program execution results in a type of generic dynamic programming matrix [35] (an explanation graph) derived through linear tabling of resolvents of the logic program. The machine learning routines subsequently run on those explanation graphs regardless of which program/model produced them, with the efficiency dictated by the structure of the probabilistic model.

Model selection is facilitated by various statistics of the learning sessions: most important of these are the log likelihood values after learning and the size of the explanation graph. Information scores are also available, including Bayesian Information Criterion (BIC) [36], Cheeseman-Stutz scores [37] and variational free energy scores [34] when using VB-EM.

A thorough introduction to PRISM is beyond the scope of this thesis. The following therefore is a very brief and compact introduction to logic programming, PRISM programs and some of the central PRISM machine learning routines, sufficient for understanding the subsequent models and their application on problems in biological sequence analysis. For a more thorough treatment I refer to Sato *et al.* 2009 [38], the PRISM manual [39], and the references herein.

## Logic Programming

Logic programming is based on methods from automated theorem proving, and use syntax and semantics from formal logic. The underlying principle is that of resolution [40] of Horn clauses [41]. Horn clauses are propositional sentences containing disjunctions of literals with at most one positive literal. Literals are atomic formulas of the form: $p(t_1,\ldots,t_l)$, where $p$ is a predicate symbol and $t$ are terms that can be either logic variables, constants or functions. Functions take the form $f(t_1,\ldots,t_l)$, where $f$ is a functor symbol and $t$ are terms (constants can be thought of as nullary functions).

There are four basic Horn clause constructs:

- Definite clauses (rules) are Horn clauses with exactly one positive literal:

  $\forall(L_0 \vee \neg L_1 \vee \cdots \vee \neg L_m)$

  or equivalently:

  $L_0 \leftarrow L_1 \wedge \cdots \wedge L_m$

- Unit clauses (facts) are definite clauses with no negative literals:

  $\forall(L_0)$

- Goal clauses (potential consequences) have no positive literal:

  $\forall(\neg L_1 \vee \cdots \vee \neg L_m) = \forall(\neg(L_1 \wedge \cdots \wedge L_m))$

- The empty clause ($\emptyset$) contains no literals at all.

Clauses that contain no variables are called ground clauses.

The PRISM model source code directly reflects the underlying structure of the given probabilistic model (see Figure 4). Being able to read PRISM source code will therefore be very helpful for the remaining part of this thesis. The following (general

Prolog) syntax should be sufficient for the programs included here: Comma denotes conjunction and semicolon disjunction. Colon dash represents the reverse implication arrow. Full stop marks the end of a clause. Variable names start with upper case or underscore. Functor symbols are lowercase and atomic formulas are written as $f(term_1, term_2, \dots)$. Underscore denotes the *anonymous variable* that can unify with anything but does not bind to values (*i.e.* multiple occurrences can have different values). Lists are represented as [Head | Tail], where Head is the first item of the list and Tail is a list of the remaining items. Using standard Prolog syntax, the Horn clause constructs are written:

- Rules: $H$ :- $B_1, \dots, B_n$.
- Facts: $H$.
- Goals: $B_1, \dots, B_n$.
- The empty clause: [].

A positive literal is called the head of a clause, the negative literals of a rule is called the body of a rule, and the literals of a goal are called sub-goals.

A logic program $DB$ (database) is a collection of facts $F$ and rules $R$, *i.e.* $DB = F \cup R$.

The canonical logic programming computational procedure is SLD-resolution [42]. SLD-resolution is a sound and refutation complete goal reduction procedure that proceeds by unification of complementary literals of goals and literals of a logic program. The procedure is left-right, top-down recursive with depth first search. Execution is an attempt to refuse a conjecture (the top goal). The basic procedure is as follows:

Given a goal $G = (A_1, A_2, \dots, A_k)$, select the left-most unresolved subgoal $A_1$ of $G$ and derive a new goal $G' = (B_1, \dots, B_n, A_2, \dots, A_k)mgu$ (resolvent) from $G$ using a definite clause "$H \leftarrow B_1, \dots, B_n$" in DB, if there exists a most general unifier $mgu$ such that $(B_1)mgu = (H)mgu$.

A unifier of two terms is a substitution that makes the two terms identical. A substitution is a most general unifier if there does not exist any substitutions that are more general. A substitution $\omega$ is more general than a substitution $\phi$ if there exists another substitution $\rho$ such that $\phi = \omega\rho$.

Procedurally, the anonymous variable unifies with any literal but does not bind the value, uninstantiated variables unifies with any literal, binding the value, instantiated variables unify like constants and constants unify if they are identical. The goal reduction procedure produces a derivation tree (AND/OR graph) with the initial top goal as root node, resolvent goals as child nodes and edges labeled with the *mgu* substitutions. If the selected literal unifies with a fact (comprised of a Head only), the derived goal is the empty clause. If there are no Heads that the selected goal can unify with, the leaf is a failure node. If the derivation contains only the empty clause (or a conjunction of empty clauses) the leaf is a success node. A path from the root node to a leaf node is called a derivation. If the derivation tree has the empty clause as a leaf, the goal is a consequence of the facts and rules of the logic program (nothing remains to be resolved - the conjecture is refuted). If the goal contains logic variables, potential variable bindings are returned via the *mgu*. The search is completed by backtracking upon failure and continuing with the remaining unresolved subgoals.

Figure 3 gives an example of a small logic program and the resulting derivation trees obtained from the resolution of a ground and an unground goal.

p(a).
p(b):-p(a).

(a)

$$\begin{tikzcd}
p(b) \arrow[d] \\
p(a) \arrow[d] \\
[\,]
\end{tikzcd}$$

(b)

1: the goal ”p(b)” does not unify with ”p(a).”
2: ”p(b)” unifies with the head of ”p(b):-p(a).”
3: new resolvent goal ”p(a)”
4: ”p(a)” unifies with ”p(a).”
5: new resolvent goal ”[].” (success)

(c)

$$\begin{tikzcd}
 & p(X) \arrow[dl, "X=a"'] \arrow[dr, "X=b"] & \\
p(a) \arrow[d] & & p(b) \arrow[d] \\
[\,] & & p(a) \arrow[d] \\
 & & [\,]
\end{tikzcd}$$

(d)

1: the goal ”p(X)” unifies with ”p(a).”
2: *mgu*(X=a).
3: new resolvent ”[].” (success)
4: solution: ”X=a.”
5: more solutions?
6: backtrack to previous goal, *i.e.* ”p(X)”
7: the goal ”p(X)” also unifies with the head of ”p(b):-p(a).”
8: *mgu*(X=b).
9: the goal ”p(b)” does not unify with ”p(a).”
10: ”p(b)” unifies with the head of ”p(b):-p(a).”
11: new resolvent goal ”p(a)”
12: ”p(a)” unifies with ”p(a).”
13: new resolvent goal ”[].” (success)
14: solution: ”X=b.”

(e)

Figure 3: (a) A small logic program (b) SLD-tree from the resolution of the ground goal ”p(b)”. (c) Procedure of the SLD-resolution of the ground goal ”p(b)”. (d) SLD-tree from the resolution of the unground goal ”p(X)”. (e) Procedure of the SLD-resolution of the unground goal ”p(X)”.

## PRISM Programs

PRISM programs define probabilistic models. The probabilistic models are constructed as logic programs with predicates that represent random variables. The facts and rules of the logic program determine the model structure, *i.e.* how the random variables are related to each other and to observable data. For the sequential models, the basic model structure is a recurrence relation. Additionally, the program must contain rules for initiating the recurrence and facts for terminating the recurrence. Optionally, the program can contain auxiliary predicates for more elaborate sub-model structures, and predicates for handling system behavior, *e.g.* options for the build-in routines. PRISM programs are generative models, that specify a joint probability over observed data and variable states in the model structure, and can be used both to parse a given observation and to sample from.

In *sample mode*, given an unground top-goal (observation) PRISM returns a sample generated from the model. In *explanation mode* PRISM can be used with *prob*,*viterbi* or *learn*. It is possible to use PRISM like regular Prolog with a ground top goal. However, for a ground top goal to succeed it requires a sample-parse through the model that generates that goal exactly. For most models the potential outcome space is enormous and the probability of success is very small - resulting in most cases of the system returning a ”no”, even if the top goal is derivable from the program in a deterministic sense.

Random variables in PRISM are declared using the *values/3* predicate. The first argument to *values* is the name of the random variable, the second argument is a list of outcomes that define the outcome space of the random variable, and the (optional) third argument is a corresponding list of point probabilities - the parameters of the random variable. (The third argument can be omitted leaving the random variable uniformly distributed). Random variables are invoked via including *msw/2* predicates in the (recurrence) relations that define a given model. The first argument of *msw(i,v)* is the name of the random variable defined via a *values* predicate, the second argument is a value (outcome) of the random variable.

## PRISM Resolution

The current version of PRISM use linear tabled SLD-resolution to enable dynamic programming (and handle left-recursion) with recursive programs [43]. The first implementations of PRISM used OLDT resolution (Ordered Linear Definite Clause Resolution with Tabling)[44] to construct the generic dynamic programming matrices for probabilistic logic programs.

The procedure starts from a ground top goal (an observation) and produces a derivation tree structure called an explanation graph. Dynamic programming behavior is achieved trough tabling subgoals and reusing their sub-solutions.

When a program encounters a subgoal that has already been resolved and is hence stored in the solution table, resolution of that goal is put on hold and previous solutions from the solution table are retrieved and applied first. Sub-goals of the sub-solutions are also tabled, resulting in a general solution table containing subgoals as keys and sub-solutions as entries.

The explanation graph is constructed from the solution table as ordered factorized iff formulas:

subgoal/tabled atom $\Leftrightarrow$ sub-exlanation, *i.e.*:

$G_i \Leftrightarrow msw_1 \wedge \cdots \wedge msw_m \wedge G'_1 \wedge \ldots G'_n$

where a sub-explanation is a recursive structure consisting of a conjunction of sub-goals and $msw/2$ predicates.

An explanation for a given observation is a conjunction of switch instances corresponding to a path in the derivation tree, from the root node to a leaf node. The probability of an explanation is the product of the switch instance probabilities in it. The probability of a goal is the sum over the switch instances of all the of its explanations.

## PRISM Routines

There are three basic routines running on explanation graphs: *prob*, *viterbi*, and *learn*.

*prob* calculates the marginal probability $P_\theta(G)$ of an observed goal $G$ given a model parameterized by $\theta$. This is achieved by traversing the explanation graph and summing over disjunctions of *msw/2* instances and multiplying over disjunctions. *prob* corresponds to the Forward algorithm for HMMs [45], the Inside algorithm for SCFGs [46] and the Belief Propagation algorithm [47] for Bayesian Networks.

*viterbi* finds the most probable explanation $E^* = \arg\max_{E \in \{E_1,\ldots,E_K\}} P_\theta(E)$ via traversing the explanation graph and composing the most probable explanation, using the most probable sub-explanation at each step. *viterbi* is a generalization of the Viterbi algorithm [48] for HMMs.

*learn* obtains the set of parameters $\theta$ that maximize the likelihood $\Pi_t P_\theta(G_t)$, given a collection of observed goals $\{G_1,\ldots,G_T\}$. The maximum likelihood parameter estimates are obtained using an Expectation-Maximization algorithm.

For models with latent variables, the expected values of the parameters (their frequencies) that optimize the Maximum Likelihood cannot be obtained simply by counting, since the states of the latent variables are unknown. Instead Maximum likelihood estimates of the parameter values can be obtained by approximating the true values of the parameters by incrementally changing the parameters in ways that optimizes the likelihood function. The Expectation-Maximization algorithm is one such method, that proceeds via successive iterations of the following two steps (starting from randomly assigned values of the parameters):

- E(xpectation) step: Compute the expected (most probable) values of the latent variables given the current parameter values and observable variable values.
- M(aximization) step: Update the parameter values by choosing the maximum likelihood estimate of the parameters given the the most probable values of the latent variables and the observable variables found in the preceding step.

Due to the concavity of the logarithmic function and Jensen's inequality [49], the likelihood value for each iteration cannot be lower than the previous likelihood value.

Hence the log-likelihood is guaranteed to converge towards a (local) maximum [33].

The PRISM EM-procedure [50] starts by initializing the parameters with random values $\hat{\theta}^0$, then iterating until convergence $(L(G_1,\dots,G_T|\hat{\theta}^{n+1}) - L(G_1,\dots,G_T|\hat{\theta}^n) \leq \tau$, (where $\tau$ is some preset threshold value):

- E-step: count the (expected) occurrences $\hat{C}_{i,v}$ of $msw(i,v)$ under $\hat{\theta}^n$
- M-step: update each parameter as $\hat{\theta}^{n+1}_{i,v} = \frac{\hat{C}_{i,v}}{\Sigma_{v'}\hat{C}_{i,v'}}$

Maximum A Posteriori estimated parameters $\hat{\theta}^{MAP}$, that maximize the posterior probability of the parameters given the data: $P(\theta|G_1,\dots,G_T) \propto P(\theta)\Pi_t P_\theta(G_t)$, are obtained by assigning a Dirichlet prior $P(\theta) = \frac{1}{Z}\Pi_{i,v}\theta_{i,v}^{\vartheta_{i,v}-1}$, where $Z$ is a normalizing constant and $\vartheta_{i,v}$ are hyper-parameters of the Dirichlet distribution of the corresponding *msw(i,v)*, and using the following M-step:

- M-step: estimate each parameter as $\hat{\theta}_{i,v} = \frac{\hat{C}_{i,v}+\sigma_{i,v}}{\Sigma_{v'}(\hat{C}_{i,v'}+\sigma_{i,v'})}$

where $\sigma_{i,v} = (\vartheta_{i,v} - 1)$ are pseudo-counts for each *msw(i,v)*.

Alternatively, Variational Bayes estimators of the hyper-parameters $\hat{\vartheta}_{i,v}$ are obtained via initializing the hyper-parameters as $\vartheta^0_{i,v} = \vartheta_{i,v} + \nu_{i,v}$, where $\nu_{i,v}$ are small positive random (gaussian) noises, and then iterating:

- E-step: for each $msw(i,v)$ count $\tilde{C}_{i,v}$, the expected occurrences of *msw(i,v)* under the hyper-parameters $\hat{\vartheta}^n_{i,v}$
- M-step: update each hyper-parameter as $\hat{\vartheta}^{n+1}_{i,v} = \hat{\vartheta}^n + \tilde{C}_{i,v}$

## Model Selection

Being able to express these different model types in a single framework enables direct comparisons of different underlying model structures in terms of their statistical characteristics, computational complexity, and performance on given tasks, without complications due to implementation differences. Such model exploration is an ideal platform for discovering sequence models that not only lead to a better understanding of the forces shaping biological sequences, but also lead to general ways of improving tasks that rely on sequence signals. In order to choose the best model a number of criteria can be evaluated for model selection based on statistical measures, computational complexity and (classification) performance.

### Statistical Measures

In addition to the (log) likelihood values after learning, PRISM provides three scores that can be used for model selection. Generally, the task is to find the most probable model $M^*$:

$$\begin{aligned} M^* &= \arg\max_M P(M|G_1,\dots,G_T) \\ &= \arg\max_M \frac{P(G_1,\dots,G_T|M)P(M)}{P(G_1,\dots,G_T)} \\ &= \arg\max_M P(G_1,\dots,G_T|M), \end{aligned} \tag{1}$$

*i.e.* the one that maximize the (log) marginal likelihood $P(G_1,\dots,G_T|M)$ assuming a uniform prior on $P(M)$, where $P(M|G_1,\dots,G_T)$ is the posterior probability of the model, and $P(G_1,\dots,G_T)$ is the marginal probability of the data (that does not depend on the model). $P(G_1,\dots,G_T|M)$ can be obtained as the expectation of the maximum a posteriori (log) marginal likelihood: $P(G_1,\dots,G_T|M) \equiv \langle P(G_1,\dots,G_T|\hat{\theta}^{MAP},M)\rangle$ [39].

PRISM provides the following three approximations to $P(M|G_1,\dots,G_T)$:

BIC:

$$Score_{BIC}(M) \overset{\text{def}}{=} P(G_1,\dots,G_T|\hat{\theta}^{MAP},M) - \frac{|\theta|}{2}\log N \tag{2}$$

where $|\theta|$ is the number of free parameters and $N$ is the sample size.

Cheeseman-Stutz score:

$$\begin{aligned} &Score_{CS}(M) \overset{\text{def}}{=} \\ &P(\tilde{D}_c|M) - P(\tilde{D}_c|\hat{\theta}^{MAP},M) + P(G_1,\dots,G_T|\hat{\theta}_{MAP},M) \end{aligned} \tag{3}$$

where $\tilde{D}_c$ is the pseudo-complete data whose sufficient statistics are the expected occurrences of the stochastic variables in the model.

Variational Free Energy score:

$$Score_{VFE}(M) \overset{\text{def}}{=} \sum_{E} \int_{\Theta} q(E,\theta|G_1,\ldots,G_T,M) \log \frac{p(E,G_1,\ldots,G_T,\theta|M)}{q(E,\theta|G_1,\ldots,G_T,M)} d\theta \qquad (4)$$

where $q$ is the (approximated) posterior distribution function that maximize the VFE, and $p$ is the joint distribution function.

## Computational Complexity

The complexity of the PRISM algorithms running on the explanation graphs is linear in the size of the explanation graph [39]. The size of the explanation graph in turn is dictated by the model structure and the number of parameters. Since the computational complexity generally is data dependent and usually is given for models in terms of a worst case upper bound, the possibility to compare the complexity performance of different models in PRISM on the same data is very useful. Generally, PRISM models behave as expected in terms of their computational complexity ($O$). Ordinary HMMs have $O(nm^2)$, where $m$ is the length of the observed sequence and $n$ is the number of hidden states. Mixed memory (and higher ordered) HMMs have $O(n^p m^2)$, where $p$ is the size of the memory and $n^p$ is the number of states. SCFG like models have $O(nm^3)$ complexity. Factorial HMMs have $O(m^n)$ complexity. Intractable models such as factorial HMMs or factorial HMM/SCFGs can be coped with by keeping the number of parameters low (*i.e.* using well-fitting sub-models) and/or applying the models on reduced amounts of data (*e.g.* prokaryotic genes, viral genomes, potential open reading frames, genome fragments etc.). For SCFG-like models tractable models can be obtained via limiting the size of the stack or more generally the number of successive identical states (see Paper III). Length modeling can then be used to keep the probability mass contained within that limit (at the cost of slightly increasing time complexity). Alternatively, complicated models can be approximated by simpler ones [51, 52], using constraint models [53], viterbi with annotations [54] or using heuristic approaches to *viterbi* or (VB-)EM.

### Classification Performance

HMMs can be used as classifiers of (sub)sequences. This can be achieved either by using the Viterbi path or the maximum posterior probability decoding path through a sequence to denote membership of a subsequence. An alternative possibility is to discriminate using log odds ratios of subsequences calculated from the probabilities obtained with a given model and those obtained with a suitable null model. The candidate set then consists of subsequences with log-odds scores exceeding a given threshold, that can be chosen to optimize the classification performance with respect to a golden standard. The classification performance can be evaluated using a confusion matrix consisting of true positives, false positives, true negatives and false negatives. For gene finding, annotation in terms of correctly predicted residues, correctly predicted start codons or correctly predicted stop codons are the typical options. Performance measures based on accurate prediction of stop codon position are the most interesting, since discovery of previously unknown genes is more interesting than correctly annotating the starting position of an already known gene. Due to the tradeoff between sensitivity and specificity, a useful tool to evaluate how models compare is a receiver operator characteristics (ROC) curve [55]. ROC curves are a plot of sensitivity vs true positive rate over all discrimination threshold values. The procedure for gene finding used in Paper I and Paper II have been to divide all potential ORFs sequences into a positive set (golden standard) and a negative set. Confusion matrices and prediction performance metrics are then calculated for all observed log odds values of the positive set as thresholds. If only a single confusion matrix is to be reported, a suitable choice is the one that maximizes the difference between true positive rate and false positive rate, corresponding to the intercept of the ROC curve with the parallel to the no-discrimination line (the ascending diagonal). The area under the curve (AUC) value [56] of the ROC curve is a useful summary of the classification performance over all thresholds for general comparisons of model prediction performance.

## Hidden Markov Models in PRISM

HMMs are generative probabilistic sequence models. HMMs are characterized by a set of transition probabilities for parsing between unobserved states and a set of emissions probabilities of emitting characters from those unobserved/hidden states. The generation of the state and observed sequence is a Markov process in the sense that the probability of being in a certain state is independent from the entire sequence up to that point.

HMMs were originally introduced by Baum and Petrie in 1966 [57], and have been widely applied [58]. A number of specific types of HMMs have been developed within biological sequence analysis that express the special functional constraints acting on biological molecules. These models include profile HMMs[59], pair HMMs [3] and more complicated models such as phylo HMMs[60, 61], coalsecent HMMs[62, 63], input-output HMMs[64] and recombination HMMs[65]. Within the past decades HMMs have been used for incorporating sequence signals for gene finding [6], sequence alignment [3], RNA structure prediction [66] protein structure prediction [67] and phylogenetic/population genetic inferences [68, 62].

The success of HMMs is due to efficient algorithms that can be used for machine learning tasks given a dataset of sequences. These can be used to infer the most probable parameters of the model given the data (the Baum-Welch algorithm [69]) and inferring the most probable annotation of the data given a parameterized model (the Viterbi algorithm [48]) [58]. The specific HMM machine learning algorithms are subsumed by the corresponding PRISM algorithms already described.

HMMs can be interpreted as instantiations of probabilistic graphical models (*i.e.* as dynamic bayesian networks [70]) as stochastic transformational grammars [71] (*i.e.* as stochastic regular grammars), and as stochastic automata [72] (*i.e.* as stochastic Moore machines [73] - a type of probabilistic finite state automatons).

An HMM is formally defined as a tuple $(S, A, T, E)$ containing a set of states $S = s_1, s_2, \ldots, s_m$, an alphabet of emitted characters $A = a_1, a_2, \ldots, a_t$, a set of transition probabilities between states $T = \{t_{kl}\}, k, l = 1, \ldots, m$ and a set of emission probabilities of the characters in the alphabet $E = \{e_k(a_j)\}, k = 1, \ldots, m, j = 1, \ldots, t$. (*i.e.* the parameters $\theta = \{T, E\}$). States that do not emit characters are termed silent states, *e.g.* the begin state $\beta$ and the end state $\varepsilon$ used for modeling the initialization and termination of a sequence. Given a sequence $X = x_1, x_2, \ldots, x_n$ of characters in $A$ and a path $\Pi = \pi_1, \pi_2, \ldots, \pi_n$ of states in $S$, the transition probabilities $t_{kl}$ are defined for position $i$ and states $k, l$ in $S$, as:

$$t_{kl} = \mathbf{P}(\pi_i = l \mid \pi_{i-1} = k) \tag{5}$$

the emission probabilities $e_k(a)$ are defined for position $i$, state $k$ in $S$ and character $a$ in $A$, as:

$$e_k(a) = \mathbf{P}(x_i = a \mid \pi_i = k) \tag{6}$$

and the joint probability of observing the sequence $X$ and the path $\Pi$ under the parameters $\theta$ is:

$$\mathbf{P}_\theta(X, \Pi) = t_{\beta\pi_1} \prod_{i=1}^{n} e_{\pi_i}(x_i) t_{\pi_i\pi_{i+1}} \tag{7}$$

where $\pi_{n+1} = \varepsilon$.

PRISM HMMs consists of a collection of *values/2* predicates that defines random variables and their outcome spaces, and recursive structures that define the relations of the observable data and the parameters.

An observable sequence is given as a list and the recursive structure corresponds to the factorization scheme of equation (7).

The basic PRISM model consists of:

- *values* predicates that define the random variables and their outcome spaces
- an initiation rule that defines the predicate (*model*) that relates the observable data and the central recurrence relation.
- A *recursion* clause that defines how the random variables are related to each other and to the observable data at each recurrence step, defining the structure of the model.
- A termination clause (typically using a *recursion* fact) defines the conditions for stopping the recursion and completing the model.

Figure 4 gives the (fully executable) source code example of a simple two-state HMM that can parse and generate DNA sequences.

```
% Transition Probabilities (eqn. 5):
values(transition(_),[state(1),state(2),end]).

% Emission Probabilities (eqn. 6):
values(emission(_),[a,c,g,t]).

% Initiation:
model(Observables):-
  msw(transition(begin),Next_state),
  recursion(Next_state,Observables).

% Recursion (eqn. 7):
recusion(state(S),[Symbol | Rest]):-
  msw(emission(S),Symbol),
  msw(transition(S),Next_state),
  recursion(Next_state,Rest).

% Termination:
recusion(end,[]).
```

Figure 4: PRISM source code of a two-state fully connected standard HMM. Note the close correspondence between the PRISM source code, the structure of equations 5-7 and the graphical model depicted in Figure 5.

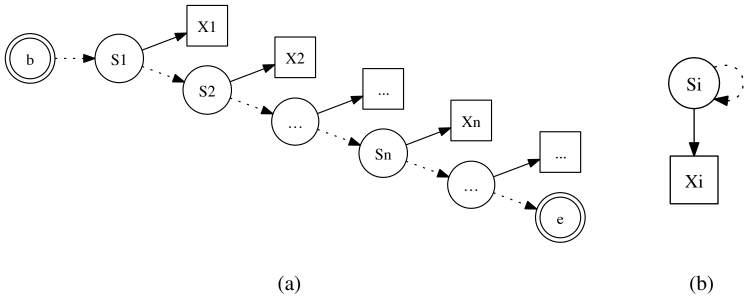


Figure 5: Graphical presentations of HMMs. Square nodes are the hidden state nodes and the circled nodes are the emitted symbols. Dotted arrows are the transition probabilities and fully drawn arrows are emission probabilities. (a) Finite State Machine depiction of standard HMM showing the sequence of states and sequence of emitted symbols. (b) Graphical Model depiction of standard HMM with a hidden state variable Si and a emission variable Xi, begin and end states not shown.

# PRISM HMM Variants for Biological Sequence Analysis

The following section describes a number of variants and extensions of standard single sequence HMMs that can be used directly for modeling biological sequences, and/or illustrate principles of sequence modeling for biological sequence analysis. Model structures that have been developed for one type of models can readily be transferred to another type of model.

For more elaborate details and examples on how the models are used for biological sequence analysis tasks I refer to Papers I-III.

## Markov Chains

By omitting the hidden state sequence and conditioning the emissions on the previous emission we obtain an ordinary (first order) Markov chain [74], that is given by:

$$\mathbf{P}_\theta(X) = \prod_{i=1}^{n} \mathbf{P}(x_i = a_i \mid x_{i-1} = a_{i-1}) \tag{8}$$

Even simpler, a zeroth order Markov chain that corresponds to an independently and identically distributed (IID) sequence model is given by:

$$\mathbf{P}_\theta(X) = \prod_{i=1}^{n} \mathbf{P}(x_i = a_i) \tag{9}$$

whereas a $d$-ordered inhomogeneous Markov chain is given by (adding $d$ extra ”dependencies”):

$$\mathbf{P}_\theta(X) = \prod_{i=1}^{n} \mathbf{P}(x_i = a_i \mid x_{i-1} = a_{i-1}, \ldots, x_{i-d} = a_{i-d}) \tag{10}$$

With the introduction of the hidden states/latent variables, the notion of higher order becomes somewhat less well defined. For a first order HMM, the emissions and transitions only depend on the present state:

$$\mathbf{P}_\theta(X,\Pi) = \mathbf{P}(\pi_1 = k_1 \mid \beta)\prod_{i=1}^{n}\mathbf{P}(x_i = a_i \mid \pi_i = k_i)\mathbf{P}(\pi_{i+1} = k_{i+1} \mid \pi_i = k_i) \tag{11}$$

(corresponding to the HMM example in figure 1.4.). For a single state zeroth order HMM, the emissions do not depend on the state, and the HMM effectively becomes a zeroth order Markov chain:

$$\mathbf{P}_\theta(X,\Pi) = \mathbf{P}(\pi_1 = k_1 \mid \beta)\prod_{i=1}^{n}\mathbf{P}(x_i = a_i)\mathbf{P}(\pi_{i+1} = k_{i+1} \mid \pi_i = k_i) = \prod_{i=1}^{n}\mathbf{P}(x_i = a_i) \tag{12}$$

The primary use of simple Markov chain like models in the work described here have been as null models for calculating log odds ratios for the models used for capturing protein coding potential in Paper I.

## Mixed Memory HMMs

In mixed memory HMMs both emissions and transitions can be conditioned on previous states or previous emissions [75]. For a second order HMM there are the following four possibilities (that adds an extra term from the immediate vicinity of the present state/emission): Emissions conditioned on present state and previous emission:

$$\mathbf{P}_\theta(X,\Pi) = \mathbf{P}(\pi_1 = k_1 \mid \beta)\prod_{i=1}^{n}\mathbf{P}(x_i = a_i \mid \pi_i = k_i, x_{i-1} = a_{i-1})\mathbf{P}(\pi_{i+1} = k_{i+1} \mid \pi_i = k_i) \tag{13}$$

Emissions conditioned on present state and previous state:

$$\mathbf{P}_\theta(X,\Pi) = \mathbf{P}(\pi_1 = k_1 \mid \beta)\prod_{i=1}^{n}\mathbf{P}(x_i = a_i \mid \pi_i = k_i, \pi_{i-1} = k_{i-1})\mathbf{P}(\pi_{i+1} = k_{i+1} \mid \pi_i = k_i) \tag{14}$$

Transitions conditioned on present state and previous state:

$$\mathbf{P}_\theta(X,\Pi) = \mathbf{P}(\pi_1 = k_1 \mid \beta)\prod_{i=1}^{n}\mathbf{P}(x_i = a_i \mid \pi_i = k_i)\mathbf{P}(\pi_{i+1} = k_{i+1} \mid \pi_i = k_i, \pi_{i-1} = k_{i-1}) \tag{15}$$

And lastly, transitions conditioned on present state and present emission:

$$\mathbf{P}_\theta(X,\Pi) = \mathbf{P}(\pi_1 = k_1 \mid \beta)\prod_{i=1}^{n}\mathbf{P}(x_i = a_i \mid \pi_i = k_i)\mathbf{P}(\pi_{i+1} = k_{i+1} \mid \pi_i = k_i, x_i = a_i) \tag{16}$$

Obviously, a large number of possible different conditioning schemes are available with even modest numbers of extra terms. Figure 6a gives a schematic presentation of a mixed memory HMM where the transitions are conditioned on the previous two states and the emissions are conditioned on the present state, the previous two states and the previous two emissions. Higher ordered HMMs with emissions conditioned on (a fixed number of) previous emissions have been used as the underlying probabilistic model for the Genemark gene finder [76]. The large set of different encodings of dependencies in the data available from specific combinations of the different conditioning schemes offers alternative ways to capture the sequence characteristics (*e.g.* codon usage and the presence of start and stop codons with the same numbers of free parameters). A number of such model structures and how they compare in terms of classification performance on prokaryotic gene finding is treated in Paper I.

The general principle of mixed memory-ness can easily be extended to more complicated models such as SCFGs or SCFG-like HMMs or any of the other examples of HMM based models presented here (*e.g.* see the mm_scfg_hmm.psm example in the Appendix).

## Multi Sequence HMMs

Ordinary first order HMMs with a single state chain and a single emission chain can be extended to a single state chain and two emission chains, *i.e.*:

$$\mathbf{P}_\theta(X,Y,\Pi) = \mathbf{P}(\pi_1 = k_1 \mid \beta)\prod_{i=1}^{n}\mathbf{P}(x_i = a_i, y_i = b_i \mid \pi_i = k_i)\mathbf{P}(\pi_{i+1} = k_{i+1} \mid \pi_i = k_i) \tag{17}$$

The two emission chains can be composed from the same alphabet or from different alphabets. This can be achieved either as two independent emissions from a given state or as emissions of two-tuples from a given state. Figure 6b gives a schematic presentation of a two-sequence double HMM. Two sequence HMMs can be used for supervised training by emitting two-tuples containing the ordinary emitted symbols and the hidden state, with each member of the tuples going into separate sequences. The first sequence is a regular emitted sequence and the other sequence corresponds to an annotation of the first sequence. Since the parameter values (the emission and transition probabilities) of a single and a two- sequence HMMs are identical, the values can be exported from a two-sequence annotation model trained on fully annotated data and imported into a single sequence model for decoding. This principle is used in Paper III for training two-sequence SCFG-like HMMs used for RNA secondary structure modeling. Three-sequence HMMs can be produced straightforward using the same principles as for the two-sequence HMMs.

The use of two-sequence HMMs for supervised learning can easily be extended to multi-sequence HMMs used for supervised learning with multiple annotations.

## Indel HMMs

Pair HMMs are two-sequence HMMs with a match state $m$ that emits to both sequences 1 and 2, and indel states $x$ (that only emits to sequence 1) and $y$ (that only emits to sequence 2), either with the same or with different emission probabilities [3].

Pair HMMs can be used for pairwise alignment [3]. A schematic presentation of a pair HMM is given in Figure 6e, and an example of the PRISM code of a pair HMM is given in the Appendix (pair_hmm.psm).

A possible extension of pair HMMs are triple HMMs *i.e.* three-sequence HMMs with indel states (see triple_hmm.psm example in the Appendix).

Potential applications for triple HMMs includes examining iterative refinement based multiple alignment with fewer (potentially better) steps than iterative refinement based multiple alignment using pairwise alignments (More generally, a comparison of pair HMM and triple HMM performance on iterative multiple alignment could be used to investigate the trade-off between the number of iterations and the complexity of each iteration common to all iterative approaches).

Another extension is to combine codon models and/or length modeling sub-models into pair HMMs for generalized pair HMMs [77] that are specifically tailored for aligning protein coding regions or more elaborate gene structure compositions (*e.g.* pair_codon_hmm.psm in the Appendix).

The principles of constructing insertion and delete states can also be used for SCFGs or SCFG-like HMMs for aligning RNA structures (*e.g.* pair_scfg.psm in the Appendix). Additionally, insertion and delete states can be used for profile HMMs [59] and for modeling sequencing errors in gene finders.

In Paper II we use a simple HMM model with a delete state to model the reading frame sequence of genes in a genome where the delete state is used to filter a set of predictions from another gene finder. Figure 6d depicts an HMM with a single delete state.

## HMMs with Acyclic Directed Phase type Length Modeling

Length modeling is a significant contribution to gene finding, to indel and match lengths in alignments and for RNA secondary structure components. Very effective length modeling can be achieved with Acyclic Discrete Phase type models [78] (*e.g.* used for gene finders Easygene [19] and Agene [20]. ADHP modeling can be achieved through breaking a single state up into a number (typically three) states that share transition probabilities. Generic ADPH code can be included in all of the above mentioned models. Figure 6f depicts the structure of an ADPH HMM. An example of the effect of length modeling on gene finder model structure performance can be found in Paper I.

The principles of ADPH length modeling can be transferred to other HMM based models or incorporated into sub-model structures (*e.g.* adph_scfg_hmm.psm in the Appendix)

## SCFG-like HMMs

SCFGs are transformational grammars that use production rules for generating sequences that contains palindromic subsequences [71]. SCFGs have been used to model the palindromic sequences arising due to the long distance base-pairing interactions in stem regions that are part of RNA secondary structures [79, 80, 81] (see scfg.psm in the Appendix for an example of an RNA SCFG model).

SCFG-like HMMs can be obtained by changing one of the lists of a two-sequence HMM into a stack (a last in - first out data structure). Emissions can be from states, from the stack or by emptying the stack onto the emitted sequence. A full SCFG model is obtained if both states and emissions can be added to the stack. However, all canonical RNA secondary structures can be modeled by SCFG-like HMMs where only right hand sides of emission tuples are added to the stack, given the right production rules. This produces a much simpler model that is not as computationally demanding as SCFGs. Paper III explores the use of SCFG-like HMMs for modeling RNA secondary structures.

## Factorial HMMs

Factorial HMMs are inverse two-sequence HMMs. They are composed of a single emission sequence $X = x_1, x_2, \ldots, x_n$ and two hidden state chains with paths $\Pi = \pi_1, \pi_2, \ldots, \pi_n$ and $\Gamma = \gamma_1, \gamma_2, \ldots, \gamma_n$ given by:

$$\begin{aligned}
&\mathbf{P}_\theta(X, \Pi, \Gamma) = \qquad (18)\\
&\mathbf{P}(\pi_1 = k_1, \gamma_1 = l_1 \mid \beta)\\
&\prod_{i=1}^{n} \mathbf{P}(x_i = a_i \mid \pi_i = k_i, \gamma_i = l_i)\mathbf{P}(\pi_{i+1} = k_{i+1} \mid \pi_i = k_i)\mathbf{P}(\gamma_{i+1} = l_{i+1} \mid \gamma_i = l_i)
\end{aligned}$$

Factorial HMMs were introduced in [82] and can be used for modeling overlapping features. Figure 6c shows the basic structure of a factorial HMM. (For a general example see factorial_hmm.psm, for an example of a factorial model of overlapping reading frames see mm_frames.psm, both in the Appendix). Extensions of factorial HMMs to factorial SCFGs or SCFG-like HMMs can potentially be used for modeling overlapping RNA secondary structures (*e.g.* riboswitches, see factorial_scfg_hmm.psm in the Appendix).

Factorial SCFG/SCFG-like codon HMM models with a codon HMM structure in one state chain and a SCFG/SCFG-like HMM in the other chain are natural models of overlapping RNA structures and protein coding sequences. For SCFG/codon HMM models, coordination of the two model types is achieved by the HMM chain ”waits” when the SCFG/SCFG-like HMM is in non-terminal states and only emits when the SCFG/SCFG-like HMM is in a terminal state (*e.g.* scfg_mrna.psm in the Appendix). In SCFG-like HMM/codon HMM models, coordination of the two model types follows from the ”emissions” from nonterminals to the stack and emissions back again from the stack to the observed sequence. A factorial SCFG-like HMM/HMM as a model of mRNAs is treated in Paper III.

## Repeat HMMs

Repeat HMMs are HMMs with two lists. The first list is a stack like in the SCFG-like HMMs. The second list is a queue that the stack empties into. Emptying the queue results in a repeated sequence. This model is computationally very demanding but this can be overcome by limiting the size of the stack (*e.g* the repeat_hmm.psm example in the Appendix)

The approach is related to repeat parsing with probabilistic regular expressions [83]. With the HMM formulation one should easily be able to incorporate the ADPH length modeling, mixed memory transitions and emissions, multi-sequence models or other model schemes described above.

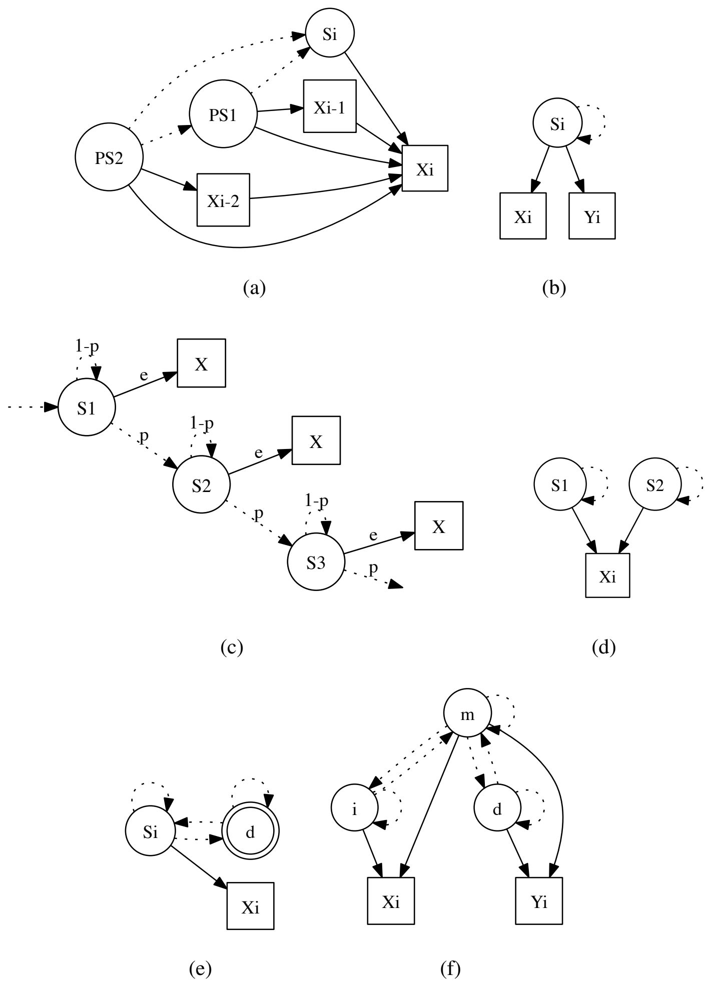


Figure 6: Examples of basic HMM variants. (a) Mixed memory HMM. (b) Double HMM. (c) ADPH HMM. (d) Factorial HMM. (e) Delete HMM. (f) Pair HMM.

# Paper Summaries

In this last section of the introduction I briefly summarize the included papers.

## Paper I

Hidden Markov models (HMMs) are classic models of protein coding potential for use in gene finding. Single sequence HMMs can be constructed in a large number of different structures determined by different conditioning schemes. However, surprisingly few fundamentally different HMM structures have so far been used for modeling protein coding potential.

In Paper I, we develop and test different HMM structures for modeling protein coding potential. The models include the two most common model structures that capture protein coding potential: a structure with 5th ordered emissions and one with inhomogeneous 3-periodic Markov chain like emissions, as well as two new types of structures consisting of a model based on a amino acid hidden state sequence and model based on a mixed memory HMM. Also included is a simple IID Markov chain-like model used as null model. All models have been constructed in single state variants, three state variants both with and without ADPH length modeling.

The new model structures perform better than the previously used structures in terms of both statistical information criteria and prediction performance, suggesting that they could serve as potential improvements for other HMM based models that incorporates protein coding potential.

## Paper II

Gene finding is typically based on probabilistic sequence models at the *intra*-gene level.

In paper II, we introduce a new class of probabilistic models of a genome level sequence signal that is interesting from a probabilistic modeling perspective, a genome organization perspective and for the gene finder development community. We use a probabilistic logic programming framework to develop and test our models, that adds much needed speed and flexibility in discovering novel types of probabilistic models for biological sequence analysis. The present work connects to previous uses of probabilistic sequence models within biological sequence analysis - especially the use of Hidden Markov Models used in contemporary gene finders for modeling the sequence constraints acting on the nucleotide composition of genes.

In the paper we present a probabilistic sequence model at the *inter*-gene level. The model consists of a simple single sequence HMM model with a delete state. The model parses the reading frame of a gene and the score attributed by a gene finder. The model produces a parse through gene candidates corresponding to the gene finder score and reading frame sequence.

The model was tested on *E. coli* predictions from state-of-the-art gene finders Genemark, Glimmer and Prodigal, trained on a number of increasingly distant prokaryotic genomes. The gene sequence signal appears to be remarkably widespread, with improved performance over Genemark, Glimmer and Prodigal using models trained on *Escherichia, Salmonella, Legionella, and Bacillus* genomes. Performance was not improved with a model trained on an Archean genome (*Thermoplasma*), indicating the potential limits of the conservation of the gene-sequence signal.

In conclusion, our approach demonstrates that delete-HMM based models is a valuable method for filtering gene finder candidates into a coherent set with respect to the sequential composition of reading frames of the genes in a genome.

## Paper III

RNA sequences are hard to handle due to the long-distance interactions of the base pairing of stem regions. Current method for RNA secondary sequence structure is either based on Minimum Free Energy approaches or on Stochastic Context Free Grammars.

In Paper III we introduce probabilistic logic programming as a platform for developing, testing and expanding models for RNA sequence analysis. We introduce and compare HMM based RNA models with SCFG based RNA models in terms of complexity, learn statistics and prediction performance using cross-validation of a small dataset of short RNA sequences. We find that RNA HMMs are equivalent to the

RNA SCFG models in terms of prediction performance, have better complexity for learning, much worse complexity for decoding and that both RNA HMM and RNA SCFG models can be improved beyond the prediction performance of RNAfold using mixed memory model structures. Lastly we provide a number of extensions of the single sequence secondary structure models for pairwise alignment, pseudo knots and kissing hairpins, and overlapping RNA structures and protein coding sequences (mRNAs).

We conclude that probabilistic logic programming is a valuable addition to the existing toolset for prototyping and testing models that can form the basis of improved bioinformatic tasks involving RNA sequences.

# Perspectives

Natural extensions of the current work include:

- Benchmarking models that so far only have been prototyped.
- Include more signals like ribosome binding sites and 3' untranslated regions (3' UTRs) signals in the codon models.
- Integration of SCFG-like HMM sub-models into codon models for modeling *e.g.* structural RNAs in 3' UTRs.
- Include splice site signal *e.g.* using SCFG-like HMM sub-models into the codon models for eukaryotic gene finding.
- Develop a systematic exploration of model space *e.g.* using genetic algorithms.
- Using the reading frame sequence model presented in Paper II to examine the distribution of parameters across a large number of prokaryotic genomes, to see how the sequence of genes of a genome is phylogenetically distributed.
- Develop and evaluate generalized pair HMM models that use codon structures and ADPH length modeling over indels and match regions to make specific pair wise alignments of protein coding sequences.
- Test the use of generalized triple HMMs for iterative refinement of multiple alignments compared to generalized pair HMMs.

The compactness of viral genomes and the abundance of overlapping genomic features therein, make them obvious targets for the more elaborate and computationally taxing models. The combination of model features presented in this work should be sufficient for capturing the most important constraints acting on the sequential composition of viral genomes. One very interesting possibility would be to use such models to try to reliably predict how viral genome sequences evolve given the large number of different constraints acting on them.

# Conclusion

The overall aim of the present PhD project has been to develop, test and benchmark probabilistic logic HMM variants for biological sequence analysis using a unified framework consisting of the probabilistic logic programming language and machine learning system PRISM.

The modeling efforts have been based on HMMs due to their relative simplicity, their abundant prevalence in computational biology and elsewhere, and their inherent sequential nature that corresponds nicely to the inherently sequential nature of the molecules of life.

The gradual extension of HMMs have lead to a number of models ranging from mixed memory HMMs and other single sequence HMM model structures for modeling protein coding potential, indel HMMs for modeling the sequence of genes in a genome, double HMMs for supervised learning, generic ADPH length modeling, factorial HMMs for overlapping features, SCFG-like HMMs for RNA secondary structure modeling, pair and triple HMMs and SCFG like HMMs for alignment and complicated combined models like the mixed memory factorial HMM/SCFG-like HMMs with ADPH length modeling over structural components for modeling mRNAs.

The expressive power and generic machine learning algorithms of PRISM have enabled the development and testing of models consisting of novel combinations of model features that it would otherwise have been exceedingly difficult to explore (*e.g.* the mRNA models), yet alone to compare in terms of model fit, computational complexity and prediction performance without the *ad-hoc* heuristics and implementation differences, that otherwise cloud the differences and similarities between the general underlying model types.

The present project have proved the feasibility of, and helped establish the use of, probabilistic logic programming for biological sequence analysis. During the project, a number of novel probabilistic models and approaches for evaluating and applying such models for various tasks within bioinformatics have been developed. The project has laid out directions that will hopefully lead to further contributions to biological sequence analysis and bioinformatics in general.

# Bibliography

[1] Christiansen, H. (2007) Logic-statistic modeling and analysis of biological sequence data: a research agenda. in *Pre-Proceedings of the 2007 International Workshop on Abduction and Induction in Artificial Intelligence*, eds Doncescu, A., Flach, P. A., Inoue, K., Kakas, A. and Ray, O. pp 42–49.

[2] De Raedt, L., Frasconi, P., Kersting, K. and Muggleton, S., eds (2008) *Probabilistic Inductive Logic Programming - Theory and Applications* (Springer-Verlag).

[3] Durbin, R., Eddy, S., Krogh, A. and Mitcheson, G. (1998) *Biological Sequence Analysis* (Cambridge University Press).

[4] Watson, J. D. and Crick, F. H. C. (1953) A Structure for Deoxyribose Nucleic Acid. *Nature* **171**:737–738.

[5] Crick, F. H. C. (1970) Central Dogma of Molecular Biology. *Nature* **227**:561–563.

[6] Krogh, A., Mian, I. S. and Haussler, D. (1994) A hidden Markov model that finds genes in *E. coli* DNA. *Nucleic Acids Res.* **22**:4768–4778.

[7] Bradley, R. K., Roberts, A., Smoot, M., Juvekar, S., Do, J., Dewey, C., Holmes, I. and Pachter, L. (2009) Fast Statistical Alignment. *PLoS Comput. Biol.* **5**:e1000392.

[8] Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.* **17**:368–376.

[9] Paten, B., Herrero, J., Fitzgerald, S., Beal, K., Flicek, P., Holmes, I. and Birney, E. (2008) Genome-wide nucleotide-level mammalian ancestor reconstruction. *Genome Res.* **18**:1829–1843.

[10] Pedersen, J. S. and Hein, J. (2003) Gene finding with a hidden Markov model of genome structure and evolution. *Bioinformatics* **19**:219–227.

[11] Lunter, G., Miklós, I., Drummond, A., Jensen, J. L. and Hein, J. (2005) Bayesian coestimation of phylogeny and sequence alignment. *BMC Bioinformatics* **6**:83.

[12] Sato, T. and Kameya, Y. (2001) Parameter Learning of Logic Programs for Symbolic-Statistical Modeling. *J. Artif. Intell. Res.* **15**:391–454.

[13] Blattner, F. R., Plunkett, G., Bloch, C. A., Perna, N. T., Burland, V., Riley, M., Collado-Vides, J., Glasner, J. D., Rode, C. K., Mayhew, G. F., Gregor, J., Davis, N. W., Kirkpatrick, H. A., Goeden, M. A., Rose, D. J., Mau, B. and Shao, Y. (1997) The Complete Genome Sequence of *Escherichia coli* K-12. *Science* **277**:1453–1462.

[14] Keseler, I. M., Bonavides-Martinez, C., Collado-Vides, J., Gama-Castro, S., Gunsalus, R. P., Johnson, D. A., Krummenacker, M., Nolan, L. M., Paley, S., Paulsen, I. T., Peralta-Gil, M., Santos-Zavaleta, A., Shearer, A. G. and Karp, P. D. (2009) EcoCyc: A comprehensive view of *Escherichia coli* biology. *Nucleic Acids Res.* **37**:D464–D470.

[15] Staden, R. and McLachian, A. (1982) Codon preference and its use in identifying protein coding regions in long DNA sequences. *Nucleic Acids Res.* **10**:141–156.

[16] Staden, R. (1984) Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Res.* **12**:505–519.

[17] Delcher, A. L., Bratke, K. A., Powers, E. C. and Salzberg, S. L. (2007) Identifying bacterial genes and endosymbiont DNA with Glimmer. *Bioinformatics* **23**:673–679.

[18] Besemer, J. and Borodovsky, M. (2005) Genemark: web software for gene finding in prokaryotes, eukaryotes and viruses. *Nucleic Acids Res.* **33**:451–454.

[19] Larsen, T. and Krogh, A. (2003) EasyGene - a prokaryotic gene finder that ranks ORFs by statistical significance. *BMC Bioinformatics* **4**:21.

[20] Munch, K. and Krogh, A. (2006) Automatic generation of gene finders for eukaryotic species. *BMC Bioinformatics* **7**:263.

[21] Hyatt, D., Chen, G., LoCascio, P., Land, M., Larimer, F. and Hauser, L. (2010) Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC bioinformatics* **11**:119.

[22] Korf, I., Flicek, P., Duan, D. and Brent, M. R. (2001) Integrating genomic homology into gene structure prediction. *Bioinformatics* **17**:S140–148.

[23] Gross, S. S. and Brent, M. R. (2006) Using multiple alignments to improve gene prediction. *Journal of computational biology* **13**:379–393.

[24] Allen, J. E., Majoros, W. H., Pertea, M. and Salzberg, S. L. (2006) JIGSAW, GeneZilla, and GlimmerHMM: puzzling out the features of human genes in the ENCODE regions. *Genome Biol.* **7 Suppl 1**:1–13.

[25] Muggleton, S., King, R. and Sternberg, M. (1992) Protein secondary structure prediction using logic-based machine learning. *Protein Engineering* **5**:647–657.

[26] Searls, D. B. (1993) The computational linguistics of biological sequences. in *Artificial Intelligence and Molecular Biology*, ed Hunter, L. (AAAI Press), pp 47–120.

[27] Searls, D. B. (1995) String variable grammar: A logic grammar formalism for the biological language of dna. *Journal of Logic Programming* **24**:73–102.

[28] Searls, D. and Murphy, K. (1995) Automata-theoretic models of mutation and alignment. *Proc. Int. Conf. Intell. Syst. Mol. Biol.* **3**:341–349.

[29] Landwehr, N., Mielikinen, T., Eronen, L., Toivonen, H. and Mannila, H. (2007) Constrained hidden markov models for population-based haplotyping. *BMC Bioinformatics* **8**:S2.

[30] Christiansen, H. and Dahmcke, C. M. (2007) A Machine Learning Approach to Test Data Generation: A Case Study in Evaluation of Gene Finders. in *Machine*

*Learning and Data Mining in Pattern Recognition*, ed Perner, P. (Springer-Verlag), pp 741–755.

[31] Christiansen, H., Have, C. T., Lassen, O. T. and Petit, M. (2011) Taming the Zoo of Discrete HMM Subspecies & some of their Relatives. in *Proceedings of the first International Work-Conference on Linguistics, Biology and Computer Science: Interplays*, eds Bel-Enguix, G., Dahl, V. and Jimnez-Lpez, M. D. (IOS Press), pp 28–42.

[32] Sato, T. and Kameya, Y. (1997) PRISM: A Language for Symbolic-Statistical Modeling. in *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, ed Pollack, M. E. (Morgan Kaufmann), pp 1330–1339.

[33] Dempster, A., Laird, N. and Rubin, D. (1977) Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Stat. Soc. B* **39**:1–38.

[34] Sato, T., Kameya, Y. and Kurihara, K. (2008) Variational Bayes via propositionalized probability computation in PRISM. *Ann. Math. Artif. Intell.* **54**:135–158.

[35] Bellman, R. (1952) On the Theory of Dynamic Programming. *Proc. Natl. Acad. Sci. U.S.A.* **38**:716–719.

[36] Schwarz, G. (1978) Estimating the dimension of a model. *Ann. Stat.* **6**:461–464.

[37] Cheeseman, P. and Stutz, J. (1995) Bayesian classification (AutoClass): Theory and results. in *Advances in Knowledge Discovery and Data Mining*, eds Fayyad, U., Piatesky, G., Smyth, P. and Uthurusamy, R. (MIT Press), pp 153–180.

[38] Sato, T. (2009) Generative Modeling by PRISM. in *Logic Programming, 25th International Conference, ICLP 2009*, eds Hill, P. M. and Warren, D. S. (Springer-Verlag), pp 24–35.

[39] Sato, T., fa Zhou, N., Kameya, Y. and Izumi, Y. (2010) PRISM Users Manual (Version 2.0). http://sato-www.cs.titech.ac.jp/prism/download/prism20.pdf.

[40] Robinson, J. A. (1965) A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM* **12**:23–41.

[41] Horn, A. (1951) On sentences which are true of direct unions of algebras. *J. Symbolic Logic* **16**:14–21.

[42] van Emden, M. and Kowalski, R. (1976) The Semantics of Predicate Logics as a Programming Language. *J. ACM* **23**:733–742.

[43] Kameya, Y., Sato, T. and Zhou, N.-F. (2004) Yet More Efficient EM Learning for Parameterized Logic Programs by Inter-Goal Sharing. in *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004*, eds de Mántaras, R. L. and Saitta, L. (IOS Press), pp 490–494.

[44] Tamaki, H. and Sato, T. (1986) Old resolution with tabulation. in *Proceedings of the Third International Conference on Logic Programming*, ed Shapiro, E. Y. (Springer-Verlag), pp 84–98.

[45] Baum, L. and Egon, J. (1967) An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology. *Bull. Am. Meteorol. Soc.* **73**:360–363.

[46] Baker, J. K. (1979) Trainable grammars for speech recognition. *J. Acoust. Soc. Am.* **65**:S132–S132.

[47] Pearl, J. (1982) Reverend Bayes on inference engines: A distributed hierarchical approach. in *Proceedings of the Second National Conference on Artificial Intelligence*, ed Waltz, D. (AAAI Press), pp 133–136.

[48] Viterbi, A. J. (1967) Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inform. Theor.* **13**:260–269.

[49] Jensen, J. L. W. V. (1906) Sur les fonctions convexes et les ingalits entre les valeurs moyennes. *Acta Math.* **30**:175–193.

[50] Kameya, Y. and Sato, T. (2000) Efficient EM Learning with Tabulation for Parameterized Logic Programs. in *Computational Logic - CL 2000: First International Conference*, eds Lloyd, J., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Palamidessi, C., Pereira, L. M., Sagiv, Y. and Stuckey, P. J. (Springer-Verlag), pp 269–284.

[51] Christiansen, H. and Lassen, O. T. (2009) Preprocessing for Optimization of Probabilistic-Logic Models for Sequence Analysis. in *Logic Programming.*

*25th International Conference, ICLP 2009*, eds Hill, P. M. and Warren, D. S. (Springer-Verlag), pp 70–83.

[52] Lassen, O. T. (2008) Biosequence Analysis in PRISM. in *Logic Programming. 24th International Conference, ICLP 2008 (doctoral consortium)*, eds Banda, M. G. and Pontelli, E. (Springer-Verlag), pp 809–810.

[53] Have, C. T. (2009) Logic-Statistic Models with Constraints for Biological Sequence Analysis. in *Logic Programming. 25th International Conference, ICLP 2009 (doctoral consortium)*, eds Hill, P. M. and Warren, D. S. (Springer-Verlag), pp 549 – 550.

[54] Christiansen, H. and Gallagher, J. P. (2009) Non-discriminating Arguments and their Uses. in *Logic Programming. 25th International Conference, ICLP 2009*, eds Hill, P. M. and Warren, D. S. (Springer-Verlag), pp 55–69.

[55] Fawcett, T. (2006) An introduction to ROC analysis. *Pattern Recogn. Lett.* **27**:861–874.

[56] DeLong, E. R., DeLong, D. M. and Clarke-Pearson, D. L. (1988) Comparing the areas under two or more correlated receiver operating characteristic curves: A nonparametric approach. *Biometrics* **44**:pp. 837–845.

[57] Baum, L. and Petrie, T. (1966) Statistical Inference for probabilistic functions of finite state Markov chains. *Ann. Math. Stat.* **37**:1554–1563.

[58] Rabiner, L. R. (1989) A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* **77**:257–286.

[59] Krogh, A., Brown, M., Mian, I. S., Sjlander, K. and Haussler, D. (1994) Hidden Markov Models in Computational Biology : Applications to Protein Modeling. *J. Mol. Biol.* **235**:1501–1531.

[60] Felsenstein, J. and Churchill, G. A. (1996) A Hidden Markov Model approach to variation among sites in rate of evolution. *Mol. Biol. Evol.* **13**:93–104.

[61] Siepel, A. and Haussler, D. (2004) Computational identification of evolutionarily conserved exons. in *Proceedings of the Eighth Annual International Conference on Research in Computational Molecular Biology*, eds Bourne, P. E. and Gusfield, D. (ACM), pp 177–186.

[62] Hobolth, A., Christensen, O. F., Mailund, T. and Schierup, M. H. (2007) Genomic Relationships and Speciation Times of Human, Chimpanzee, and Gorilla Inferred from a Coalescent Hidden Markov Model. *PLoS Genet.* **3**:e7.

[63] Dutheil, J. Y., Ganapathy, G., Hobolth, A., Mailund, T., Uyenoyama, M. K. and Schierup, M. H. (2009) Ancestral Population Genomics: The Coalescent Hidden Markov Model Approach. *Genetics* **183**:259–274.

[64] Bradley, R. K. and Holmes, I. (2007) Transducers: an emerging probabilistic framework for modeling indels on trees. *Bioinformatics* **23**:3258–3262.

[65] Westesson, O. and Holmes, I. (2009) Accurate detection of recombinant breakpoints in whole-genome alignments. *PLoS Comput. Biol.* **5**:e1000318.

[66] Frellsen, J., Moltke, I., Thiim, M., Mardia, K. V., Ferkinghoff-Borg, J. and Hamelryck, T. (2009) A probabilistic model of RNA conformational space. *PLoS Comput. Biol.* **5**:e1000406.

[67] Boomsma, W., Mardia, K. V., Taylor, C. C., Ferkinghoff-Borg, J., Krogh, A. and Hamelryck, T. (2008) A generative, probabilistic model of local protein structure. *Proc. Natl. Acad. Sci. U.S.A.* **105**:8932–8937.

[68] Miklós, I., Novák, A., Satija, R., Lyngsø, R. and Hein, J. (2009) Stochastic models of sequence evolution including insertion-deletion events. *Stat. Meth. Med. Res.* **18**:453–485.

[69] Baum, L. E., Petrie, T., Soules, G. and Weiss, N. (1970) A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Stat.* **41**:164–171.

[70] Ghahramani, Z. (1998) Learning Dynamic Bayesian Networks. in *Adaptive Processing of Sequences and Data Structures, International Summer School on Neural Networks, "E.R. Caianiello"-Tutorial Lectures*, eds Lee Giles, C. and Marco Gori, M. (Springer-Verlag), pp 168–197.

[71] Chomsky, N. (1959) On certain formal properties of grammars. *Inform. Contr.* **2**:137–167.

[72] Rabin, M. O. (1963) Probabilistic Automata. *Inform. Contr.* **6**:230–245.

[73] Moore, E. F. (1956) Gedanken-experiments on sequential machines. in *Automata Studies*, eds Shannon, C. E. and McCarthy, J. (Princeton University Press), pp 129–153.

[74] Markov, A. A. (1907) Extension of the limit theorems of probability theory to a sum of variables connected in a chain. *The Notes of the Imperial Academy of Sciences of St. Petersburg, Ser. 8, Physio-Mathematical College, No. 1* **22**:1.

[75] Saul, L. K. and Jordan, M. I. (1999) Mixed Memory Markov Models: Decomposing Complex Stochastic Processes as Mixtures of Simpler Ones. *Mach. Learn.* **37**:75–87.

[76] Lukashin, A. and Borodovsky, M. (1998) GeneMark.hmm: new solutions for gene finding. *Nucleic Acids Res.* **26**:1107–1115.

[77] Stormo, G. D. and Haussler, D. (1994) Optimally Parsing a Sequence into Different Classes Based on Multiple Types of Information. in *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, eds Altman, R., Brutlag, D., Karp, P., Lathrop, R. and Searls, D. (AAAI/MIT Press), pp 369–375.

[78] Bobbio, A., Horváth, A., Scarpa, M. and Telek, M. (2003) Acyclic discrete phase type distributions: properties and a parameter estimation algorithm. *Perform. Eval.* **54**:1–32.

[79] Sakakibara, Y., Brown, M., Underwood, R. C., Mian, I. S. and Haussler, D. (1994) Stochastic Context-Free Grammars for Modeling RNA. in *Proceedings of the 27th Annual Hawaii International Conference on System Sciences. Volume 5 : Biotechnology Computing*, ed Hunter, L. (IEEE Computer Society Press), pp 284–294.

[80] Eddy, S. R. and Durbin, R. (1994) RNA sequence analysis using covariance models. *Nucleic Acids Res.* **22**:2079–2088.

[81] Knudsen, B. and Hein, J. (2003) Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Res.* **31**:3423–3428.

[82] Ghahramani, Z. and Jordan, M. I. (1996) Factorial Hidden Markov Models. *Mach. Learn.* **29**:245–273.

[83] Theil Have, C. and Christiansen, H. (2011) Modeling repeats in DNA using extended probabilistic regular expressions. in *Proceedings of the 1st International Work-Conference on Linguistics, Biology and Computer Science: Interplays*, eds Bel-Enguix, G., Dahl, V. and Jimnez-Lpez, M. D. (IOS Press), pp 55–70.

# Paper I

# Probabilistic Logic Models of Protein Coding Potential

Søren Mørk[1] and Ian Holmes[2]

[1]Department of Science, Systems and Models, Roskilde University, Denmark

[2]Department of Bioengineering, University of California, Berkeley, USA

**Publication Status:**

*Published*

BIOINFORMATICS ORIGINAL PAPER

Vol. 28 no. 5 2012, pages 636–642
doi:10.1093/bioinformatics/btr698

Sequence analysis

Advance Access publication January 3, 2012

# Evaluating bacterial gene-finding HMM structures as probabilistic logic programs

Søren Mørk[1,*] and Ian Holmes[2]

[1]Department of Science, Systems and Models, Roskilde University, 4000 Roskilde, Denmark and [2]Department of Bioengineering, University of California, Berkeley, CA 94720, USA

Associate Editor: John Quackenbush

**ABSTRACT**

**Motivation:** Probabilistic logic programming offers a powerful way to describe and evaluate structured statistical models. To investigate the practicality of probabilistic logic programming for structure learning in bioinformatics, we undertook a simplified bacterial gene-finding benchmark in PRISM, a probabilistic dialect of Prolog.

**Results:** We evaluate Hidden Markov Model structures for bacterial protein-coding gene potential, including a simple null model structure, three structures based on existing bacterial gene finders and two novel model structures. We test standard versions as well as ADPH length modeling and three-state versions of the five model structures. The models are all represented as probabilistic logic programs and evaluated using the PRISM machine learning system in terms of statistical information criteria and gene-finding prediction accuracy, in two bacterial genomes. Neither of our implementations of the two currently most used model structures are best performing in terms of statistical information criteria or prediction performances, suggesting that better-fitting models might be achievable.

**Availability:** The source code of all PRISM models, data and additional scripts are freely available for download at: http://github.com/somork/codonhmm.

**Contact:** soer@ruc.dk

**Supplementary information:** Supplementary data are available at *Bioinformatics* online.

Received on July 7, 2011; revised on December 13, 2011; accepted on December 19, 2011

## 1 INTRODUCTION

Protein coding potential has long been recognized as the most important signal for automated gene finding (Fickett and Tung, 1992; Staden and McLachian, 1982; Staden, 1984). The introduction of Hidden Markov Models (HMMs) for gene finding by Krogh *et al.* (1994a) sparked the production of a large number of HMM-based, single-sequence gene finders that capture this signal and other signals (Besemer *et al.*, 2001; Burge and Karlin, 1997; Henderson *et al.*, 1997; Korf, 2004; Krogh, 1997; Kulp *et al.*, 1996; Larsen and Krogh, 2003; Lomsadze *et al.*, 2005; Lukashin and Borodovsky, 1998; Majoros *et al.*, 2003, 2004; Munch and Krogh, 2006; Reese *et al.*, 2000; Shmatkov *et al.*, 1999).

*To whom correspondence should be addressed.

Restricting our survey to the simplest case of bacterial gene-finding, the basic codon structure of protein-coding genes has so far been modeled using the following structures:

The Ecoparse gene finder introduced by Krogh *et al.* (1994a) is based on a standard HMM architecture with a silent state governing codon distributions via transitions to 64 separate three state submodels where each state of the codon submodels had fixed emissions of a single character. Stormo and Haussler (1994) introduced the Generalized Hidden Markov Model (GHMM), a type of HMM with duration offering the possibility of emissions of sequences rather than just characters from each state (Rabiner, 1989). Most single sequence *de novo* gene finders have since been based on GHMM's using either emissions of codons according to a three-periodic inhomogeneous Markov Chain (Besemer and Borodovsky, 1999; Borodovsky and McInich, 1993) or using higher ordered emissions, typically fifth ordered (Lukashin and Borodovsky, 1998) or variable ordered emissions (Delcher *et al.*, 1999; Salzberg *et al.*, 1998). The single character emitting HMM based gene finder models has subsequently been elaborated by also using higher ordered emissions (Krogh, 1997), as well as using Acyclic Discrete Phase type length modeling (Bobbio *et al.*, 2003), in Easygene and Agene (Larsen and Krogh, 2003; Munch and Krogh, 2006).

A large number of different HMM architectures have been developed during the recent decades including profile-HMMs (Krogh *et al.*, 1994b), pair-HMMs (Durbin *et al.*, 1998), input–output HMMs or transducers (Bradley and Holmes, 2007), factorial-HMMs (Ghahramani and Jordan, 1996) and mixed memory HMMs (Saul and Jordan, 1999). These models each combine different numbers of emitted sequences, hidden state chains, delete and insert states and different conditioning schemes for emission probabilities and transition probabilities. Employing as efficient model structures as possible for biological sequence analysis is paramount for coping with the vast amounts of sequence data currently being generated. The structure space of possible different combinations is large and exploring it for efficient models for biological sequence analysis is limited by the time-consuming development of dedicated machine learning algorithms and benchmarking procedures. Additionally, efforts to directly compare models are clouded by implementation differences and the heuristic fine-tunings developed through the decades that optimize the models in terms of the sequence analysis task at hand. To overcome these challenges, we use the probabilistic logic programming language and machine learning system PRISM. PRISM offers a generic representation of a large number of diverse model types that subsumes HMMs, SCFGs and Bayesian Networks. The PRISM machine learning system uses a general set of algorithms to perform machine learning tasks for all models (Sato and Kameya, 2001). Using this approach, we can directly compare the performance of the underlying model structures of various gene finders without differences due to training procedures, the inclusion of different kinds of additional signals, pre/post-processing of data or other implementation differences. The generalized nature of PRISM programs allows the formulation of all types of model structures previously used as well as non-standard conditioning schemes (such as mixed memory HMM's) and the ability to export those conditioning schemes to other types of models [such as pair-HMMs, factorial-HMMs and Stochastic Context Free Grammars (Christiansen *et al.*, 2011)]. An alternative approach for using PRISM for evaluating gene finder programs based on artificially generated datasets is given in Christiansen and Dahmcke (2007).

## 2 APPROACH

To keep things as simple as possible, our preliminary benchmark uses the well-studied test case of bacterial gene finding. We use *Escherichia coli* as a test bed for our model structure comparisons, due to the availability of experimentally verified annotations and textbook gene structures. To test the robustness of the performance of the models, we have duplicated the experiments using a distantly related (and less well annotated) *Bacillus subtilis* genome. We evaluate the performance of the models based on learning statistics from learn sessions on the complete experimentally verified *E.coli* dataset as well as on prediction performances based on 5-fold cross-validation experiments for both genomes. In order to compare the performances of different gene finder model structures, we have implemented the first HMM-based gene finder model structure (ecoparse), the two most common current model structures that capture protein coding potential: a structure with fifth ordered emissions and one with inhomogeneous three-periodic Markov chain like emissions, as well as two new types of structures consisting of a model based on a amino acid hidden state sequence and a model based on a mixed memory HMM. In addition to the basic model structures, we have also included two straightforward extensions of the model structures: the first extension involves length modeling, an important feature of contemporary gene finders. Since the codon usage of highly expressed, normally expressed and laterally transferred/phage genes are known to differ (Blattner *et al.*, 1997), the second extension are three-state versions of the models that encode these three separate classes of genes.

## 3 METHODS

### 3.1 PRISM

PRISM is a logic programming language and machine learning system (Sato and Kameya, 2001). The earliest general-purpose engine for bioinformatics automata was implemented in Prolog by Searls and Murphy (1995); PRISM is effectively a probabilistic dialect of B-prolog, allowing a pure declarative approach that unifies the description of model and data. The distinguishing feature of PRISM is the build-in predicate *msw/2*, that represents discrete random variables. This allows Prolog's abducible facts to be assigned probabilistic parameters. Executing a query using a tabled variant of the prologs SLD resolution produces a tabled search tree—an explanation graph (corresponds to a dynamic programming matrix), enabling efficient parameter estimation using a generic expectation–maximization algorithm running on explanation graphs as reviewed in Sato (2009).

The usefulness of HMM's is based on a small set of algorithms for 'decoding' a sequence, i.e. calculating the most probable path $\Pi^*$ and its probability $\mathbf{P}(\Pi^*)$(the Viterbi algorithm), the total probability of a sequence (the Forward algorithm or the Backward Algorithm) and the posterior probability that a specific character at a given position is emitted by a certain state (a combination of the Forward and Backward algorithms) and training models on data (the Baum–Welch algorithm) (Rabiner, 1989).

These algorithms are subsumed by the graphical EM algorithm that PRISM runs on the proof tree-like structures generated by a PRISM model, which means that as soon as one has formulated a PRISM model, one can parameterize it via EM from training data, calculate various probabilities of interest, use the parameterized model to decode data as well as generate simulated data from the parameterized model (Sato *et al.*, 2010). In addition to the standard EM algorithm and a Deterministic Annealing EM algorithm (Ueda and Nakano, 1998) that produces estimates of the maximum likelihood parameter values of a model, PRISM also offers a variational Bayes EM algorithm (Sato *et al.*, 2008) and a Deterministic Annealing Variational Bayes EM algorithm (Katahira *et al.*, 2008).

### 3.2 HMMs

An HMM is fully characterized by a set of transition probabilities between unobserved states and a set of emission probabilities of observed characters emitted from states. The structure of an HMM can be identified from the factorization scheme used to calculate the joint probability $P(O,\Pi)$ of an observed sequence $O$ and a hidden state path $\Pi$. Given an observed sequence $O=x_1,x_2,...,x_n$ and a path $\Pi=\pi_1,\pi_2,...,\pi_n$, the transition probabilities $a_{kl}$ are defined for position $i$ and states $k$ and $l$, as:

$$a_{kl}=\mathbf{P}(\pi_i=l\,|\,\pi_{i-1}=k), \tag{1}$$

and the emission probabilities $e_k(b)$ are defined for path $\Pi$, character $x$ in position $i$, state $k$ and character $b$, as:

$$e_k(b)=\mathbf{P}(x_i=b\,|\,\pi_i=k). \tag{2}$$

The joint probability of observing the sequence $X$ and the path $\Pi$ is:

$$\mathbf{P}(X,\Pi)=a_{0\pi_1}\prod_{i=1}^{n}e_{\pi_i}(x_i)a_{\pi_i\pi_{i+1}}. \tag{3}$$

The PRISM equivalent of a HMM is a collection of "values" predicates, declaring the values that can be taken by various random variables. These random variables represent the outcome spaces for probabilistic transitions and emissions. A recursive structure, with predicates for initiation and termination, connects these random variables to the state and emission sequences, specifying the partitioning scheme of the joint probability. The following is the complete source code of a 2 state DNA HMM (% marks comments)

```
%  parameters:
values(transition(state(begin)),[state(1),state(2),end]).
values(transition(state(1)),[state(1),state(2),end]).
values(transition(state(2)),[state(1),state(2),end]).
values(emission(state(1)),[a,c,g,t]).
values(emission(state(2)),[a,c,g,t]).

% initiation:
model(Observables) :-
   recursion(state(begin),Observables).

% recursion structure:
recursion(state(Si),[Xi|Rest]) :-
   msw(emission(state(Si)),Xi),
   msw(transition(state(Si)),NS),
   recursion(NS,Rest).

% termination:
recursion(end,[]).
```

This fully functional PRISM program can be parameterized, decoded to or sampled from using built-in PRISM functions. For example, to fit the model to a sequence 'aaagt', one could use learn([model([a,a,a,g,t])]).; to Viterbi-decode that same sequence, viterbif(model([a,c,g,t])).; and to sample a sequence into samples once, get_samples(1,model(X),Samples). Since the outcome spaces are shared for transitions and emissions a more compact program would be the replacement of the parameters section with:

```
values(transition(state(_)),[state(1),state(2),end]).
values(emission(state(_)),[a,c,g,t]).
```

where the underscore denotes 'the anonymous variable' which is simply a placeholder for any logic variable. Note that except for the *values* and *msw* predicates that are special PRISM predicates, the names of the remaining predicates and variables are arbitrary and replacing them with single letters would result in a program with the exact same properties (consisting of a single line of code with 138 characters including 5 white spaces).

The elegant brevity of the source code for a HMM given as a PRISM program and its close structural resemblance with the model architecture makes it very easy to produce novel models via small changes, e.g. changing the emission probabilities to:

```
values(emission(_,_),[a,c,g,t]).
```

and the recursive formula to:

```
 recursion(state(Si),P1,[Xi|Rest]): -
    msw(emission(state(Si),P1),Xi),
    msw(transition(state(Si)),NS),
    recursion(NS,Xi,Rest).
```

transforms a standard HMM into a second-order HMM with emissions conditioned on the present state and the previous emission, whereas changing to:

```
 values(transition(_,_),[state(1),state(2)]).
```

with

```
  recursion(state(Si),P1,[Xi|Rest]):-
     msw(emission(state(Si)),Xi),
     msw(transition(state(Si)),P1),NS),
     recursion(NS,Xi,Rest).
```

is a second order like mixed memory HMM with transitions conditioned on present state and previous emission. In the following, we will use such model extensions to develop novel model structures suitable for modeling protein coding potential.

## 3.3 Models of protein coding potential

- iid.psm is an IID like zeroth order emission HMM with a single state that emits over the alphabet of {acgt}. The model captures base frequencies and has a geometric length distribution. This type of model has traditionally been used as null model or as a model of intergenic sequences.
- mc5.psm is a fifth order Markov chain like HMM, with a single state and emissions conditioned on state and five previous emissions. The model captures di-codon preferences of coding regions.
- i3pmc.psm is a inhomogeneous three-periodic Markov chain with three sequential states with the emission of the first state conditioned on state, the emission of the second state conditioned on state and the previous emission, and the emission of the third state conditioned on state and the two previous emissions. All states emits from {acgt}.
- eco.psm the ecoparse architecture with three consecutive states with single symbol emissions for each codon combination and a single silent state to control the codon distribution.
- aa.psm has 20 hidden states corresponding to the 20 amino acids that emits synonymous codons. The hidden state path corresponds to the translated amino acid sequence of the encoded protein. aa models

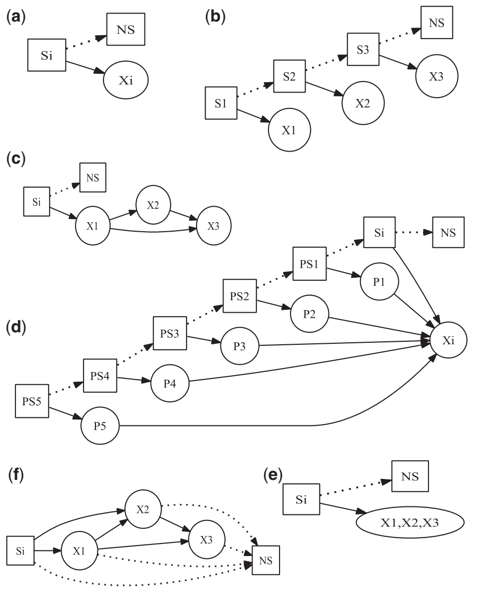


**Fig. 1.** Graphical representation of the conditioning schemes of the underlying structure of the models. (**a**) iid.psm; (**b**) eco.psm; (**c**) i3pmc.psm; (**d**) mc5.psm; (**e**) aa.psm; and (**f**) mm.psm. Squares represent the hidden State(S), Previous State(PS) or Next State(S); circles represent emissions (X) or past emissions (P). The dotted arrows are conditional transition probabilities and the full arrows are conditional emission probabilities.

codon bias and amino acid sequence composition of the encoded protein simultaneously. The model assigns higher probability to synonymous than non-synonymous sequences, hence it is capable of attaining higher likelihood if the genes have more similar amino acid sequences than nucleotide sequences. The amino acid frequencies are governed by transitions from a silent state.

- mm.psm is a three state mixed memory HMM (Saul and Jordan, 1999) with higher ordered transitions, i.e. with transitions conditioned on the previous emissions. The model recognizes start and stop codons via making the transition probabilities conditional on the previous two and the present emission. Triplet emissions with first position conditioned on state, second position conditioned on state and previous emission and third emission conditioned on state and previous two emissions.

All models (except iid.psm that is only used as null model) comes in three variants: a standard version as outlined above, an Acyclic Discrete Phase Type (ADPH) length modeling version, and a three-state version with three separate states that cannot transition to each other (e.g. once there is a transition to one of the states, the state path stays in that state until the end state) that corresponds to the three classes of bacterial genes: highly expressed genes, normally expressed genes and laterally transferred/phage genes. ADPH versions are created via adding six lines of code to the models and the three-state versions by adding two lines of code to the models. A graphical presentation of the standard version model structures are given in Figure 1.

## 4 EXPERIMENTS AND RESULTS

### 4.1 Training data

The *E.coli* genome used is the Refseq NC_000913.2 wild-type K12 strain MG1665 genome (Blattner *et al.*, 1997). The *E.coli* training set comprise the 2413 experimentally verified *E.coli* MG1665 genes from the ecocyc annotation (Keseler *et al.*, 2009) with canonical start and stop codons and no frameshift mutations. The complete verified ecocyc training set comprises a total of 2.487.654 nucleotides. The *Bacillus* training set used is the Refseq NC_000964 *B.subtilis* subsp. subtilis str. 168 (Kunst *et al.*, 1997). The complete *Bacillus* training set comprise 4155 genes with canonical start and stop codons and no frameshift mutations from the .gbk annotation (3.695.139 nt in total). The *E.coli* test set comprises all 68.826 potential ORFs from canonical start codon (ATG, GTG, TTG) to canonical stop codon (TAA, TAG, TGA) with a length over 60 nt (13.554.717 nt in total). The *Bacillus* test set comprises the equivalent 63.198 potential ORFs (11.438.079 nt in total). Training sets for all cross-validation studies were produced by randomly assigning the genes of the full training set into five subsets. The *E.coli* cross-validation training sets contains $\sim$ 500 genes each (see Supplementary Material for exact dataset sizes). Each cross-validation training sets were removed from the test sets and golden standards for the prediction performance evaluations. All sequence sets were converted to a format compatible with the PRISM models, i.e. a collection of e.g.: 'model([a,t,g,a,a,a,t,a,a]).'. Prediction performances were evaluated using the complete training sets as golden standards.

### 4.2 Training algorithms

All models are written as .psm files (available in for download). An additional prolog file *default_setting.pl* contains code for batch execution and settings of learning modes. The models were trained on the training datasets using *learn* with standard settings of VB-EM [see default_setting.pl and Sato *et al.* (2010) for instructions and other available options] with learning statistics and parameter values stored in separate files. Viterbi probabilities were calculated using *viterbi*.

### 4.3 Learning statistics

PRISM reports the following information relevant for model selection: the size of the explanation graph, the size of the table space used, the number of EM iterations, the total time of learning, the number of parameters, the number of parameter instances and the variational free energy values obtained after VB-EM learning. The variational free energy score is an approximation of log of the marginal likelihood (like the Bayesian Information Criterion), and is explained in detail in Sato *et al.* (2008). Table 1 shows a summary of the statistics from the learning sessions.

### 4.4 Prediction accuracy

Prediction accuracy was obtained using log-odds values obtained from the Viterbi probabilities of all potential open reading frames with a length over 60 nt for a given model and its null model (iid.psm). The log odds scores of the potential ORFs sequence were divided into a positive set (according to the golden standard) and a negative set. Confusion matrices and prediction performance metrics were calculated using all observed log odds scores of the positive set as thresholds. In order to ensure that a choice of log odds threshold is not biased in favor of any particular model, the prediction metrics example reported is the one that for each model maximizes the difference between true positive rate (TPR) and false positive rate (FPR), corresponding to the intercept of the ROC curve with the parallel to the no-discrimination line. Performance measures are all based on accurate prediction of stop codon position only, since discovery of novel genes is more interesting than correctly annotating the starting position of an already known gene. Prediction performance was determined by TPR (sensitivity/precision), FPR (Recall) and ROC curve area under the curve (AUC) value. AUC values were calculated from the pairs of TPRs and FPRs using the trapezoidal rule, which given the large number of points should be a relatively close approximation (DeLong *et al.*, 1988). Subsequently, the significance of the ranking of the models based on AUC were calculated through pairwise paired $t$-tests in R.

The receiver operator characteristics (ROC) curves from the *E.coli* cross-validation experiments is given for the standard versions of the models in Figure 2. (ROC curves for all the cross-validation experiments are available in Supplementary Materials.)

Table 2 gives the average ROC optimized prediction performances of the *E.coli* cross-validation experiments $\pm 1$ SD as well as the average AUC values $\pm 1$ SD. ROC optimized confusion matrices and prediction performances including match coefficient and Mathews correlation coefficient are available in Supplementary Materials. $P$-values of pairwise paired $t$-tests of the ranking of the *E.coli* AUC values are given in Table 3. $P$-values from pairwise paired $t$-tests of the prediction performance ranking: three-state version > standard version > adph version, using the *E.coli* AUC values are given in Table 4. The corresponding tables from the *Bacillus* cross-validation experiments are given as Tables 5–7. ($P$-values of pairwise paired $t$-tests of the ranking based on variational free energy scores are available in Supplementary Materials.)

**Table 1.** Learn Statistics of the different models trained on the entire *E.coli* verified ecocyc dataset

| Model | Free parameters | Variational free energy | Graph nodes | Learn time |
|---|---|---|---|---|
| aa_s | 507 | $-3.229 \times 10^6$ | $2.476 \times 10^6$ | 750 |
| eco_s | 67 | $-3.252 \times 10^6$ | $9.061 \times 10^6$ | 3024 |
| i3pmc_s | 64 | $-3.26 \times 10^6$ | $4.124 \times 10^6$ | 201 |
| mc5_s | 3139 | $-3.307 \times 10^6$ | $7.448 \times 10^6$ | 5 |
| mm_s | 127 | $-3.244 \times 10^6$ | $4.124 \times 10^6$ | 190 |
| aa_3 | 1523 | $-3.211 \times 10^6$ | $7.428 \times 10^6$ | 4451 |
| eco_3 | 203 | $-3.235 \times 10^6$ | $27.183 \times 10^6$ | 24 128 |
| i3pmc_3 | 74 | $-3.257 \times 10^6$ | $12.371 \times 10^6$ | 492 |
| mc5_3 | 9419 | $-3.307 \times 10^6$ | $22.343 \times 10^6$ | 25 |
| mm_3 | 383 | $-3.228 \times 10^6$ | $12.371 \times 10^6$ | 528 |
| aa_a | 486 | $-3.245 \times 10^6$ | $16.433 \times 10^6$ | 3193 |
| eco_a | 67 | $-3.250 \times 10^6$ | $34.470 \times 10^6$ | 12 138 |
| i3pmc_a | 64 | $-3.26 \times 10^6$ | $24.646 \times 10^6$ | 597 |
| mc5_a | 3139 | $-3.306 \times 10^6$ | $49.572 \times 10^6$ | 15 |
| mm_a | 64 | $-3.26 \times 10^6$ | $24.646 \times 10^6$ | 621 |

Values reported includes the number of free parameters, the Variational Free Energy score after learning, the number of nodes in the explanation graph, and the learn time in minutes.

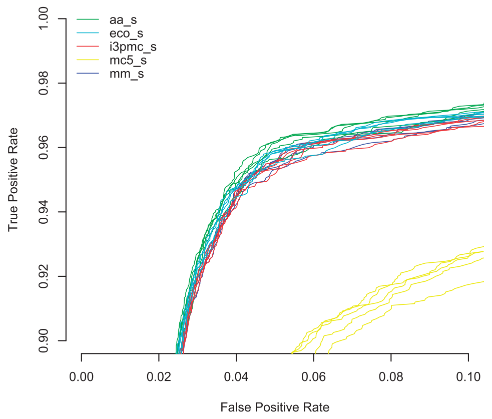


**Fig. 2.** *E.coli* cross-validation ROC curves for the standard models using thresholds over all log odds values in the positive set. Notice that only the area from 0.0–0.1 FPR and 0.9–1.0 TPR is shown.

**Table 2.** Prediction performances of the *E.coli* verified ecocyc cross-validation experiments

| Model | TPR | FPR | AUC |
|---|---|---|---|
| aa | 0.957 ± 0.004 | 0.045 ± 0.0 | 0.978 ± 0.001 |
| eco | 0.956 ± 0.002 | 0.047 ± 0.0 | 0.976 ± 0.001 |
| i3pmc | 0.95 ± 0.002 | 0.043 ± 0.0 | 0.975 ± 0.001 |
| mc5 | 0.902 ± 0.002 | 0.062 ± 0.0 | 0.951 ± 0.003 |
| mm | 0.951 ± 0.001 | 0.044 ± 0.0 | 0.976 ± 0.001 |
| aa_3 | 0.972 ± 0.002 | 0.042 ± 0.0 | 0.987 ± 0.0 |
| eco_3 | 0.971 ± 0.007 | 0.046 ± 0.0 | 0.985 ± 0.002 |
| i3pmc_3 | 0.959 ± 0.004 | 0.048 ± 0.0 | 0.979 ± 0.001 |
| mc5_3 | 0.9 ± 0.004 | 0.064 ± 0.0 | 0.952 ± 0.002 |
| mm_3 | 0.965 ± 0.003 | 0.046 ± 0.0 | 0.983 ± 0.002 |
| aa_a | 0.954 ± 0.003 | 0.044 ± 0.0 | 0.977 ± 0.001 |
| eco_a | 0.951 ± 0.005 | 0.044 ± 0.0 | 0.975 ± 0.001 |
| i3pmc_a | 0.947 ± 0.003 | 0.042 ± 0.0 | 0.974 ± 0.001 |
| mc5_a | 0.896 ± 0.005 | 0.064 ± 0.0 | 0.947 ± 0.002 |
| mm_a | 0.954 ± 0.005 | 0.045 ± 0.0 | 0.976 ± 0.001 |

Mean values ±1 SD.

## 5 RESULTS

The performance of the models both in terms of statistical fit ranked as per the variational free energy value after training on the complete training set and in terms of prediction performance evaluated via cross-validation experiments suggests the following general ranking: aa >= eco >= mm > i3pmc > mc5. Interestingly, the model structures that performed the worst were the the two currently most popular model structures: the fifth order Markov chain model (mc5) and the three-periodic inhomogeneous Markov chain model (i3pmc). With a single exception (mm), ADPH length modeling has an averse effect on the prediction performance, whereas using the three-state versions dramatically improved the performance of all models. Even though the difference in prediction performance for the standard models is less pronounced for the *B.subtilis* experiments, the general ranking of the models are still the same as for the *E.coli* experiments.

**Table 3.** *P*-values from pairwise paired *t*-tests of the ranking of the standard models of the AUC values of the ROC curves from the *E.coli* verified ecocyc cross-validation experiments

| > | aa_ s/3/a | eco_ s/3/a | i3pmc_ s/3/a | mc5_ s/3/a | mm_ s/3/a |
|---|---|---|---|---|---|
| aa | | 2.048e-05 | 9.768e-06 | 2.095e-05 | 1.676e-05 |
| eco | 1 | | 1.848e-06 | 3.612e-05 | 1.393e-05 |
| i3pmc | 1 | 1 | | 4.526e-05 | 1 |
| mc5 | 1 | 1 | 1 | | 1 |
| mm | 1 | 1 | 1.586e-07 | 4.322e-05 | |
| aa_3 | | 0.0727 | 0.0001077 | 1.857e-06 | 0.005065 |
| eco_3 | 0.9273 | | 0.005022 | 2.35e-05 | 0.2019 |
| i3pmc_3 | 0.9999 | 0.995 | | 1.1e-05 | 0.9887 |
| mc5_3 | 1 | 1 | 1 | | 1 |
| mm_3 | 0.995 | 0.7981 | 0.01135 | 2.120e-06 | |
| aa_a | | 0.0001479 | 1.495e-05 | 3.621e-06 | 0.0001170 |
| eco_a | 0.9999 | | 2.171e-06 | 3.826e-06 | 0.9988 |
| i3pmc_a | 1 | 1 | | 4.879e-06 | 1 |
| mc5_a | 1 | 1 | 1 | | 1 |
| mm_a | 0.9999 | 0.001164 | 6.347e-06 | 4.17e-06 | |

For each of the model versions, values are reported for comparisons within class only. Alternative hypothesis is that row entries are greater than column entries.

**Table 4.** *P*-values from pairwise paired *t*-tests of the ranking of the three-state, standard and ADPH models of the AUC values from the *E.coli* verified ecocyc cross-validation experiments

| | aa | eco | i3pmc | mc5 | mm |
|---|---|---|---|---|---|
| 3> *s* | 1.276e-05 | 0.001599 | 0.0004739 | 0.02713 | 0.0007062 |
| s> *a* | 0.01031 | 0.04478 | 0.03815 | 0.005672 | 0.5935 |

## 6 DISCUSSION

The findings of this study have been based on very architecturally simple models of protein coding potential only. We have used bacterial gene finding as a test environment favoring a thorough comparative test of the most prominent current single sequence bacterial gene finder model structures. Our results are not directly comparable to prediction performances of current state of the art gene finders in absolute terms, since we do not employ the heuristics that have been carefully developed through the last decades. However, as more and more data amasses we need as powerful models as possible. We propose that systematic testing of underlying model structures in a language where the model is foregrounded is a valuable approach to constructing reliable models for real-world applications.

## 7 CONCLUSION

We have developed and tested a number of both new and existing gene finder architectures for modeling protein coding potential in

**Table 5.** Prediction performances of the *Bacillus* cross-validation experiments

| Model | TPR | FPR | AUC |
|---|---|---|---|
| aa | $0.955 \pm 0.002$ | $0.056 \pm 0.0$ | $0.973 \pm 0.001$ |
| eco | $0.951 \pm 0.006$ | $0.053 \pm 0.0$ | $0.973 \pm 0.001$ |
| i3pmc | $0.953 \pm 0.002$ | $0.055 \pm 0.0$ | $0.972 \pm 0.001$ |
| mc5 | $0.804 \pm 0.006$ | $0.087 \pm 0.0$ | $0.884 \pm 0.004$ |
| mm | $0.955 \pm 0.004$ | $0.055 \pm 0.0$ | $0.973 \pm 0.001$ |
| aa_3 | $0.963 \pm 0.004$ | $0.052 \pm 0.0$ | $0.976 \pm 0.001$ |
| eco_3 | $0.959 \pm 0.002$ | $0.051 \pm 0.0$ | $0.975 \pm 0.001$ |
| i3pmc_3 | $0.953 \pm 0.005$ | $0.063 \pm 0.0$ | $0.971 \pm 0.001$ |
| mc5_3 | $0.804 \pm 0.005$ | $0.087 \pm 0.0$ | $0.884 \pm 0.003$ |
| mm_3 | $0.957 \pm 0.001$ | $0.051 \pm 0.0$ | $0.975 \pm 0.001$ |
| aa_a | $0.951 \pm 0.003$ | $0.054 \pm 0.0$ | $0.972 \pm 0.001$ |
| eco_a | $0.944 \pm 0.004$ | $0.048 \pm 0.0$ | $0.972 \pm 0.001$ |
| i3pmc_a | $0.949 \pm 0.003$ | $0.052 \pm 0.0$ | $0.972 \pm 0.001$ |
| mc5_a | $0.8 \pm 0.009$ | $0.091 \pm 0.0$ | $0.877 \pm 0.004$ |
| mm_a | $0.946 \pm 0.004$ | $0.049 \pm 0.0$ | $0.972 \pm 0.001$ |

Mean values $\pm 1$ SD.

**Table 6.** *P*-values from pairwise paired *t*-tests of the ranking of the AUC values of the standard models from the *Bacillus* cross-validation experiments

| > | aa_ s/3/a | eco_ s/3/a | i3pmc_ s/3/a | mc5_ s/3/a | mm_ s/3/a |
|---|---|---|---|---|---|
| aa | | 0.0853 | 0.006177 | 2.187e-07 | 0.02411 |
| eco | 0.9147 | | 0.0001165 | 1.691e-07 | 0.0003565 |
| i3pmc | 0.9938 | 0.9999 | | 1.588e-07 | 1 |
| mc5 | 1 | 1 | 1 | | 1 |
| mm | 0.9759 | 0.9996 | 5.357e-05 | 1.641e-07 | |
| aa_3 | | 0.05268 | 5.516e-06 | 2.975e-07 | 2.137e-05 |
| eco_3 | 0.9473 | | 0.001716 | 1.603e-07 | 0.6412 |
| i3pmc_3 | 1 | 0.9983 | | 2.427e-07 | 1 |
| mc5_3 | 1 | 1 | 1 | | 1 |
| mm_3 | 1 | 0.3588 | 1.72e-05 | 3.23e-07 | |
| aa_a | | 0.2583 | 0.01180 | 2.349e-07 | 0.9564 |
| eco_a | 0.7417 | | 0.0001139 | 1.833e-07 | 1 |
| i3pmc_a | 0.9882 | 0.9999 | | 1.78e-07 | 1 |
| mc5_a | 1 | 1 | 1 | | 1 |
| mm_a | 0.04364 | 6.121e-05 | 7.598e-05 | 1.905e-07 | |

For each of the model versions, values are reported for comparisons within class only. Alternative hypothesis is that row entries are greater than column entries.

a generalized framework that permits direct comparisons of the performance of the underlying model structures. Our approach demonstrates that there are very efficient model structures hidden in the vast structure space of non-standard HMMs. Specifically, the novel structures presented here seems promising candidates for advancing gene finding and additional biological sequence analysis tasks that rely on protein coding potential. Additionally, the general approach that we have used for exploring HMM structures for capturing protein coding potential is a promising route to exploring and discovering efficient models used for more complex biological sequence analysis tasks (such as RNA structure prediction or modeling viral genomes). Lastly, we have shown that

**Table 7.** *P*-values from pairwise paired *t*-tests of the ranking of the three-state, standard and ADPH models of the AUC values from the *Bacillus* cross-validation experiments

| | aa | eco | i3pmc | mc5 | mm |
|---|---|---|---|---|---|
| $3 > s$ | 0.001636 | 0.01845 | 0.9704 | 0.8063 | 0.007597 |
| $s > a$ | 2.219e-05 | 5.076e-05 | 3.881e-05 | 1.553e-05 | 0.0001157 |

the probabilistic logic programming language, PRISM, is a capable framework for the rapid prototyping and benchmarking of statistical models in bioinformatics, up to datasets the size of small (bacterial) genomes.

*Funding*: ‘Logic-statistic modeling and analysis of biological sequence data’ funded by the NABIIT program under the Danish Strategic Research Council (to S.M.); NIH/NIGMS grant (R01-GM076705 to I.H.).

*Conflict of Interest*: none declared.

## REFERENCES

Besemer,J. and Borodovsky,M. (1999) Heuristic approach to deriving models for gene finding. *Nucleic Acids Res.*, **27**, 3911–3920.

Besemer,J. *et al.* (2001) GeneMarkS: a self-training method for prediction of gene starts in microbial genomes. Implications for finding sequence motifs in regulatory regions. *Nucleic Acids Res.*, **29**, 2607–2618.

Blattner,F.R. *et al.* (1997) The complete genome sequence of Escherichia coli K-12. *Science*, **277**, 1453–1462.

Bobbio,A. *et al.* (2003) Acyclic discrete phase type distributions: properties and a parameter estimation algorithm. *Perform. Eval.*, **54**, 1–32.

Borodovsky,M. and McInich,J. (1993) GENMARK: parallel gene recognition for both DNA strands. *Comput. Chem.*, **17**, 123.

Bradley,R.K. and Holmes,I. (2007) Transducers: an emerging probabilistic framework for modeling indels on trees. *Bioinformatics*, **23**, 3258–3262.

Burge,C. and Karlin,S. (1997) Prediction of complete gene structures in human genomic dna. *J. Mol. Biol.*, **268**, 78–94.

Christiansen,H. and Dahmcke,C.M. (2007) A machine learning approach to test data generation: a case study in evaluation of gene finders. In Perner,P. (ed.) *Machine Learning and Data Mining in Pattern Recognition*. Springer Verlag, Berlin, Germany. pp. 741–755.

Christiansen,H. *et al.* (2011) Taming the zoo of discrete HMM subspecies & some of their relatives. In Bel-Enquix,G. (eds) *Biology, Computation and Linguistics, New Interdisciplinary Paradigms*, vol. 228 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands. pp. 28–42.

Delcher,A.L. *et al.* (1999) Improved microbial gene identification with GLIMMER. *Nucleic Acids Res.*, **27**, 4636–4641.

DeLong,E.R. *et al.* (1988) Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, **44**, 837–845.

Durbin,R. *et al.* (1998) *Biological Sequence Analysis*. Cambridge University Press, Cambridge, UK.

Fickett,J.W. and Tung,C.-S. (1992) Assessment of protein coding measures. *Nucleic Acids Res.*, **20**, 6441–6450.

Ghahramani,Z. and Jordan,M.I. (1996) Factorial hidden Markov models. *Mach. Learn.*, **29**, 245–273.

Henderson,J. *et al.* (1997) Finding genes in DNA with a Hidden Markov Model. *J. Comp. Biol.*, **4**, 127–141.

Katahira,K. *et al.* (2008) Deterministic annealing variant of variational Bayes method. *J. Phys. Conf.*, **95**, 012015.

Keseler,I.M. *et al.* (2009) EcoCyc: a comprehensive view of Escherichia coli biology. *Nucleic Acids Res.*, **37** (Suppl. 1), D464–D470.

Korf,I. (2004) Gene finding in novel genomes. *BMC Bioinformatics*, **5**, 59.

Krogh,A. (1997) Two methods for improving performance of an hmm and their application for gene finding. *Proc. Int. Conf. Intell. Syst. Mol. Biol.*, **5**, 179–186.

Krogh,A. *et al.* (1994a) A hidden Markov model that finds genes in *E.coli* DNA. *Nucleic Acids Res.*, **22**, 4768–4778.

Krogh,A. *et al.* (1994b) Hidden Markov Models in computational biology : applications to protein modeling. *J. Mol. Biol.*, **235**, 1501–1531.

Kulp,D. *et al.* (1996) A generalized hidden markov model for the recognition of human genes in dna. In *Proceedings of the Fourth International Conference on Intelligent System for Molecular Biology*. AAAI Press, Menlo Park, CA, USA.

Kunst,F. *et al.* (1997) The complete genome sequence of the gram-positive bacterium bacillus subtilis. *Nature*, **390**, 249–256.

Larsen,T. and Krogh,A. (2003) Easygene - a prokaryotic gene finder that ranks orfs by statistical significance. *BMC Bioinformatics*, **4**, 21.

Lomsadze,A. *et al.* (2005) Gene identification in novel eukaryotic genomes by self-training algorithm. *Nucleic Acids Res.*, **33**, 6494–6506.

Lukashin,A. and Borodovsky,M. (1998) GeneMark.hmm: new solutions for gene finding. *Nucleic Acids Res.*, **26**, 1107–1115.

Majoros,W.H. *et al.* (2003) GlimmerM, Exonomy and Unveil: three ab initio eukaryotic genefinders. *Nucleic Acids Res.*, **31**, 3601–3604.

Majoros,W.H. *et al.* (2004) TigrScan and GlimmerHMM: two open source ab initio eukaryotic gene-finders. *Bioinformatics*, **20**, 2878–2879.

Munch,K. and Krogh,A. (2006) Automatic generation of gene finders for eukaryotic species. *BMC Bioinformatics*, **7**, 263.

Rabiner,L.R. (1989) A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, **77**, 257–286.

Reese,M.G. *et al.* (2000) Genie, gene finding in Drosophila melanogaster. *Genome Res.*, **10**, 529–538.

Salzberg,S.L. *et al.* (1998) Microbial gene identification using interpolated Markov models. *Nucleic Acids Res.*, **26**, 544–548.

Sato,T. and Kameya,Y. (2001) Parameter learning of logic programs for symbolic-statistical modeling. *J. Artif. Intell. Res.*, **15**, 391–454.

Sato,T. *et al.* (2008) Variational Bayes via propositionalized probability computation in PRISM. *Ann. Math. Artif. Intell.*, **54**, 135–158.

Sato,T. *et al.* (2010). *PRISM User Manual (Version 2.0)*. at: http://sato-www.cs.titech.ac.jp/prism/download/prism20.pdf.

Sato,T. (2009) Generative modeling by PRISM. In Hill,P.M. and Warren,D.S. (eds) *ICLP*, vol. 5649 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany. pp. 24–35.

Saul,L.K. and Jordan,M.I. (1999) Mixed memory markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Mach. Learn.*, **37**, 75–87.

Searls,D.B. and Murphy,K.P. (1995) Automata-theoretic models of mutation and alignment. *Proc. Int. Conf. Intell. Syst. Mol. Biol.*, **3**, 341–349.

Shmatkov,A.M. *et al.* (1999) Finding prokaryotic genes by the frame-by-frame' algorithm: targeting gene starts and overlapping genes. *Bioinformatics*, **15**, 874–886.

Staden,R. and McLachian,A. (1982) Codon preference and its use in identifying protein coding regions in long DNA sequences. *Nucleic Acids Res.*, **10**, 141–156.

Staden,R. (1984) Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Res.*, **12**, 505–519.

Stormo,G.D. and Haussler,D. (1994) Optimally parsing a sequence into different classes based on multiple types of information. In *Proceedings of Second International Conference on Intelligent Systems for Molecular Biology*. AAAI/MIT Press, Menlo Park, CA, pp. 369–375.

Ueda,N. and Nakano,R. (1998) Deterministic annealing em algorithm. *Neural Netw.*, **11**, 271–282.

# Supplementary Materials

Evaluating Bacterial Gene-Finding HMM Structures as Probabilistic Logic Programs

Table 1: Size of the *E. coli* (E) and the *Bacillus* (B) cross-validation training set partitions. Seq are the number of sequences and nts. the total number of nucleotides in the datasets.

| partitions: | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| E seq: | 488 | 459 | 507 | 464 | 495 |
| E nts: | 515.289 | 457.527 | 525.822 | 472.692 | 523.548 |
| B seq: | 787 | 816 | 840 | 868 | 844 |
| B nts: | 732.015 | 723.471 | 700.881 | 798.621 | 740.139 |

Table 2: Confusion matrices based on cross-validation experiments using the *E. coli* verified ecocyc dataset as golden standard. Values reported are optimal in terms of ROC, corresponding to the intercept with the ROC curve parallel to the no-discrimination line. Mean values $\pm$ 1 standard deviation.

| Model | TP | FP | FN | TN |
|---|---|---|---|---|
| aa | $1861 \pm 14$ | $3006 \pm 217$ | $84 \pm 7$ | $63411 \pm 217$ |
| eco | $1860 \pm 15$ | $3152 \pm 70$ | $85 \pm 3$ | $63265 \pm 70$ |
| i3pmc | $1847 \pm 14$ | $2883 \pm 47$ | $97 \pm 2$ | $63534 \pm 47$ |
| mc5 | $1754 \pm 14$ | $4089 \pm 164$ | $190 \pm 4$ | $62328 \pm 164$ |
| mm | $1849 \pm 14$ | $2894 \pm 68$ | $95 \pm 2$ | $63523 \pm 68$ |
| | | | | |
| aa_3 | $1874 \pm 20$ | $2783 \pm 45$ | $53 \pm 4$ | $63634 \pm 45$ |
| eco_3 | $1871 \pm 20$ | $3026 \pm 106$ | $56 \pm 14$ | $63391 \pm 106$ |
| i3pmc_3 | $1849 \pm 13$ | $3156 \pm 151$ | $78 \pm 8$ | $63261 \pm 151$ |
| mc5_3 | $1734 \pm 18$ | $4239 \pm 244$ | $193 \pm 8$ | $62178 \pm 244$ |
| mm_3 | $1860 \pm 19$ | $3041 \pm 272$ | $67 \pm 5$ | $63376 \pm 272$ |
| | | | | |
| aa_a | $1838 \pm 20$ | $2948 \pm 153$ | $89 \pm 7$ | $63469 \pm 153$ |
| eco_a | $1833 \pm 20$ | $2892 \pm 282$ | $94 \pm 9$ | $63525 \pm 282$ |
| i3pmc_a | $1826 \pm 15$ | $2778 \pm 105$ | $101 \pm 6$ | $63639 \pm 105$ |
| mc5_a | $1727 \pm 22$ | $4242 \pm 369$ | $200 \pm 10$ | $62175 \pm 369$ |
| mm_a | $1838 \pm 21$ | $3014 \pm 231$ | $89 \pm 9$ | $63403 \pm 231$ |

Table 3: Prediction performances of the *E. coli* verified ecocyc cross-validation experiments. TPR and FPR values reported are optimal in terms of ROC, corresponding to the intercept with the ROC curve parallel to the no-discrimination line. Mean values $\pm$ 1 standard deviation.

| model | TPR | FPR | AUC | MC | MCC |
|---|---|---|---|---|---|
| aa | $0.957 \pm 0.004$ | $0.045 \pm 0.0$ | $0.978 \pm 0.001$ | $0.955 \pm 0.003$ | $0.59 \pm 0.013$ |
| eco | $0.956 \pm 0.002$ | $0.047 \pm 0.0$ | $0.976 \pm 0.001$ | $0.953 \pm 0.001$ | $0.58 \pm 0.003$ |
| i3pmc | $0.95 \pm 0.002$ | $0.043 \pm 0.0$ | $0.975 \pm 0.001$ | $0.956 \pm 0.001$ | $0.594 \pm 0.002$ |
| mc5 | $0.902 \pm 0.002$ | $0.062 \pm 0.0$ | $0.951 \pm 0.003$ | $0.937 \pm 0.002$ | $0.5 \pm 0.009$ |
| mm | $0.951 \pm 0.001$ | $0.044 \pm 0.0$ | $0.976 \pm 0.001$ | $0.956 \pm 0.001$ | $0.594 \pm 0.005$ |
| | | | | | |
| aa_3 | $0.972 \pm 0.002$ | $0.042 \pm 0.0$ | $0.987 \pm 0.0$ | $0.959 \pm 0.001$ | $0.611 \pm 0.003$ |
| eco_3 | $0.971 \pm 0.007$ | $0.046 \pm 0.0$ | $0.985 \pm 0.002$ | $0.955 \pm 0.001$ | $0.594 \pm 0.007$ |
| i3pmc_3 | $0.959 \pm 0.004$ | $0.048 \pm 0.0$ | $0.979 \pm 0.001$ | $0.953 \pm 0.002$ | $0.58 \pm 0.009$ |
| mc5_3 | $0.9 \pm 0.004$ | $0.064 \pm 0.0$ | $0.952 \pm 0.002$ | $0.935 \pm 0.003$ | $0.49 \pm 0.01$ |
| mm_3 | $0.965 \pm 0.003$ | $0.046 \pm 0.0$ | $0.983 \pm 0.002$ | $0.955 \pm 0.004$ | $0.591 \pm 0.019$ |
| | | | | | |
| aa_a | $0.954 \pm 0.003$ | $0.044 \pm 0.0$ | $0.977 \pm 0.001$ | $0.956 \pm 0.002$ | $0.59 \pm 0.008$ |
| eco_a | $0.951 \pm 0.005$ | $0.044 \pm 0.0$ | $0.975 \pm 0.001$ | $0.956 \pm 0.004$ | $0.593 \pm 0.017$ |
| i3pmc_a | $0.947 \pm 0.003$ | $0.042 \pm 0.0$ | $0.974 \pm 0.001$ | $0.958 \pm 0.001$ | $0.598 \pm 0.007$ |
| mc5_a | $0.896 \pm 0.005$ | $0.064 \pm 0.0$ | $0.947 \pm 0.002$ | $0.935 \pm 0.005$ | $0.489 \pm 0.014$ |
| mm_a | $0.954 \pm 0.005$ | $0.045 \pm 0.0$ | $0.976 \pm 0.001$ | $0.955 \pm 0.003$ | $0.586 \pm 0.012$ |

Table 4: Confusion matrices based on the *Bacillus* cross-validation experiments. Values reported are optimal in terms of ROC, corresponding to the intercept with the ROC curve parallel to the no-discrimination line. Mean values $\pm$ 1 standard deviation.

| Model | TP | FP | FN | TN |
|---|---|---|---|---|
| aa | $1666 \pm 13$ | $3409 \pm 97$ | $79 \pm 3$ | $57608 \pm 97$ |
| eco | $1659 \pm 16$ | $3233 \pm 336$ | $86 \pm 11$ | $57784 \pm 336$ |
| i3pmc | $1663 \pm 15$ | $3338 \pm 240$ | $82 \pm 4$ | $57679 \pm 240$ |
| mc5 | $1403 \pm 11$ | $5319 \pm 146$ | $341 \pm 12$ | $55698 \pm 146$ |
| mm | $1666 \pm 18$ | $3376 \pm 188$ | $79 \pm 7$ | $57641 \pm 188$ |
| | | | | |
| aa_3 | $1683 \pm 19$ | $3167 \pm 138$ | $64 \pm 6$ | $57850 \pm 138$ |
| eco_3 | $1675 \pm 13$ | $3099 \pm 232$ | $72 \pm 3$ | $57918 \pm 232$ |
| i3pmc_3 | $1664 \pm 11$ | $3833 \pm 264$ | $83 \pm 9$ | $57184 \pm 264$ |
| mc5_3 | $1405 \pm 13$ | $5324 \pm 147$ | $342 \pm 10$ | $55693 \pm 147$ |
| mm_3 | $1673 \pm 13$ | $3093 \pm 103$ | $74 \pm 2$ | $57924 \pm 103$ |
| | | | | |
| aa_a | $1659 \pm 15$ | $3295 \pm 114$ | $86 \pm 4$ | $57722 \pm 114$ |
| eco_a | $1648 \pm 8$ | $2930 \pm 273$ | $97 \pm 8$ | $58087 \pm 273$ |
| i3pmc_a | $1655 \pm 13$ | $3197 \pm 135$ | $90 \pm 6$ | $57820 \pm 135$ |
| mc5_a | $1395 \pm 16$ | $5526 \pm 476$ | $350 \pm 17$ | $55491 \pm 476$ |
| mm_a | $1650 \pm 6$ | $2983 \pm 253$ | $94 \pm 7$ | $58034 \pm 253$ |

Table 5: Prediction performances of the *Bacillus* cross-validation experiments. TPR and FPR values reported are optimal in terms of ROC, corresponding to the intercept with the ROC curve parallel to the no-discrimination line. Mean values ± 1 standard deviation.

| model | TPR | FPR | AUC | MC | MCC |
|---|---|---|---|---|---|
| aa | 0.955 ± 0.002 | 0.056 ± 0.0 | 0.973 ± 0.001 | 0.944 ± 0.002 | 0.542 ± 0.007 |
| eco | 0.951 ± 0.006 | 0.053 ± 0.0 | 0.973 ± 0.001 | 0.947 ± 0.005 | 0.552 ± 0.018 |
| i3pmc | 0.953 ± 0.002 | 0.055 ± 0.0 | 0.972 ± 0.001 | 0.946 ± 0.004 | 0.546 ± 0.013 |
| mc5 | 0.804 ± 0.006 | 0.087 ± 0.0 | 0.884 ± 0.004 | 0.91 ± 0.002 | 0.381 ± 0.004 |
| mm | 0.955 ± 0.004 | 0.055 ± 0.0 | 0.973 ± 0.001 | 0.945 ± 0.003 | 0.544 ± 0.009 |
| | | | | | |
| aa_3 | 0.963 ± 0.004 | 0.052 ± 0.0 | 0.976 ± 0.001 | 0.949 ± 0.002 | 0.562 ± 0.006 |
| eco_3 | 0.959 ± 0.002 | 0.051 ± 0.0 | 0.975 ± 0.001 | 0.949 ± 0.004 | 0.564 ± 0.015 |
| i3pmc_3 | 0.953 ± 0.005 | 0.063 ± 0.0 | 0.971 ± 0.001 | 0.938 ± 0.004 | 0.518 ± 0.012 |
| mc5_3 | 0.804 ± 0.005 | 0.087 ± 0.0 | 0.884 ± 0.003 | 0.91 ± 0.002 | 0.381 ± 0.004 |
| mm_3 | 0.957 ± 0.001 | 0.051 ± 0.0 | 0.975 ± 0.001 | 0.95 ± 0.002 | 0.563 ± 0.007 |
| | | | | | |
| aa_a | 0.951 ± 0.003 | 0.054 ± 0.0 | 0.972 ± 0.001 | 0.946 ± 0.002 | 0.547 ± 0.007 |
| eco_a | 0.944 ± 0.004 | 0.048 ± 0.0 | 0.972 ± 0.001 | 0.952 ± 0.004 | 0.567 ± 0.017 |
| i3pmc_a | 0.949 ± 0.003 | 0.052 ± 0.0 | 0.972 ± 0.001 | 0.948 ± 0.002 | 0.552 ± 0.008 |
| mc5_a | 0.8 ± 0.009 | 0.091 ± 0.0 | 0.877 ± 0.004 | 0.906 ± 0.007 | 0.373 ± 0.012 |
| mm_a | 0.946 ± 0.004 | 0.049 ± 0.0 | 0.972 ± 0.001 | 0.951 ± 0.004 | 0.565 ± 0.016 |

Table 6: P-values from pairwise paired t-tests of the ranking of the standard models of the AUC values of the ROC curves from the *E. coli* verified ecocyc cross-validation experiments. For each of the model versions values are reported for comparisons within class only. Alternative hypothesis is that row entries are greater than column entries.

| > | aa_ s/3/a | eco_ s/3/a | i3pmc_ s/3/a | mc5_ s/3/a | mm_ s/3/a |
|---|---|---|---|---|---|
| aa | NA | 2.048e-05 | 9.768e-06 | 2.095e-05 | 1.676e-05 |
| eco | 1 | NA | 1.848e-06 | 3.612e-05 | 1.393e-05 |
| i3pmc | 1 | 1 | NA | 4.526e-05 | 1 |
| mc5 | 1 | 1 | 1 | NA | 1 |
| mm | 1 | 1 | 1.586e-07 | 4.322e-05 | NA |
| | | | | | |
| aa_3 | NA | 0.0727 | 0.0001077 | 1.857e-06 | 0.005065 |
| eco_3 | 0.9273 | NA | 0.005022 | 2.35e-05 | 0.2019 |
| i3pmc_3 | 0.9999 | 0.995 | NA | 1.1e-05 | 0.9887 |
| mc5_3 | 1 | 1 | 1 | NA | 1 |
| mm_3 | 0.995 | 0.7981 | 0.01135 | 2.120e-06 | NA |
| | | | | | |
| aa_a | NA | 0.0001479 | 1.495e-05 | 3.621e-06 | 0.0001170 |
| eco_a | 0.9999 | NA | 2.171e-06 | 3.826e-06 | 0.9988 |
| i3pmc_a | 1 | 1 | NA | 4.879e-06 | 1 |
| mc5_a | 1 | 1 | 1 | NA | 1 |
| mm_a | 0.9999 | 0.001164 | 6.347e-06 | 4.17e-06 | NA |

Table 7: P-values from pairwise paired t-tests of the ranking of the 3-state, standard and ADPH models of the AUC values from the *E. coli* verified ecocyc cross-validation experiments.

| | aa | eco | i3pmc | mc5 | mm |
|---|---|---|---|---|---|
| 3> *s* | 1.276e-05 | 0.001599 | 0.0004739 | 0.02713 | 0.0007062 |
| s> *a* | 0.01031 | 0.04478 | 0.03815 | 0.005672 | 0.5935 |

Table 8: P-values from pairwise paired t-tests of the ranking of the AUC values of the standard models from the *Bacillus* cross-validation experiments. For each of the model versions values are reported for comparisons within class only. Alternative hypothesis is that row entries are greater than column entries.

| > | aa_ s/3/a | eco_ s/3/a | i3pmc_ s/3/a | mc5_ s/3/a | mm_ s/3/a |
|---|---|---|---|---|---|
| aa | NA | 0.0853 | 0.006177 | 2.187e-07 | 0.02411 |
| eco | 0.9147 | NA | 0.0001165 | 1.691e-07 | 0.0003565 |
| i3pmc | 0.9938 | 0.9999 | NA | 1.588e-07 | 1 |
| mc5 | 1 | 1 | 1 | NA | 1 |
| mm | 0.9759 | 0.9996 | 5.357e-05 | 1.641e-07 | NA |
| | | | | | |
| aa_3 | NA | 0.05268 | 5.516e-06 | 2.975e-07 | 2.137e-05 |
| eco_3 | 0.9473 | NA | 0.001716 | 1.603e-07 | 0.6412 |
| i3pmc_3 | 1 | 0.9983 | NA | 2.427e-07 | 1 |
| mc5_3 | 1 | 1 | 1 | NA | 1 |
| mm_3 | 1 | 0.3588 | 1.72e-05 | 3.23e-07 | NA |
| | | | | | |
| aa_a | NA | 0.2583 | 0.01180 | 2.349e-07 | 0.9564 |
| eco_a | 0.7417 | NA | 0.0001139 | 1.833e-07 | 1 |
| i3pmc_a | 0.9882 | 0.9999 | NA | 1.78e-07 | 1 |
| mc5_a | 1 | 1 | 1 | NA | 1 |
| mm_a | 0.04364 | 6.121e-05 | 7.598e-05 | 1.905e-07 | NA |

Table 9: P-values from pairwise paired t-tests of the ranking of the 3-state, standard and ADPH models of the AUC values from the *Bacillus* cross-validation experiments.

| | aa | eco | i3pmc | mc5 | mm |
|---|---|---|---|---|---|
| 3> *s* | 0.001636 | 0.01845 | 0.9704 | 0.8063 | 0.007597 |
| s> *a* | 2.219e-05 | 5.076e-05 | 3.881e-05 | 1.553e-05 | 0.0001157 |

Table 10: P-values from pairwise paired t-tests of the ranking of the Variational Free Energy values from the *E. coli* cross-validation experiments. For each of the model versions values are reported for comparisons within class only. Alternative hypothesis is that row entries are greater than column entries.

| > | aa_ s/3/a | eco_ s/3/a | i3pmc_ s/3/a | mc5_ s/3/a | mm_ s/3/a |
|---|---|---|---|---|---|
| aa_s | NA | 1.988e-06 | 1.302e-06 | 7.629e-07 | 2.166e-06 |
| eco_s | 1 | NA | 5.119e-07 | 7.48e-07 | 1 |
| i3pmc_s | 1 | 1 | NA | 8.082e-07 | 1 |
| mc5_s | 1 | 1 | 1 | NA | 1 |
| mm_s | 1 | 2.839e-06 | 1.102e-06 | 8.122e-07 | NA |
| | | | | | |
| aa_3 | NA | 0.07047 | 1.674e-07 | 1.157e-07 | 2.756e-05 |
| eco_3 | 0.9295 | NA | 0.2624 | 0.001820 | 0.7868 |
| i3pmc_3 | 1 | 0.7376 | NA | 2.730e-07 | 1 |
| mc5_3 | 1 | 0.9982 | 1 | NA | 1 |
| mm_3 | 1 | 0.2132 | 7.014e-06 | 3.695e-07 | NA |
| | | | | | |
| aa_a | NA | 0.03175 | 1.136e-06 | 2.936e-07 | 6.534e-05 |
| eco_a | 0.9682 | NA | 0.5598 | 0.002098 | 0.9546 |
| i3pmc_a | 1 | 0.4402 | NA | 1.672e-07 | 1 |
| mc5_a | 1 | 0.998 | 1 | NA | 1 |
| mm_a | 1 | 0.04538 | 2.709e-07 | 1.737e-07 | NA |

Table 11: P-values from pairwise paired t-tests of the ranking of the Variational Free Energy values from the *Bacillus* cross-validation experiments. For each of the model versions values are reported for comparisons within class only. Alternative hypothesis is that row entries are greater than column entries.

| > | aa_ s/3/a | eco_ s/3/a | i3pmc_ s/3/a | mc5_ s/3/a | mm_ s/3/a |
|---|---|---|---|---|---|
| aa_s | NA | 3.487e-07 | 2.555e-07 | 1.617e-07 | 3.416e-07 |
| eco_s | 1 | NA | 1.340e-07 | 2.147e-07 | 1 |
| i3pmc_s | 1 | 1 | NA | 3.731e-07 | 1 |
| mc5_s | 1 | 1 | 1 | NA | 1 |
| mm_s | 1 | 6.151e-07 | 2.697e-07 | 1.965e-07 | NA |
| | | | | | |
| aa_3 | NA | 1.161e-07 | 2.984e-07 | 9.715e-09 | 4.856e-07 |
| eco_3 | 1 | NA | 1.062e-06 | 1.922e-08 | 1 |
| i3pmc_3 | 1 | 1 | NA | 3.281e-07 | 1 |
| mc5_3 | 1 | 1 | 1 | NA | 1 |
| mm_3 | 1 | 3.345e-05 | 4.948e-07 | 4.477e-08 | NA |
| | | | | | |
| aa_a | NA | 3.821e-07 | 2.596e-07 | 1.696e-07 | 4.806e-06 |
| eco_a | 1 | NA | 1.476e-07 | 1.976e-07 | 1 |
| i3pmc_a | 1 | 1 | NA | 3.734e-07 | 1 |
| mc5_a | 1 | 1 | 1 | NA | 1 |
| mm_a | 1 | 1.777e-07 | 1.612e-07 | 1.342e-07 | NA |

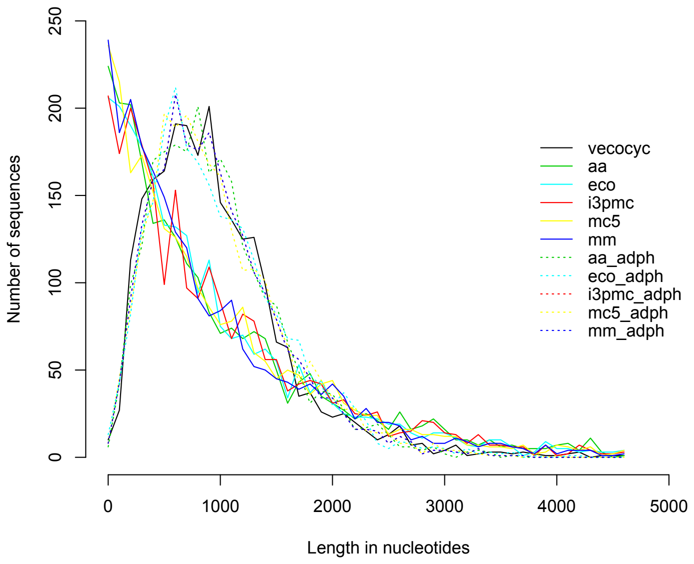


Figure 1: Length distribution plots of the *E. coli* verified ecocyc dataset, and 2413 sampled sequences from standard and ADPH version models trained on the *E. coli* verified ecocyc dataset.

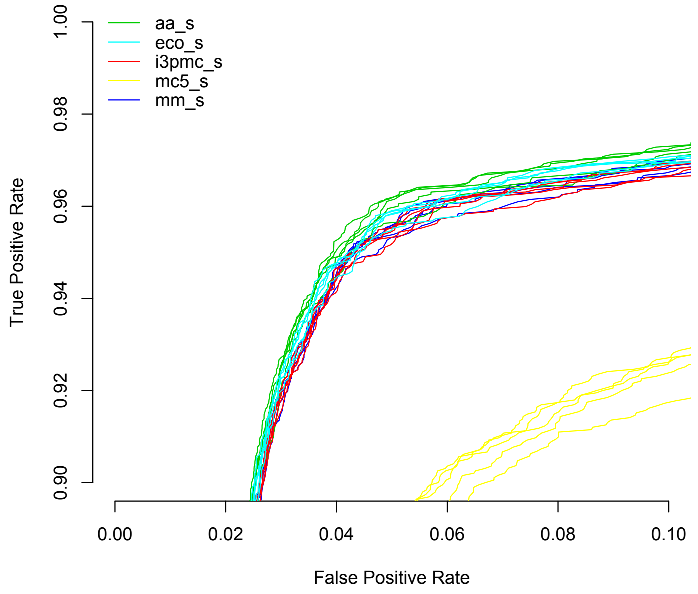


Figure 2: Cross-validation Receiver Operator Characteristics (ROC) curves for the *E. coli* standard models using thresholds over all log odds values in the positive set. Notice that only the area from 0.0-0.1 FPR and 0.9-1.0 TPR is shown.

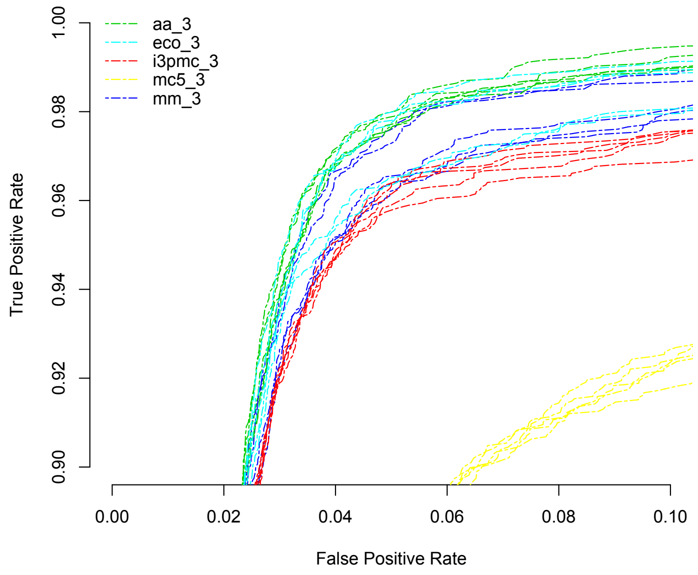


Figure 3: Cross-validation Receiver Operator Characteristics (ROC) curves for the *E. coli* 3-state models using thresholds over all log odds values in the positive set. Notice that only the area from 0.0-0.1 FPR and 0.9-1.0 TPR is shown.

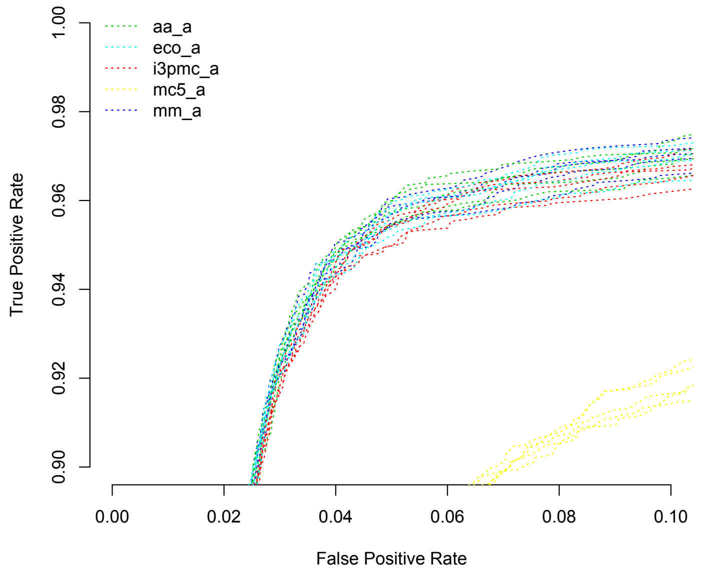


Figure 4: Cross-validation Receiver Operator Characteristics (ROC) curves for the *E. coli* ADPH models using thresholds over all log odds values in the positive set. Notice that only the area from 0.0-0.1 FPR and 0.9-1.0 TPR is shown.

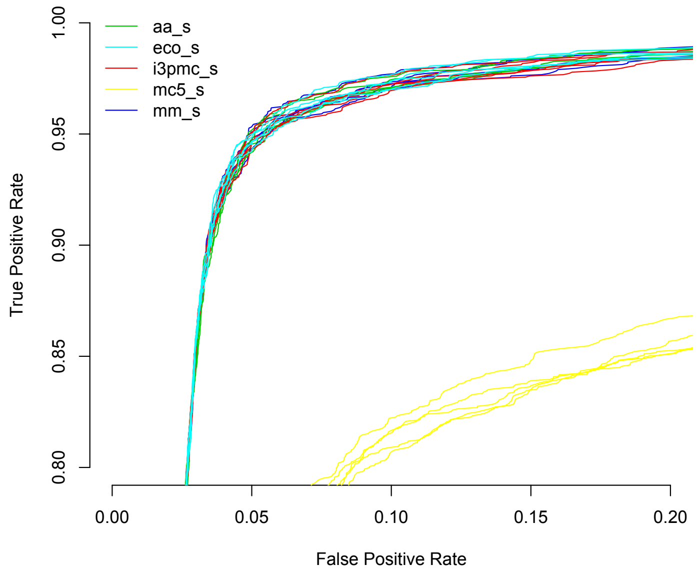


Figure 5: Cross-validation Receiver Operator Characteristics (ROC) curves for the *Bacillus* standard models using thresholds over all log odds values in the positive set. Notice that only the area from 0.0-0.1 FPR and 0.9-1.0 TPR is shown.

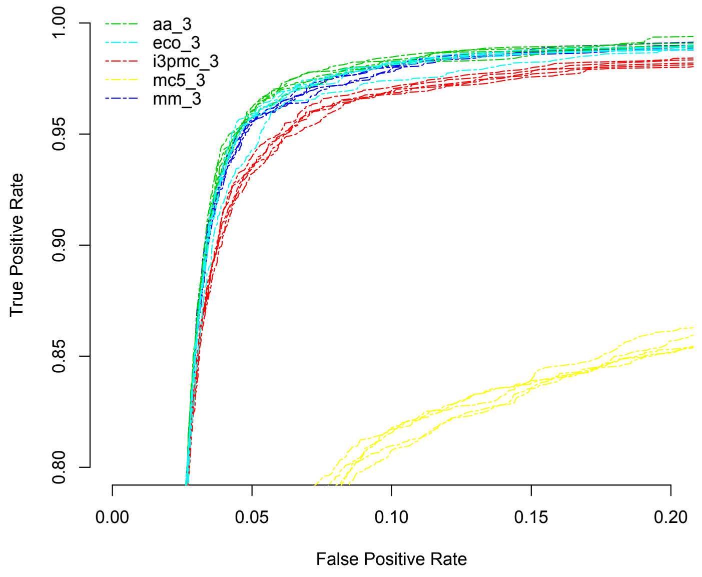


Figure 6: Cross-validation Receiver Operator Characteristics (ROC) curves for the *Bacillus* 3-state models using thresholds over all log odds values in the positive set. Notice that only the area from 0.0-0.1 FPR and 0.9-1.0 TPR is shown.

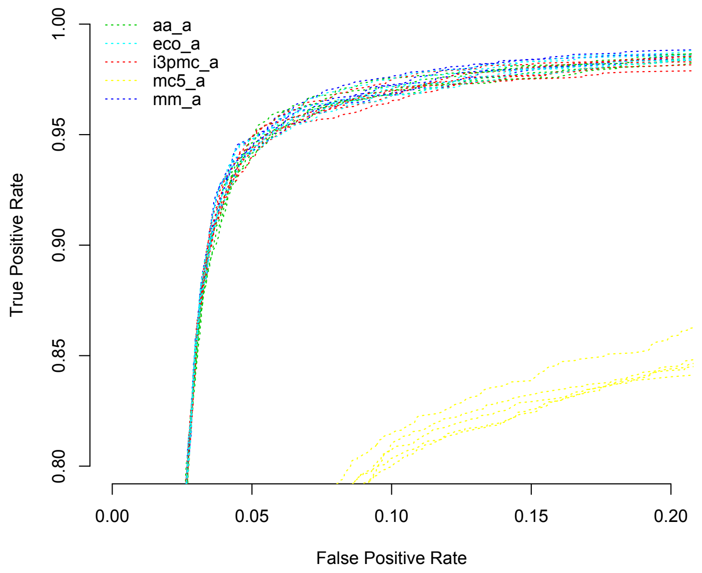


Figure 7: Cross-validation Receiver Operator Characteristics (ROC) curves for the *Bacillus* ADPH models using thresholds over all log odds values in the positive set. Notice that only the area from 0.0-0.1 FPR and 0.9-1.0 TPR is shown.

# Paper II

# A Probabilistic Genome-Wide Reading Frame Sequence Model

Christian Theil Have[1] and Søren Mørk[2*]

[1]Department of Communication, Busines and Information Technologies, Roskilde University

[2]Department of Science, Systems and Models, Roskilde University

* joint first authorship

**Publication Status:**

*In Review - PLOS One*

# A Probabilistic Genome-Wide Reading Frame Sequence Model

Christian Theil Have 1[1,*], Søren Mørk[2,*]

**1 Department of Communication, Business and Information Technologies, Roskilde University, 4000 Roskilde, Denmark**

**2 Department of Science, Systems and Models, Roskilde University, 4000 Roskilde, Denmark**

∗ **E-mail: cth@ruc.dk, soer@ruc.dk**

## Abstract

We introduce a new type of probabilistic sequence model, that model the sequential composition of reading frames of genes in a genome. Our approach extends gene finders with a model of the sequential composition of genes at the genome-level – effectively producing a sequential genome annotation as output. The model can be used to obtain the most probable genome annotation based on a combination of i: a gene finder score of each gene candidate and ii: the sequence of the reading frames of gene candidates through a genome. The model — as well as a higher order variant — is developed and tested using the probabilistic logic programming language and machine learning system PRISM - a fast and efficient model prototyping environment, using bacterial gene finding performance as a benchmark of signal strength. The model is used to prune a set of gene predictions from an underlying gene finder and are evaluated by the effect on prediction performance. Since bacterial gene finding to a large extent is a solved problem it forms an ideal proving ground for evaluating the explicit modeling of larger scale gene sequence composition of genomes.

We conclude that the sequential composition of gene reading frames is a consistent signal present in bacterial genomes, that can be effectively modeled with probabilistic sequence models.

## Introduction

Automated genome annotation is essential for exploiting the enormous amounts of genome sequence data currently being generated [1]. The initial steps of genome annotation relies heavily on probabilistic nucleotide sequence models, for generating sets of predicted genes. Such models typically estimate the probability that each open reading frame (ORF) is a gene. This estimate is usually based on only a limited context comprising the ORF nucleotide sequence and perhaps a few hundred bases upstream and downstream to include signals such as promoters and ribosomal binding sites. The subsequent steps to assemble a genome annotation typically involves selecting the highly scoring predictions using a significance criteria or threshold. In some recent gene finders [2–4] the selection of predictions is done as a genome-wide optimization where the predictions are chosen to form a coherent genome annotation by taking into account the extent of overlap between genes.

In a similar vein, we introduce a probabilistic sequence model which select the set of predictions that form the genome annotation, but which is based on sequential composition of gene reading frames, which we believe is a novel signal to be explored in gene finding. Our purpose is not to build the next state-of-the-art gene finder, but to present a class of simple models which clearly demonstrates the efficacy of exploiting the gene-reading-frame-sequence bias.

## The gene reading frame sequence bias

The existence of a gene-strand bias in prokaryotes is well established [5]. One source for this bias is a tendency for genes to be placed on the leading strand due to replication efficiency consequences of co-directional and head-on collisions of the replication and transcription apparatus [6]. It has also been argued that the preferential placement of genes in the leading strand is driven by essentiality rather than expression [7].

A gene-reading-frame-sequence bias is a general signal that can incorporate gene-strand bias, bias due to clusters of orthologous genes [8], operonic structures [9], phase preference for overlapping genes [10] and other potential effects yielding non-random sequence composition.

The gene-strand bias account for a large proportion of the gene-reading-frame-sequence bias, but a pronounced bias is detectable even within the strands. Furthermore, the gene-reading-frame-sequence bias seems to be symmetric for the two strands, cf. Table 1. This is a convenient property, especially considering the arbitrary designation of which is the forward and which is the reverse strand.

## Methods

Our gene-reading-frame-sequence model are implemented in PRISM, a probabilistic logic programming language and machine learning system with generic algorithms for parameter estimation and decoding [11]. We use PRISM as a convenient model comparison platform, since it is powerful enough to express the different models and enables a level execution provided by its generic machine learning routines. The use of probabilistic logic programming for evaluating sequence models as the heart of contemporary gene finders has recently been demonstrated in [12].

### The Frameseq model

The basic *Frameseq* model is a variant of a fully connected Hidden Markov Model (HMM) [13] with a state for each of the six possible reading frames — the *frame states* — and a delete state. Given a sequence of gene predictions sorted by position, a path through the model capable of emitting this prediction sequence represents a classification of predictions into presumed true positives emitted from the frame states and presumed false positives emitted from the delete state. A path with optimal probability represents a best hypothesis about the classification of predictions into positives and negatives. This path can be calculated using the Viterbi algorithm which is provided by PRISM.

Each state emits a score symbol and a frame for each gene prediction. Frame states only emit predictions with a corresponding frame, whereas the delete state may emit predictions of any frame. The score symbol is a symbolic value representing a range of confidence scores for the predictions of the input gene finder. The emission probabilities thus reflect the prediction confidence scores in the training set.

Traditionally, the transition probabilities of an HMM are conditioned only on the previous state (the Markov property). In our model the transition probability is conditioned on the previous *frame state* rather than just the previous state. The frame state transition probabilities are thus assumed to reflect the probability of a seeing a gene in a particular reading frame given the reading frame of the previous gene.

Higher ordered Markov models have generally shown to be an improvement over standard models for the nucleotide sequence models used in bacterial gene finding (*e.g.* as used in Genemark and Glimmer). To explore the possibility that the same might be true for the gene reading frame sequence, we have also employed a second order version of Frameseq, *i.e.*, which conditions transitions on the two previous frame states.

The transition probabilities between the *frame states* are estimated as the relative frequency of observed adjacent genes in the various frames observed in the set of verified genes.

The probability of a transition to the delete state (from any state) reflects the probability that a gene finder prediction is a false positive,

$$P(delete) = 1 - \frac{TP}{TP + FP}$$

where $TP$ is the number of true positives predicted by the gene finder and $FP$ is the number of false positives. This probability is directly related to gene finder specificity and may be tweaked for different sensitivity/specificity trade-offs. We exploit this in experiments reported below.

The frame state transition probabilities are estimated as relative frequencies, which have the interpretation of conditional probabilities given that a transition to the delete state did not occur. We normalize each of these transition probabilities by multiplying them by $1 - P(delete)$.

Each state is capable of emitting a finite set of $i$ symbols $\delta_1 \ldots \delta_n$ corresponding to ranges of prediction scores, *i.e.*n the states emit a discretized symbol corresponding to the confidence score of a prediction as supplied by the gene finder. The ranges are selected to ensure that each score symbol correspond to an equal proportion of gene finder predictions. The number of ranges, $n$, is a configurable parameter; when $n$ is high the model can better exploit the scores from the gene finder, but the estimated emission probabilities become more fragile, *i.e.*, more data is needed to reliably estimate them. The emission probabilities of the delete state are estimated as the relative frequency of each of the possible score symbols for all false positives predictions, *i.e.*,

$$P(\delta_i | state = delete) = \frac{FP_{\delta_i}}{FP}$$

where $FP_{\delta_i}$ is the number of false positives with a confidence score within the range symbolized by $\delta_i$.

Similarly, the emission probabilities of frame states are estimated as the fraction of true positive predictions belonging to a particular range within the corresponding frame, *i.e.*,

$$P(\delta_i | state = frame_j) = \frac{TP_{\delta_i}^{frame_j}}{TP^{frame_j}}$$

where $TP^{frame_j}$ is the total number true positive predictions in reading frame $j$ and $TP^{frame_j}_{\delta_i}$ is the number of true positive predictions in reading frame $j$ with a confidence score within the range symbolized by $\delta_i$.

A illustration of the states and transitions and of the model is shown in figure 1.

Instead of using exact empirical frequency counts as described above, we use a variational Bayes version of the EM algorithm [14] provided with PRISM. This algorithm puts Dirichlet priors (pseudo-counts) on random variables ensuring that all estimated probabilities are non-zero.

## Results and discussion

### The phylogenetic reach of the gene reading frame bias

To test the generalization capability and potential phylogenetic reach of our model, we train models on five different prokaryotic genomes and use them to filter predictions for the *E. coli* genome. We expect *E. coli* to have the most reliable genome annotation and by using it for validation we obtain the most reliable validation results. By training on distant organisms, we show the robustness of our approach with regard to both training set quality and phylogenetic distance. Good performance on *E. coli* should also imply that we can train our model on a well annotated genome and filter gene finder predictions in other genomes with increased reliability. To validate this we also do this experiment in reverse, *i.e.*, we also train our model on *E. coli* to predict on each of the other genomes. For all models trained, we set the number of score ranges to $n = 15$.

The five genomes, listed here in ascending order of phylogenetic distance from *E. coli*:

- *Escherichia coli* [REFSEQ:NC_000913],
- *Salmonella enterica* [REFSEQ:NC_004631.1],
- *Legionella pneumophila* [REFSEQ:NC_002942],
- *Bacillus subtilis* [REFSEQ:NC_000964]
- *Thermoplasma acidophilum* [REFSEQ:NC_002578].

We use Genemark 2.5 [15] which is available as a web-service to produce a large initial set of candidate genes.

Genemark is currently available in newer versions (GeneMarkS and GenemarkHMM) with improved prediction performance.

However, as our main interest is to introduce a new type of genome-sequence model, not to improve gene finding, the older version of Genemark provides a number of advantages for our purposes that are not present in other available single-sequence gene finders: Genemark 2.5 use a very simple scoring model and do not employ any post-scoring prediction selection algorithm, but is capable of producing a large set of predictions simply by enforcing a (low) score cut-off. As it does not otherwise prune predictions, we eliminate factors which could affect and reduce the pruning potential available to our model. Obviously, the accuracy of this gene finder is slightly below what is now state-of-the-art.

The full dataset offered by being able to produce a large set of predictions provides a better evaluation of the contribution of the reading frame signal than pruning a small optimal prediction set or (for completeness we do include such more limited experiments for state-of-the-art gene finders below).

We set the configurable score cut-off as low as possible, *i.e.*, to 0.1, to allow as many false positive predictions as possible. The gene finder predictions are preprocessed to contain only the best scoring prediction for each distinct stop codon. For each genome, we train using the preprocessed Genemark predictions and use the RefSeq annotation as golden standard. By inspection of transition probabilities, we observe that the gene-reading-frame-sequence bias tends to be almost symmetric for the strands, see tables 1-5.

We test the performance of each model on the Genemark predictions for the target genome by measuring sensitivity and specificity in terms of predicted stop codons with respect to the RefSeq annotation.

We repeat this process with incrementally increasing delete state probabilities resulting in a range of sensitivity/specificity trade-offs. These are plotted in Figure 2 (predictions on *E. coli*) and Figure 3 (predictions on the other genomes) as a Receiver Operator Characteristic (ROC) curves.

For comparison we provide a baseline ROC curve, produced via incrementally increasing a cut off value of the scores for the Genemark predictions for the target genome.

For all organisms except the phylogenetically very distant *T. acidophilum*, Frameseq improves accuracy and the margin of the improvement correlates with phylogenetic distance. The pronounced improvement in the accuracy which can be observed in ROC curves for the frame-bias model as compared to the baseline demonstrates that for comparable levels sensitivity, Frameseq achieves a lower false positive rate.

### A higher order signal?

It is plausible that the gene-reading-frame-sequence bias is more complex than just pairwise dependencies between the frames of genes. More complex dependencies on previous gene reading frames can be modeled using a higher order model.

We test this hypothesis by using a second order HMM based Frameseq model which is trained and applied on *E. coli.* We compare this to the basic Frameseq model which uses a first order HMM. We also investigate the phylogenic of conservation of a possible higher order signal by training the same model on *S. enterica* and decoding on *E. coli.* In both cases, the we use predictions from the Genemark 2.5 gene finder, with a score cut-off of 0.1. As in the previous experiments we set the number of score ranges to $n = 15$.

We derive and compare ROC curves for threshold selection and Frameseq selection like in the previous experiments, but here for both the first order and second order models. We provide a separate plot with the *E. coli* trained models (Figure 4) and the *S. enterica* trained models (Figure 5).

In the case where we train on *E. coli*, the second order Frameseq model results in significantly better accuracy than with the first order model. The improvement degrades quite a bit when we instead train the model on *S. enterica*, but there is still a detectable improvement in accuracy for the second order model.

It should be noted that, higher order models effectively increase the amount of transition probabilities involved, but the amount of training data used to estimate these are fixed in our case. This means that increasing the order of the model results in less reliable transition probabilities. This may explain some of the loss of accuracy when observed when training on *S. enterica* as compared to training on *E. coli.*

On the other hand, the experiment with the second order Frameseq model trained on *E. coli* demonstrates the maximal potential of utilizing a higher order signal.

### Effect on state-of-the-art gene finders

In this section we explore using Frameseq with Glimmer 3 and Prodigal 2.50 — to evaluate the contribution of a reading frame sequence signal for state-of-the art gene finders.

Genemark 2.5 which was used in the previous experiments, scores each open reading frame individually and does not attempt to stitch such individual predictions together into a more coherent set of predictions for the genome.

The algorithms employed by the two other gene finders have some similarities to the delete-HMM of Frameseq. Both gene finders use custom dynamic programming algorithms to achieve a more coherent set of predictions for a genome. Prodigal use several features including hexamer scores, ribosomal binding site detection, maximal overlap and distance between predictions combined using a custom dynamic programming algorithm. Similarly, Glimmer 3 also use a dynamic programming algorithm which restricts the size of overlaps between predictions. Neither of these algorithms utilize the gene frame bias.

Our algorithm is quite simplistic in comparison since it only considers one signal – the gene-reading-frame-sequence bias. It could undoubtedly be improved by considering other signals and constraints inherent between predictions such as distance and overlaps. In being simplistic, however, it clearly demonstrates the utility of the gene-reading-frame-sequence bias without the inherent noise from the impact of other features – which is our purpose.

In these experiments, we use Frameseq to filter the predictions of the state-of-the-art gene finders in order to explore the potential beneficial effect they could achieve by incorporating the gene-reading-frame-sequence bias. For each gene finder — Prodigal 2.50 and Glimmer 3 — we apply the second order Frameseq model trained on *E. coli* to filter their respective predictions on also on *E. coli.* The number of score ranges is in this experiment set to $n = 100$ to better capture the more detailed variations of the scores.

In all cases the filtered predictions have significantly improved specificity for comparable levels of sensitivity. The effect of Frameseq seems most pronounced with reduced sensitivities which could indicate that the scores of the gene finders are more reliable for the top-scoring predictions.

These experiments do not conclusively prove that all the gene finders could achieve improved specificity for the desired levels of sensitivity (close to one) by incorporating the gene-reading-frame-sequence bias. It should be noted that we slightly over-fit the model by training on *E. coli* and by doing this we get more impressive results than would have been the case if the models where trained using other organisms. Training Frameseq on, *e.g.*, *S. enterica* and filtering predictions for these gene finders does not result in significantly improved accuracy (data not shown). We believe this to be mainly a problem of sparsity of the training data, but also due to the reduced margins for possible improvement as compared to Genemark 2.5. We demonstrated the phylogenetic reach using Genemark 2.5, but the margin for possible improvement is significantly smaller with Glimmer and Prodigal. Due to this, a slightly under-fitted model will generalize sufficiently to improve Genemark 2.5 results, but insufficiently with the state-of-the art gene finders.

The experiment here, however, does show that the gene-reading-frame-sequence bias signal provides useful information which is complementary to the signals used by the contemporary methods.

## Conclusions

We have demonstrated the feasibility of modeling the sequential composition of genes in a genome with simple sequential reading frame models. We obtain surprisingly good results when predicting on one organism with models trained on phylogenetically distant genomes, which implies both the generality of the approach and the potential importance of gene reading frame sequence structure across taxa.

The impact of our method is most pronounced for reduced levels of sensitivity. Ideally we would like to achieve as significant improvements in specificity for a higher level of sensitivity, but improved specificity with a lower sensitivity is still a good result with important implications; It means that our approach is capable of supplying a larger set of gene predictions with a specified upper bound on the false positive rate, than is possible with any other gene finder. This may be useful when selecting candidate genes for experimental verification and can reduce the likelihood of wasted lab effort.

We also believe that the gene-reading-frame-sequence bias signal can be useful for improving automated computational genome annotation, but in order to achieve this, it will need to be integrated with the algorithms of state-of-the-art gene finders instead of the relatively superficial augmentation we do here.

In order to clearly illustrate the gene-reading-frame-sequence bias, we engineered our method to be as simple as possible, which in effect have several limitations:

- It relies on gene finder scores rather integrating with the algorithm of the gene finder, thereby missing out on exploiting possible correlations with signals incorporated in the gene finder.
- It relies on discretization of gene finder scores, *i.e.*, it summarizes of the information contained in prediction scores and hence cannot fully exploit these. The discretization procedure could be improved by using variable sized bins, *e.g.*, as in [16], or by instead using a continuous Hidden Markov Model.
- We do not fully exploit the nature of the gene finder score distribution for parameter smoothing.

We do apply limited parameter smoothing by using the variational Bayes EM algorithm, but we could probably achieve better generalization by fitting a suitable function to the gene finder score distribution.

- We use only a single organism as training data which become too sparse; This results in slightly over-fitted models when training on the same genome and slightly under-fitted models when training on an other genome. This situation could be amended by training on several genomes.

Despite these limitations, our method achieves good results which illustrate the usefulness of the signal, yet still leaves room for potential improvement.

We choose the problem of bacterial gene finding to exemplify the gene-reading-frame-sequence bias and its use. This problem has the nice property that it is almost solved, which enables us to use reference annotations to validate the approach. It should be noted, however, that many reference annotations are unverified results from the gene finders that we try to improve upon. This bias gives our method a slight disadvantage.

Lastly, we believe that the gene-reading-frame-sequence bias signal could have applications beyond gene finding. For instance, it may potentially benefit next generation sequencing and genome assembly where a complete model of the overall gene content of a genome would be applicable.

**Availability** The source code of the model and accessory scripts are freely available at:
http://github.com/frameseq/frameseq

## Acknowledgments

This work is part of the project "Logic-statistic modeling and analysis of biological sequence data" funded by the NABIIT program under the Danish Strategic Research Council.

## References

1. Brent MR (2008) Steady progress and recent breakthroughs in the accuracy of automated genome annotation. Nat Rev Genet 9: 62–73.

2. Delcher AL, Bratke KA, Powers EC, Salzberg SL (2007) Identifying bacterial genes and endosymbiont DNA with Glimmer. Bioinformatics 23: 673-679.

3. Hyatt D, Chen G, LoCascio P, Land M, Larimer F, et al. (2010) Prodigal: prokaryotic gene recognition and translation initiation site identification. BMC bioinformatics 11: 119.

4. Have CT (2011) Constraints and global optimization for gene prediction overlap resolution. In: Palu AD, Dovier A, Formisano A, editors, Proceedings of Workshop on Constraint Based Methods for Bioinformatics.

5. Brewer BJ (1988) When polymerases collide: replication and the transcriptional organization of the e. coli chromosome. Cell 53: 679–686.

6. Pomerantz RT, O'Donnell M (2008) The replisome uses mRNA as a primer after colliding with RNA polymerase. Nature 456: 762–766.

7. Rocha E, Danchin A, et al. (2003) Essentiality, not expressiveness, drives gene-strand bias in bacteria. Nature genetics 34: 377–378.

8. Rogozin I, Makarova K, Murvai J, Czabarka E, Wolf Y, et al. (2002) Connected gene neighborhoods in prokaryotic genomes. Nucleic Acids Research 30: 2212–2223.

9. Wolf Y, Rogozin I, Kondrashov A, Koonin E (2001) Genome alignment, evolution of prokaryotic genome organization, and prediction of gene function using genomic context. Genome research 11: 356–372.

10. Cock PJA, Whitworth DE (2010) Evolution of relative reading frame bias in unidirectional prokaryotic gene overlaps. Molecular Biology and Evolution 27: 753-756.

11. Sato T (2009) Generative modeling by PRISM. In: Hill PM, Warren DS, editors, ICLP. Springer, volume 5649 of *Lecture Notes in Computer Science*, pp. 24–35.

12. Mørk S, Holmes I (2012) Evaluating bacterial gene-finding hmm structures as probabilistic logic programs. Bioinformatics .

13. Rabiner LR (1989) A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In: Proceedings of the IEEE. volume 77.

14. Sato T, Kameya Y, Kurihara K (2008) Variational bayes via propositionalized probability computation in prism. Annals of Mathematics and Artificial Intelligence 54: 135–158.

15. Besemer J, Borodovsky M (2005) Genemark: web software for gene finding in prokaryotes, eukaryotes and viruses. Nucleic Acids Res 33: 451–454.

16. Kontkanen P, Myllymäki P (2007) MDL histogram density estimation. Journal of Machine Learning Research - Proceedings Track 2: 219–226.

## Tables and Figures

**Table 1. Gene reading frame transition probabilities *E. coli***

| from \ to | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0.18 | 0.20 | 0.28 | 0.13 | 0.1 | 0.1 |
| 2 | 0.29 | 0.2 | 0.2 | 0.12 | 0.09 | 0.1 |
| 3 | 0.22 | 0.3 | 0.17 | 0.1 | 0.11 | 0.1 |
| 4 | 0.11 | 0.1 | 0.1 | 0.19 | 0.23 | 0.27 |
| 5 | 0.11 | 0.9 | 0.1 | 0.29 | 0.19 | 0.22 |
| 6 | 0.09 | 0.08 | 0.1 | 0.23 | 0.30 | 0.19 |

A cell indicates the probability that a gene in the frame indicated by the row is followed by the gene in the frame indicated by the column. Note that the strands have almost symmetrical probabilities.

**Table 2. Gene reading frame transition probabilities *S. enterica***

| from \ to | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0.19 | 0.23 | 0.30 | 0.09 | 0.09 | 0.09 |
| 2 | 0.29 | 0.20 | 0.25 | 0.08 | 0.10 | 0.07 |
| 3 | 0.23 | 0.29 | 0.18 | 0.09 | 0.11 | 0.10 |
| 4 | 0.10 | 0.09 | 0.10 | 0.21 | 0.22 | 0.28 |
| 5 | 0.11 | 0.10 | 0.10 | 0.28 | 0.18 | 0.22 |
| 6 | 0.11 | 0.10 | 0.10 | 0.20 | 0.30 | 0.21 |

A cell indicates the probability that a gene in the frame indicated by the row is followed by the gene in the frame indicated by the column. Note that the strands have almost symmetrical probabilities.

**Table 3. Gene reading frame transition probabilities *L. pneumophila***

| from \ to | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0.18 | 0.23 | 0.26 | 0.10 | 0.11 | 0.13 |
| 2 | 0.27 | 0.18 | 0.22 | 0.12 | 0.09 | 0.12 |
| 3 | 0.24 | 0.27 | 0.18 | 0.10 | 0.12 | 0.10 |
| 4 | 0.09 | 0.11 | 0.11 | 0.18 | 0.20 | 0.31 |
| 5 | 0.12 | 0.11 | 0.11 | 0.26 | 0.17 | 0.23 |
| 6 | 0.09 | 0.09 | 0.11 | 0.24 | 0.29 | 0.18 |

*L. pneumophila*

A cell indicates the probability that a gene in the frame indicated by the row is followed by the gene in the frame indicated by the column. Note that the strands have almost symmetrical probabilities.

**Table 4. Gene reading frame transition probabilities *B. subtilis***

| from \ to | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0.22 | 0.24 | 0.25 | 0.09 | 0.10 | 0.10 |
| 2 | 0.29 | 0.20 | 0.23 | 0.11 | 0.08 | 0.09 |
| 3 | 0.25 | 0.25 | 0.22 | 0.08 | 0.10 | 0.11 |
| 4 | 0.09 | 0.10 | 0.06 | 0.23 | 0.24 | 0.28 |
| 5 | 0.11 | 0.09 | 0.09 | 0.26 | 0.22 | 0.23 |
| 6 | 0.10 | 0.07 | 0.08 | 0.26 | 0.28 | 0.21 |

*B. subtilis*

A cell indicates the probability that a gene in the frame indicated by the row is followed by the gene in the frame indicated by the column. Note that the strands have almost symmetrical probabilities.

**Table 5. Gene reading frame transition probabilities *T. acidophilum***

| from \ to | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0.18 | 0.20 | 0.23 | 0.17 | 0.09 | 0.13 |
| 2 | 0.25 | 0.16 | 0.24 | 0.13 | 0.13 | 0.09 |
| 3 | 0.22 | 0.26 | 0.21 | 0.11 | 0.10 | 0.10 |
| 4 | 0.12 | 0.14 | 0.12 | 0.18 | 0.23 | 0.20 |
| 5 | 0.15 | 0.16 | 0.06 | 0.24 | 0.18 | 0.20 |
| 6 | 0.10 | 0.11 | 0.15 | 0.20 | 0.22 | 0.21 |

A cell indicates the probability that a gene in the frame indicated by the row is followed by the gene in the frame indicated by the column. Note that the strands have almost symmetrical probabilities.

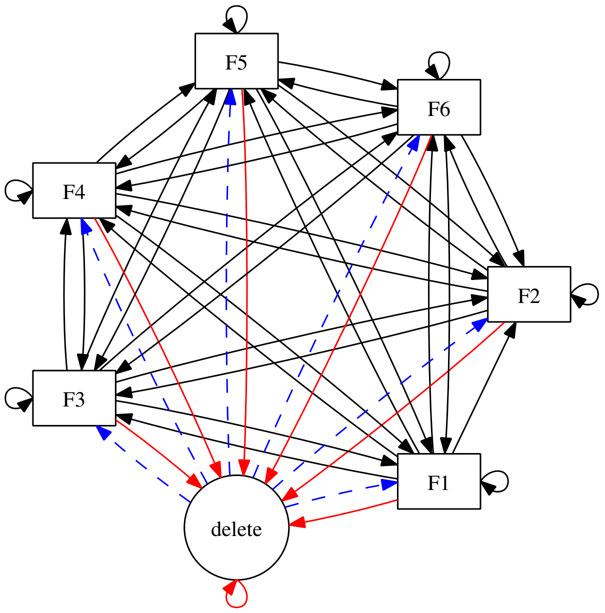


**Figure 1. The Frameseq delete-HMM model** All frame states F1 . . . F6 have transitions to each other and to themselves. Transitions to the delete state are symbolized by red arrows, to indicate that they share the transition probability, $P(delete)$. The dashed blue arrows illustrate transitions from the delete state to a frame state – the probability of which depend on the last frame state visited before the delete state. Furthermore, the delete state is drawn as circle rather than a box to convey that it resembles a silent state – it does produce emissions (predicted false positives) but we are only interested in emissions from the frame states (predicted true positives). To minimize visual clutter, a begin and end state have been omitted.

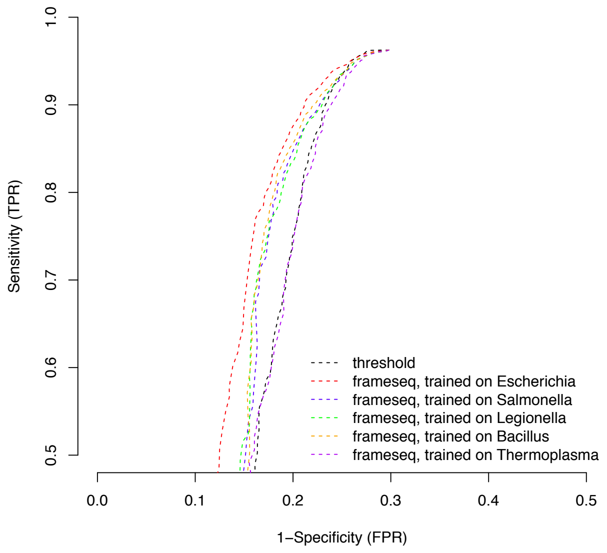


**Figure 2. ROC curves illustrating phylogenetic reach.** The figure shows ROC curves for filtering of all Genemark 2.5 predictions with score $> 0.1$ for *E. coli.* The black curve shows selection using a threshold and the colored curves show filtering using Frameseq. Note that the ROC curve does not extend all the way to the right; this is due to the 0.1 Genemark cutoff which still eliminates a lot of candidate predictions.

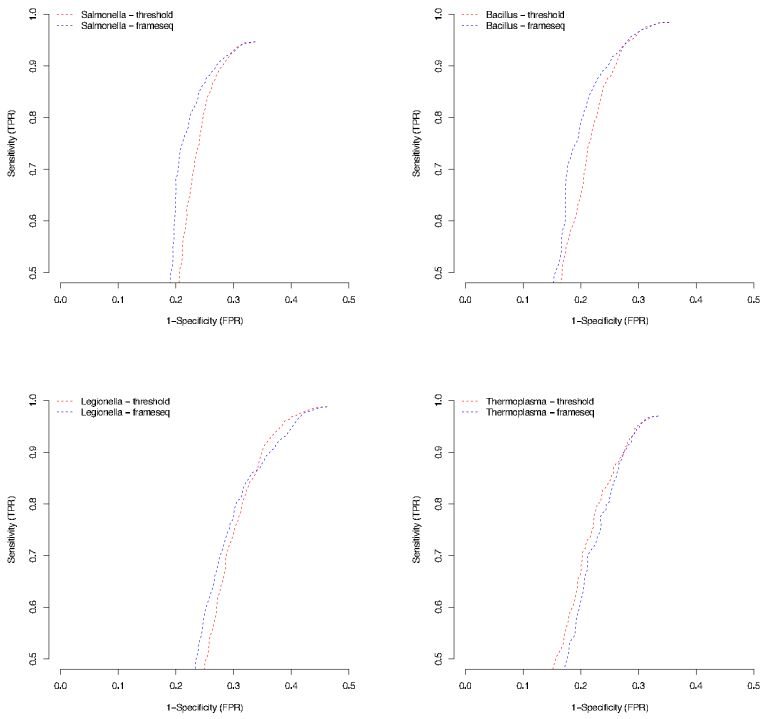


**Figure 3. ROC curve illustrating phylogenetic reach (inverse).** The figure shows ROC curves for filtering of all Genemark 2.5 predictions with score > 0.1 for different organisms. The black curve shows selection using a threshold and the colored curves show filtering using Frameseq. The Frameseq model is trained the *E. coli* for all organisms. The experiment shows that it is possible to train Frameseq on a well-known and well-annotated organism and apply it to filter predictions on phylogenetically distant organisms with improved accuracy. Accuracy is improved for both *S. enterica* and *B. subtilis* and in part for *L. pneumophila*, but not for the phylogenetically distant *T. acidophilum*.

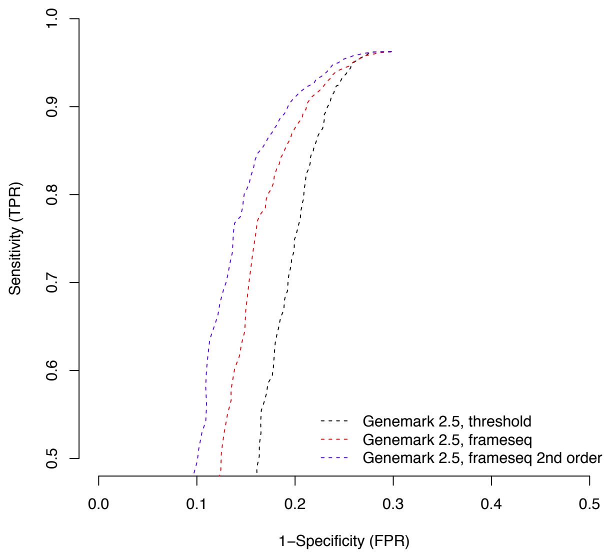


**Figure 4. ROC curves illustrating potential accuracy gain with second order model.** This ROC curve compares the basic first order Frameseq model to a second order model. The black curve indicate threshold selection, the red curve is the first order model and the blue curve is the second order model. Both the first order model and the second order model are trained on *E. coli* and applied to filter the predictions of the same genome. The second order model results in markedly better accuracy than with the first order model.

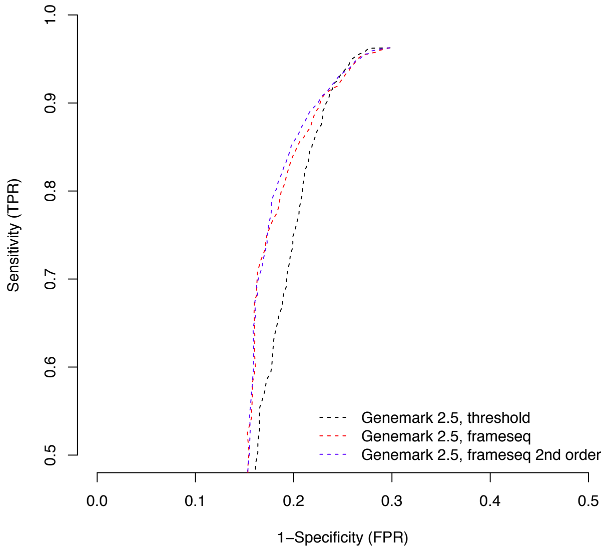


**Figure 5. Phylogenetic robustness of a second order signal.** This ROC curve compares the basic first order Frameseq model to a second order model. The black curve indicate threshold selection, the red curve is the first order model and the blue curve is the second order model. Both the first order model and the second order model are trained on *S. enterica* and applied to filter the predictions of *E. coli.* The second order model results in marginally better accuracy.

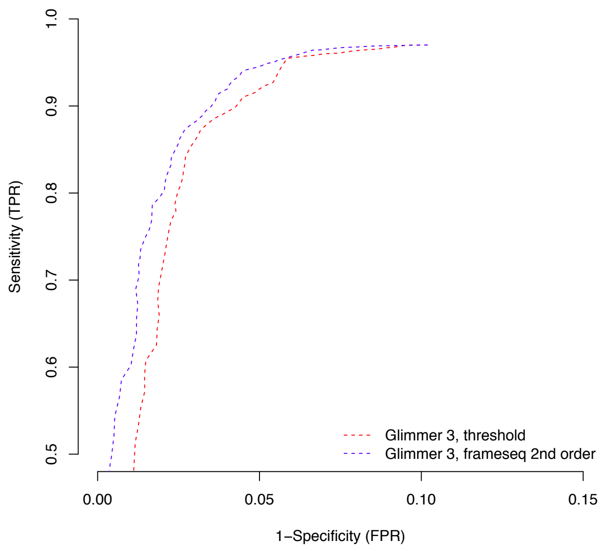


**Figure 6. Frameseq with Glimmer.** The red ROC curve shows threshold selection with Glimmer 3 predictions on *E. coli* and blue curve shows results of filtering these predictions with a second order Frameseq model trained on *E. coli*.

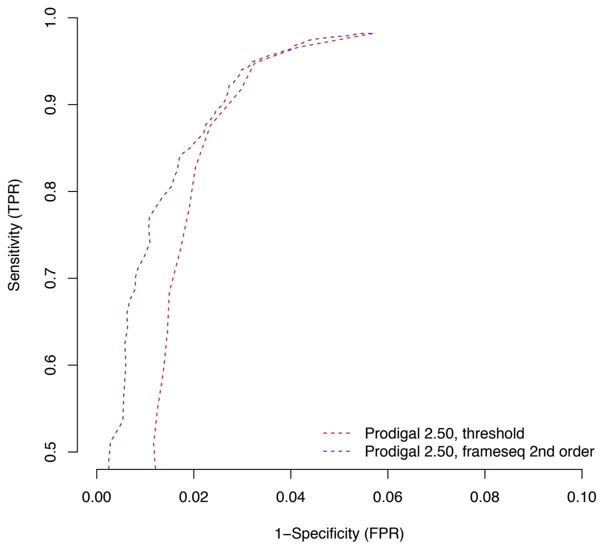


**Figure 7. Frameseq with Prodigal.** The red ROC curve shows threshold selection with Prodigal 2.50 predictions on *E. coli* and blue curve shows results of filtering these predictions with a second order Frameseq model trained on *E. coli*.

# Paper III

# Prototyping RNA Models as Probabilistic Logic Programs

Søren Mørk[1] Christian Theil Have[2] Jakob Hull Havgaard[3]

[1]Department of Science, Systems and Models, Roskilde University

[2]Department of Communication, Busines and Information Technologies, Roskilde University

[3]Center for Non-Coding RNAs in Technology and Health, University of Copenhagen

**Publication Status:**

In Preparation

# Prototyping RNA Models as Probabilistic Logic Programs

Søren Mørk, [1,3], Christian Theil Have[2], Jakob Hull Havgaard[3]

[1]Department of Science, Systems and Models, Roskilde University
[2]Department of Communication, Business and Information Technologies, Roskilde University
[3]Center for Non-Coding RNAs in Technology and Health, University of Copenhagen.

## Abstract

*Motivation:* RNA bioinformatics has seen tremendous growth recently. Sequence analysis tasks involving RNA are complicated by the long-distance interactions inherent in RNA secondary structure. RNA secondary structure prediction is typically based on either non-probabilistic Minimum Free Energy approaches or probabilistic approaches like Stochastic Context Free Grammars. These basic approaches have been used in methods for single sequence RNA secondary structure prediction, incorporated into RNA structural alignment, structure predictions from multiple alignments, RNA-RNA interaction prediction and phylogenetic inferences of RNAs.

*Approach:* We introduce probabilistic logic programming as a powerful tool for the development and application of probabilistic models for RNA sequence analysis. We develop a suite of RNA models as probabilistic logic programs in the probabilistic logic programming language and machine learning system PRISM. We start by introducing SCFG-like HMMs capable of parsing dot-bracket structures. Such models enable generic model structure compositions that have previously been developed for HMMs to be applied for modeling RNA secondary structures. These models are complemented by SCFG based RNA models - also with generic model structure compositions. The model structures are evaluated in terms of complexity measurements, learn statistics and prediction performance on single sequence secondary structure prediction using data from Rfam - benchmarked against the prediction performance of RNAfold 2.0.0. To examine the effect of over-fitting due to family structure of the dataset, we perform cross-validation on three test sets: i: an intra-family set with all families represented in all partitions, ii: an inter-family set with families present only in specific partitions, and iii: an inter-family test set augmented with more data for parametrization.

*Results:* We demonstrate the utility of probabilistic logic programming for RNA model development and testing. The HMM based models are equivalent to the SCFG based models in terms of single sequence secondary structure prediction performance. The prediction performance of both HMM based and SCFG based models can be improved using mixed memory structures, when trained on families, but is prone to over fitting. The effect is partially reversible by parametrizing models with a larger training set. Standard versions of both HMM based and SCFG based models perform slightly worse than RNAfold, whereas the mixed memory versions where better than RNAfold on the intra-family test set, but much worse on the inter-family set. The performance was recoverable by training on more data. The HMM based models are generally faster during learning than the SCFGs, obtaining higher likelihoods after EM learning. The HMM based models however, are much slower on decoding than the SCFGs (and both are orders of magnitude slower than RNAfold). A main advantage of expressing the models as probabilistic logic programs is the ease of which one can extend the models for other types of bioinformatic tasks involving RNA. To demonstrate this potential we briefly introduce a suite of models that can perform a number of different bioinformatic tasks involving RNA including models for pairwise alignment, pseudoknot and kissing hairpins models, models of overlapping RNAs, RNA-RNA interaction models and models of overlapping protein coding potential and RNA structure (mRNAs).

## Introduction

RNA has a unique potential triple capacity as information carrier, catalytic agent and as a key regulatory player. There is an increasing appreciation of the importance of RNA via its centrality in molecular biology. This is evident in the recent explosive growth in transcriptome sequencing and expression analysis that is driving major discoveries in contemporary biology. In parallel, there has been steady growth in the last decades of computational approaches for RNA bioinformatic tasks such as secondary and 3D structure prediction, alignment, phylogenetic inferences and genome scanning techniques. The long distance base-pairing interactions that determine RNA secondary structure, and the low level of primary sequence conservation due compensatory base changes makes RNA sequence analysis more challenging than DNA and protein sequence analysis.

Traditionally, RNA secondary structure prediction have primarily been performed via minimization of free energy (MFE) algorithms [1, 2, 3, 4]. An alternative approach to RNA secondary structure prediction has been a probabilistic machine learning approach based on Stochastic Context Free Grammars (SCFGs)[5, 6, 7, 8]. The use of MFE approaches for RNA analysis is currently far more widespread than that of SCFGs. The recent growth in RNA sequence data [9, 10] however makes probabilistic modeling more and more attractive. Additionally, using probabilistic sequence modeling makes the analysis of RNA amendable to the increasing repertoire of computational probabilistic modeling techniques (including probabilistic logic programming). Stochastic Context Free Grammars (SCFGs) [11, 12] are stochastic transformational grammars that capture the long distance interactions of base pairs related by a parse tree that can be generated and parsed by a number of (recurrent) production rules that builds a sequence top-down. SCFGs are equivalent with probabilistic push-down automatons [13] and can be viewed as generalizations of stochastic regular grammars such as Hidden Markov Models (HMMs). HMMs are ubiqitous in biological sequence analysis [14] . A large number of special model structures and extensions have been developed, including profile-HMMs [15] , pair-HMMs [14], factorial-HMMs [16], mixed memory HMMs [17] and HMMs incorporating Acyclic Discrete Phase Type (ADPH) length modeling [18] . The number of alternative SCFG model types and their application to RNA bioinformatics is somewhat more limited.

A recent paper by Rivas *et. al.* [19] show the potential of examining alternative model structures in a probabilistic setting. The possible structure variants are not exempted, and are currently limited to single sequence secondary structure prediction. We strongly believe that there is plenty of room for the discovery of novel probabilistic models for RNA bioinformatics that will provide the basis of better and more efficient bioinformatic inferences. We also believe that if the barrier to model development and testing due to specific algorithm development for specific models could be overcome, the discovery rate of better models would be greatly increased.

Probabilistic Logic programming provides a generic and unified framework for procedural representation of discrete probability distributions such as stochastic grammars and graphical models, combining the strength of a probabilistic approach with the expressiveness and the declarative advantages of logical programming. Logic Programming is a complete high-level programing paradigm that formalize relations in terms of logical predicates ensuring valid inferences [20]. Adding random variables enables a generic modeling of both stochastic grammars and graphical models. Probabilistic logic programming (based on the logic programming language prolog) is an ideal prototyping environment for preliminary exploration of this potential model space, since it offers a high level declarative approach to model development and testing where model alterations are easily produced and can be tested on a level platform. The power of probabilistic logic programming for bioinformatic model development and testing has previously been demonstrated by the use of the probabilistic logic programming language and machine learning system PRISM[21, 22] for evaluation of models underlying bacterial gene finders [23]. Both HMMs and SCFGs can be expressed as PRISM programs, easily manipulated and use the same generic machine learning routines for parameter estimation, decoding and sampling. Additionally, logic programs are amendable to various types of systematic program transformations and specifications that modifies the computational complexity yielding powerful yet realistic processing of even highly complicated programs.

Here, we present and compare a number of probabilistic logic models for RNA secondary structure modeling, developed and tested in the probabilistic logic language and machine learning system PRISM. The models are of two different types: HMM based models with a stack or SCFG based models. All models express the basic structural components of secondary RNA structures including stem regions, hairpin loops, multi-loops, internal loops, bulges, and external loops. The advantage of modeling RNA structure with a stochastic push-down automaton/SCFG-like-HMM is that such models can be extended and evaluated using the same model structure variants as are available for HMMs [24, 23]. Hence, it should be quite easy to develop and test different model structures such as different conditioning schemes, ADPH length modeling over structural components and pair- and triple-SCFG-like HMMs for pairwise alignment, repeat structures for pseudoknot models, complementary pair-HMMs for RNA-RNA interactions, factorial RNA-HMMs for overlapping RNAs and factorial HMM-SCFG-like HMMs for overlapping RNA structure and protein coding sequences (mRNAs).

To demonstrate the feasibility of using probabilistic logic programs as RNA models, we perform a number of experiments with variants of HMM based and SCFG based models on single sequence structure prediction. The models are benchmarked against RNAfold on a small subset of Rfam data- cleaned up to allow for moderately complex models to be trained and used for decoding. To show the ease of which new types of models can be developed from well performing models, we also provide a number of model extensions that include both HMM based and SCFG models for alignment, modeling pseudo knot and kissing hairpins, RNA-RNA interactions and for modeling overlapping RNA structure and protein coding sequences (mRNAs).

## Materials & Methods

### PRISM

We use PRISM version 2.1 (available at http://sato-www.cs.titech.ac.jp/prism/) for our experiments. PRISM models are extremely brief and elegant, with a close correspondance to the equations and recursive formulas usually used to present probabilistic models such as HMMs and SCFGs. This results in code that is versatile and easy to manipulate. The execution of a PRISM program (as with any other logic program) results in a solution tree. The PRISM machine learning system use this solution tree to build a generic Dynamic Programming matrix and run its inference and parameter estimation procedures on. PRISM 2.1 offers a generic Expectation-Maximization (EM) algorithm for parameter inference, a variational Bayes version (VB-EM) and a Markov Chain Monte Carlo parameter inference method. Decoding is produced with a Viterbi -like algorithm, returning an explanation graph from with the most probable state of hidden variables can be extracted (the Viterbi explanation). For a general introduction to PRISM see [25, 26].

### Hidden Markov Models

Before we go on to our probabilistic RNA models, we start by introducing how HMMs can be implemented in PRISM, and then proceed to extending these models towards models capable of modeling RNA secondary structure.

An HMM is characterized by a set of transition probabilities between unobserved states and a set of emission probabilities of observed characters emitted from states. Given an observed sequence $O = x_1, x_2, ..., x_n$ and a path $\Pi = \pi_1, \pi_2, ..., \pi_n$, the transition probabilities $a_{kl}$ are defined for position $i$ and states $k$ and $l$, as:

$$a_{kl} = \mathbf{P}(\pi_i = l \mid \pi_{i-1} = k), \tag{1}$$

and the emission probabilities $e_k(b)$ are defined for path $\Pi$, position $x_i$, state $k$ and character $b$, as:

$$e_k(b) = \mathbf{P}(x_i = b \mid \pi_i = k). \tag{2}$$

The joint probability of observing the sequence $X$ and the path $\Pi$ is:

$$\mathbf{P}(X,\Pi) = a_{0\pi_1} \prod_{i=1}^{n} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}. \tag{3}$$

A PRISM implementation of a standard two-state HMM that emits sequences from {a,c,g,u}, is given by a a collection of *values* predicates that are unit clauses (facts) that represents the transition and emission probabilities by conditionals and outcome space, combined with a recursive structure corresponding to the partitioning scheme of the joint probability equation (and predicates for initiation and termination):

```
%  parameters:
values(transition(state(begin)),[state(1),state(2),end]).
values(transition(state(1)),[state(1),state(2),end]).
values(transition(state(2)),[state(1),state(2),end]).
values(emission(state(1)),[a,c,g,u]).
values(emission(state(2)),[a,c,g,u]).

% initiation:
model(Observables) :-
   recursion(state(begin),Observables).

% recursive structure:
recursion(state(Si),[Xi|Rest]) :-
   msw(emission(state(Si)),Xi),
   msw(transition(state(Si)),NS),
   recursion(NS,Rest).

% termination:
recursion(end,[]).
```

## RNA-HMMs

An HMM based model that is capable of generating/parsing palindromic sequences (e.g. RNA stem regions) can be obtained via adding a stack and providing separate recursive structures for each state, where state 1 writes both to the sequence (the left side of a stem) and the stack (that will become the right side of a stem) and state 2 empties the stack into the sequence (producing the right side of a stem). The following PRISM code is of a model that emits dot-bracket structures corresponding to RNA secondary structures.

```
%  parameters:
values(transition(state(begin)),[state(1),state(2),end]).
values(transition(state(1)),[state(1),state(2),end]).
values(transition(state(2)),[state(1),state(2),end]).
values(emission(state(1)),[(<,>)).
values(emission(state(2)),[':']).

% initiation:
model(Observables) :-
   recursion(state(begin),[],Observables).

% recursion structure:
recursion(state(1),Stack,[Xi|Rest]) :-
   msw(emission(state(1)),(Xi,Wi)),
   msw(transition(state(1)),NS),
   recursion(NS,[Wi|Stack],Rest).

recursion(state(2),Stack,[Stack_head|Rest]) :-
   get_stack(Stack,Stack_head,Stack_tail),
   msw(transition(state(1)),NS),
   recursion(NS,Stack_tail,Rest).

get_stack([H|T],H,T).

% termination:
recursion(end,[],[]).
```

Figure 1 shows the complete PRISM source code of an RNA HMM model that can generate RNA secondary structures. The model generates/parses both a nucleotide sequence and a dot-bracket annotation.

## RNA SCFGs

Stochastic Context-Free Grammars is the probabilistic extension of Context-Free Grammars. Context-Free Grammars (CFG) were introduced by Chomsky [27] and is a generative grammatical formalism which describe language through recursive structures. A context-free grammar can be characterized using a number of (recursive) production rules consisting of non-terminals ($N$) and terminals ($t$). Without loss of generality we say that a rule $head \rightarrow body$ of a CFG is on the form,

$$N \rightarrow t|tN|tNt$$

Terminals are actual symbols in the language and the non-terminals represent the compositional syntax of the grammar. The meaning of a rule is that the head can be replaced by the sequence of terminals and non-terminals in the body. Top-down parsing with CFGs derives an observed sequence of terminals from a single non-terminal. CFGs can be ambiguous in the sense that they may have several rules with the same non-terminal in the head.

Stochastic Context-Free Grammars (SCFGs) is an extension of CFGs which associate a probability to each rule. Parsing with SCFGs usually amounts to finding the most probable derivation of the observed terminal sequence from a (particular) non-terminal.

HMMs are equivalent to (may express the same set of languages as) right-recursive SCFGs, where rules are on the restricted form,

$$N \rightarrow t|tN$$

Probabilistic Push-Down Automata (PPDAs), e.g., RNA HMMs, and SCFGs are theoretically equivalent under certain conditions [28], but are different with regards to parsing strategies. Essentially, PPDAs are operational specifications with an explicit parsing strategy whereas SCFGs are purely declarative. In the SCFGs considered here, however, parsing is a top-down process which starts with a non-terminal and ultimately derives the observed sequence. In contrast, RNA HMMs can be seen as using a combination of bottom-up parsing (pushing non-terminals to the stack) and top-down parsing (popping non-terminals from the stack).

SCFGs for modeling RNA structures can be composed of production rules representing structural components including stem regions, hairpin loops, internal loops, multi-loops, bulges, and external loops [14].

The PRISM code of an SCFG that can emit the same types of dot-bracket structures as the RNA-HMM is given here:

```
% parameters:
values(transition(free),[[':',free],[free,free],[stem],[':']]).
values(transition(stem),[
                        ['<',stem,'>'],
                        [stem,stem],
                        ['<',stem,rb,'>'],
                        ['<',lb,stem,'>'],
                        [loop]
                        ]).

values(transition(loop),[[':',loop],[c,loop],[g,loop],[u,loop],[stem,loop],[':']]).
values(transition(lb),[[':',lb],[':']]).
values(transition(rb),[[':',rb],[':']]).

% initiation:
model(L):-
        scfg(free,L-[]).

% recursion:
scfg(LHS,L0-L1):-
        (
        terminal(LHS)->
        annot(LHS,W,A),
        L0=[W|L1]
        ;
        msw(transition(LHS),RHS),
        projection(RHS,L0-L1)
        ).

% "stack termination"
projection([],L-L).

% "stack handling"
projection([RHS_head|RHS_tail],L0-L2):-
        scfg(RHS_head,L0-L1),
        projection(RHS_tail,L1-L2).

% terminal designation:
terminal(':').
terminal(''<).
terminal('>').
```

Figure 2 show the entire prism code of the an RNA SCFG (rnascfg.psm) in annotation version.

## Model structures for supervised learning

In order to parameterize the models using supervised learning model variants are constructed that can parse and generate both a nucleotide sequence and a dot-bracket notation representation of the secondary structure (The models can also be be parameterized in sequence mode (unsupervised) from unannotated RNA sequences). Supervised learning of the SCFGs can be done using annotation versions trained on data of the form "$model([a,c,g,u],[<,:,:,>])$.". The parameters of the annotation and the sequence models are equivalent, so the sequence models can be used to decode sequences using parameter values obtained via supervised training by the corresponding annotation model.

## Extended Structures for Single-Sequence RNA Models

More elaborate model structure can be obtained by using mixed memory [17] versions were transitions and emissions can be conditioned on the state, previous states, the content of the stack, previous emissions or any combination hereof. This can be done to allow for larger scale structural composition modeling and modeling of context effects like stacking. Such models are easily produced by changing just a few characters in the values declarations and the recursions. See supplementary materials for the PRISM code of all the models used and presented here.

The five model variants used for the experiments are:

- *rnahmm* - a standard RNA HMM model with canonical base pairing and wobble base pairs
- *mmrnahmm* - a mixed memory version of *rnahmm* with transitions conditioned on state and emission
- *mmrnahmm*2 - a mixed memory version of *rnahmm* with transitions conditioned on state, previous state, emission and previous emission
- $rnascfg$ - a standard RNA SCFG model with canonical base pairing and wobble base pairs
- *mmrnahmm*2 - a mixed memory version of $rnascfg$ with transitions conditioned on the previous Left Hand Side

## Data

The experiments were conducted using RNA sequences and secondary structure annotations derived from Rfam version 10.1. We use only a subset of Rfam namely families with maximum average pairwise sequence similarity of 90%, on individual pairwise sequences of $< 95\%$ similarity and at minimum 30 members pr family. The dataset contains 166 families with 15276 sequences in total.

This dataset was then additionally filtered to only have canonical base-pairing, removing pseudo knot annotations, and removing sequences with non $\{acgu\}$ characters, and sequences with ”Predicted” in the comment field - leaving only sequences with a ”Published” tag. This dataset contains 9796 sequences in 104 families. The length distribution of this dataset is given in figure 3.

Due to the complexity constraints of the RNAHMM models a smaller dataset of sequences with a maximum length of 50 its was extracted (467 seq.) and partitioned for 5x cross-validation. SCFGs for RNA secondary sequence prediction are prone to over fitting [19]. To examine this effect we made 3 data sets for cross-validation studies: Test set 1 partitions were produced such that each family was partitioned separately into the the 5 fold cross validation partitions in order to ensure that they were represented in each training and decode datasets. Test set 2 partitions were made using families, such that no family had members in more than one partition. Test set 3 use test set 2 augmented with the 4430 sequences between 100 and 50 nts in our initial dataset for training and test set 2 for decoding (to compensate for the data sparseness of test set 2). The training sets for each partition where the concatenation of the other 4 partitions. Training of both rnahmm and rnascfg models were done on data in $model([a,c,g,u,...],[<,:,:,>,...])$. format. Decoding was done on data in $lst$ format: ”$[a,c,g,u,...]$”.

## Model parameterization and decoding

In order to parameterize the models using supervised learning annotation variants are constructed that can parse and generate both a nucleotide sequence and a dot-bracket notation representation of the secondary structure.

To speed up training and decoding of the HMM based models, we use the autoAnnotations tool by Christiansen and Gallagher [29] - that enables efficient supervised learning using an annotation model - and outputs the most probable annotation sequence after decoding.

Decoding was performed using the single-SCFG like HMM versions with the PRISM $viterbif$ returning an annotated viterbi graph from which the state path was retrieved. Evaluation of the models were based on statistical measures returned by PRISM after learning sessions as well as classification performance in terms of correctly predicting base-pairing regions with respect to the annotations.

We use the consensus structure of RNA alignments as a golden standard. Prediction performance was measured as base pairing True Positive Rate *TPR* (Sensitivity/Precision), Positive Prediction Value *PPV* (Specificity) and *MCC* value. With True Positive Rate given as:

$$TPR = \frac{TP}{TP+FN} \tag{4}$$

Positive Predictive Value given as

$$PPV = \frac{TP}{TP+FP} \tag{5}$$

and MCC value given as

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{6}$$

## Results

Table 1 show the learn statistics of the learning sessions (average over all training sets) on test set 1. In learning mode the RNA HMMs are faster than the RNA SCFGs with slightly lower likelihoods obtained after training, except *mmrnahmm*2 which have the highest likelihood of all the models.

### Complexity analysis

The complexity of the decoding of the different models was determined by decoding random sequences of lengths 10,20,30,40 and 50 nts - measuring time and memory complexity as user time (in seconds) and maximum resident set size (in kbytes). The complexity of the decoding of each sequence was recorded in the same way. Table 2 show average time complexity of the models based on decoding of 5 sets of 10 random sequences in different lengths from 10 its to 50 nts. Table 3 show the corresponding average memory complexity of the models. Figure 4 show the linear relationship between time and memory complexity of decoding the test set with *rnahmm*. Figure 5 show the relationship between time and memory complexity of decoding the test set with $rnascfg$. Figure 6 and 7 show the memory complexity of *rnahmm* and $rnascfg$, respectively.

Usually, the complexity of RNA folding algorithms are $O(M^3)$ in time and $O(M^2)$ in memory, where M is the length of the sequence. SCFGs are $O(M^3)$ in both time and memory and $O(M^2)$ in memory, where M is the length of the sequence and N the number of nonterminals. This is a theoretical (worst case) upper bound, with actual complexity depending on the sequence composition (*e.g.* sequences like 'aaaaauuuuu' has only a single derivation, and hence folding of such sequences hence scale linearly in their lengths. The RNA HMMs are orders of magnitude slower than the SCFG and these are again much slower than RNAfold.

The RNA HMMs in their current implementation are $O(M^3)$ in both time and memory for all sequences (see Figure 4), whereas the RNA SCFG models only for some sequences are also $O(M^3)$ in both time and memory.

There is a very large sequence dependent difference in time and memory use of the RNAHMMs (and some of the RNASCFGs), due to their current implementation (please see Discussion for further details).

## Prediction Performance

Prediction performance is given in table 4 for *rnahmm*, table 5 for *mmrnahmm*, table 6 for *mmrnahmm*2, table 7 for *rnascfg*, table 8 for *mmrnascfg* and table 9 for RNAfold. The corresponding results on test set 2 are given in tables 10 to 15. The results from test set 3 for *rnascfg* is given in table 13.

There is a large variation in prediction performance for all models on the different families. The prediction of "exotic" RNAs such as JUMPstart, Prion_pknot (it is a pseudoknoted sequence - for training and verification only non-pseudo knotting base pairs are counted), glnA and PK-IAV using test set 1 and Bacteria_large_SRP as well for test set 2, is very poor for all models. This is most likely due to "unusual" structures in the "exotic" RNAs that leads to data-sparseness when trained only on other families. RNAfold performs very well on the Bacteria_large_SRP family, but also poorly on prion_pknot and PK-IAV.

Overall, the standard RNA HMM and RNA SCFG models (*rnahmm* and *rnascfg*) are equivalent in total prediction performance, both being slightly lower than RNAfold. *mmrnahmm* is significantly worse than the other models and both *mmrnahmm*2. and *mmrnascfg* are better than RNAfold in terms of total prediction performance on test set 1.

The opposite effect is seen in the results from test set 2. None of the models perform nearly as well as RNAfold and the use of mixed memory structures generally impair the prediction performance. This again is most likely also due to data-sparseness, as the more complex models are more vulnerable to "lack of data".

The data-sparseness problem can be compensated by training on a larger and more diverse dataset as can be seen from Figure 17, that gives the results of the prediction performance of *rnahmm* on test set 3 that includes 4430 additional sequences of length between 50 and 100 nts in the training sets (not present in the decoding sets). Training on more divergent data increases the prediction performance even as compared to the inter-family results from using test set 1, especially for the ”exotic” RNAs. The prediction performance Prion_pknot, Bacteria large SRP and PK-IAV was not recoverable by using test set 3, whereas JUMPstart and pan were partially recovered. *rnascfg* trained on test set 3 only outperformed RNAfold on JUMPstart, glnA, and HIV-FE. *mmrnascfg* trained on test set 3 show similar recovery as with *rnascfg*, with improved prediction over *mmrnascfg* on test 2, better prediction than *rnascfg* on test 3 yet not as good prediction as RNAfold on test 2 nor of *mmrnascfg* on test 1. Prediction performance of *mmrnascfg* trained on test set 3 is close to that of RNAfold, with 6 out of 14 families achieving better prediction than RNA fold.

Using the single sequence secondary structure model variants described here with parameters optioned from training on either all sequences under 100 nts in our initial dataset, possibly augmented by fragments of longer sequences, would provide powerful secondary structure modeling both in its own right, and when imported into the models in our RNA model suite.

## RNA model suite

One of the main advantages of developing RNA models as probabilistic logic programs is that they are easily changed into novel model types. These models can be based on modeling principles for well performing models, and even in some cases use parameter values imported from other models (such as single sequence RNA models). The following section gives a short description of a suite of model types that we have developed. The PRISM code of all models are available as supplementary materials.

### Pairwise Alignment Models

Pairwise sequence alignment is a cornerstone of contemporary bioinformatics. The papers by Needleman-Wunsch [30] and Smith-Waterman [31, 32] laid the foundation of dynamic programming pairwise alignment algorithms that have since been generalized to probabilistic models [33] including pair-HMMs [14], extended to multiple alignments [34, 35, 36] and incorporation of phylogenetic structure [37, 38, 39], as well as serving as the basis for search tools like BLAST [40].

Pairwise alignment of noncoding RNAs is an important problem, that combines alignment and RNA secondary structure prediction. The seminal paper by Sankoff [41] described the simultaneous RNA structure prediction and alignment with a number of approximations of the Sankov 85 algorithm developed subsequently *e.g.* [42].

We have made two models for pairwise alignments constrained by RNA secondary structure.

One model is an RNA HMM based model consisting of a combination of a Smith-Waterman style pair-HMM with a stochastic push-down automaton (hairpin pair.hmm). The other model is a pair-SCFG model (pair_scfg.psm). The PRISM code of both are available in the supplementary materials. Figure 8 give the PRISM code of a pair SCFG model.

## Pseudoknot and Kissing Hairpins Models

Pseudo knots and kissing hairpins are tertiary structures formed by base pairing interactions between loop and non-loop regions (pseudoknots) and between loop regions (kissing hairpins). Pseudoknots and kissing hairpins cannot be expressed by SCFGs, due to the repeat like sequences that require a Context Sensitive Grammar [14]. We have developed pseudo knot and kissing hairpins models based on RNA HMM models by adding an extra stack to the model for recoding the pseudo knotted or kissing sequences. Figure 9 gives the PRISM source code for HMM based pseudoknot and a kissing hairpin models.

## mRNA models

Messenger RNA (mRNA) genes are the most abundant type of genes in any organism. mRNA genes gets transcribed into RNA that then gets translated into protein. Hence, mRNA genes have dual functions: they encode RNA molecules and they encode protein molecules. RNA molecules adopt various structures and so do proteins, both of which impose constraints on the DNA sequences that encodes them.

mRNAs are multitasking molecules - the sequential composition of mRNA genes is constrained by both RNA structure and protein coding potential simultaneously. Modeling this overlapping functionality would be important for sequence analysis of mRNA genes including gene finding, structure prediction, alignment and phylogenetic inferences - as well as for providing an interesting insight into how nature has resolved the inherent problems in multitasking through the past 4 billion years.

There is a growing catalogue of overlapping RNA structures and protein coding sequences, from examples in HIV, [43] to the Human genome [44] as well as statistical indications [45, 46] of prevalent overlap, even indications that bacterial mRNAs are subject to constraints by RNA structure in general [47].

In addition, it is known that synonymous mutations can have effect on gene expression [48, 49, 50, 51], that mRNA folding effects translational efficiency [52] and even that synonomous substitutions possibly affects protein structure [53]. There are all indications that RNA structure is potentially important feature of mRNA sequence composition.

Modeling overlapping RNA structure and protein coding sequences have been performed by [54, 55, 56, 57], but with an emphasis on phylogenetics with relatively limited models of the overlaps.

The sequential constraints imposed on DNA by protein coding potential is typically modeled using Hidden Markov Models (HMMs) that captures the local dependencies between nucleotides. HMMs of protein coding potential are extensively used for gene finding[14], and protein structure modeling[58]. A number of different model structures for capturing protein coding potential were explored in Mørk-Holmes 2012 [23].

Here, we demonstrate the feasibility of producing a full probabilistic model encompasing both the RNA coding and protein coding potential of mRNAs. The model is a factorial-HMM model with two conditionally independent state chains and a single emitted nucleotide sequence chain. One state chain is a mixed memory HMM that models the protein coding potential of the mRNA. The other state chain is an SCFG-like HMM that models the RNA secondary structure of the mRNA. The RNA submodel and the protein coding submodel can be parameterized separately and combined for decoding mRNA sequences.

The backbone of the mRNA model is a factorial-HMM [16] like model with two hidden state sequence chains and a single observable sequence chain.

Each chain emits sequences from $\{a,c,g,t\}$ and coordination of the behavior of the two sub-models is achieved by a specific "agreement" emission of the single observed sequence conditioned on the emissions of the two sub-models. Coordination is achieved by fixing the parameter values to have point probability close to 1 for the outcome agreed upon, and uniformly distributed outcomes if the two sub-models disagree.

Figure 10 give the PRISM code of an RNA HMM based mRNA model. An SCFG based mRNA model is available in the supplementary materials (mrna_scfg.psm).

## Discussion

We have demonstrated utility of using probabilistic logic programming as a prototyping environment for the development and testing of novel probabilistic models for RNA bioinformatics. We have shown how probabilistic logic programming enables a level comparison of different models. We have tested three different HMM based models and two different SCFG based models, and shown the prediction performance equivalence of HMM based and SCFG based RNA models. Furthermore we have shown that these are on par with RNAfold in terms of secondary sequence structure prediction when trained on family members, and that more capable models can be developed by using mixed memory model architectures.

The driving idea underlying our introduction of probabilistic logic modeling for RNA secondary sctructure modeling is to be able to swiftly develop models of RNA secondary structure, to test them on a level platform and to expand them into model types used for other types of bioinformatic tasks involving RNA sequences.

The single sequence secondary structure models are available in both pure sequence versions and annotation versions. The latter can be used for visualizing the underlying state sequence of sequences sampled from the model and for supervised learning, given sequences with dot-bracket notation annotated secondary structure.

The currently very inefficient implementation of the HMM based models makes it hard to evaluate the performance of the models on larger datasets, that would increase the reliability of our statistical approach.

The complexity of the RNA HMM models to a large extend is due to an inefficient tabling strategy for the RNA HMM, where the stack contents are tabled in a way where they add to the number of nonterminals. Even though the general complexity of the models are the usual $O(N^3L^3)$ as for the Inside-Outside algorithm [12], the dynamic variation of N produces a very large number of derivations in a highly sequence dependent manner. Future work on using an alternative tabling strategy or incorporation of difference lists into the RNA HMMs might make the models as efficient as SCFGs are.

Despite the horrendous complexity of the RNA HMM models, the ease of which they are transformed into other types of models, such as those for pseudo knots and kissing hairpins, these models may still be relevant in their present form. They can, e.g., useful for incorporating the repeat like base pairing regions of pseudo knots and kissing hairpins, given that those regions are usually very short, and much simpler than ordinary RNA secondary structure that have bifurcation and nested structures that needs to be recoded in the stack and gives rise to the largest growth in complexity due to structural variation.

In order to produce probabilistic models that are comparable to the complexity of thermodynamic DP approaches one could either implement the (well performing) models in efficient generalized dynamic programming languages like Bellman's GAP [59], and try dynamic pruning of the explanation graph in PRISM, reducing the number of parses as in Havgaard *et. al.* 2007 [60].

The performance of our models are very data dependent. We find that our models are prone to over-fitting when trained on sparse datasets. This leads to good prediction performance on data that resamples the training data (*i.e.* data trained on representatives of the same family), but bad performance when decoding families that have not been used for training. This effect is somewhat compensated by training on a more diverged dataset. In that respect it must be remembered that our benchmark MFE RNA folding method (RNAfold) make use of thousands of energy parameters . The validation of RNA alignments (and indeed RNA secondary structure prediction) is further complicated by a limited amount of proper golden standard data due to the difficulties in experimental elucidation of RNA structure as a consequence of the instability of most RNA molecules.

The resemblance of the RNA HMM models to left-corner parsers [61] that parse the sequence bottom up and sideways, should make it easier to incorporate ordinary sequence constraints due to neighboring effects such as stacking or higher order signals (longer distance memory of what has been emitted or the sequence of structural classes represented by the states of the model) in order to provide a model that e.g. models local interactions in 5' to 3' directions prior to downstream base pairing. This would provide models that mimics the time and orientation dependent aspect of the folding of RNA as it is transcribed, something that current models do not incorporate. Further work could also be done on examining implementations of RNA models using left-corner parsers along the same lines as we have provided here.

The limited number of single sequence secondary structure models that we examine here by no means exhaust the structure space of models capable of parsing RNA sequences. Our formulation of the models as declarative logic programs should also make it easier to provide a genetic algorithm based approach for systematically exploring this structure space.

In addition to the single sequence secondary structure models, we have provided a suite of extended models for alignment, RNA-RNA interaction modeling, pseudo knot and kissing hairpin modeling and modeling overlapping RNA structure and protein coding potential, that remain to be benchmarked but demonstrates that well performing single sequence models can potentially be used as a basis for improving other types of RNA bioinformatic tasks too.

We conclude that the use of probabilistic logic programming is a promising platform for prototyping and discovering novel models for RNA bioinformatics, that is likely to produce future models that will improve and expand the computational prediction and inferences involving RNA sequences.

## References

[1] Zuker, M. and Stiegler, P. (1981) Optimal computer folding of large RNA sequences using thermodynamic and auxiliary information. *Nucleic Acids Res.* **9**:133–148.

[2] Zuker, M. (1989) Computer prediction of RNA structure. *Methods Enzymol.* **180**:262–288.

[3] Zuker, M. (1994) Prediction of RNA secondary structure by energy minimization. *Methods Mol. Biol.* **25**:267294.

[4] Hofacker, I., Fekete, M. and Stadler, P. (2002) Secondary structure prediction for aligned RNA sequences. *J. Mol. Biol.* **319**:1059–1066.

[5] Sakakibara, Y., Brown, M., Underwood, R. C., Mian, I. S. and Haussler, D. (1994) Stochastic context-free grammars for modeling RNA. in *Proceedings of the 27th Annual Hawaii International Conference on System Sciences. Volume 5 : Biotechnology Computing*, ed Hunter, L. (IEEE Computer Society Press, Los Alamitos, CA, USA), pp 284–294.

[6] Eddy, S. R. and Durbin, R. (1994) RNA sequence analysis using covariance models. *Nucleic Acids Res.* **22**:2079–2088.

[7] Knudsen, B. and Hein, J. (2003) Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Res.* **31**:3423–3428.

[8] Holmes, I. (2005) Accelerated probabilistic inference of RNA structure evolution. *BMC Bioinformatics* **6**:73.

[9] Gardner, P., Wilm, A. and Washietl, S. (2005) A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic Acids Res* **33**:24332439.

[10] Gardner, P., Daub, J., Tate, J., Nawrocki, E., Kolbe, D., Lindgreen, S., Wilkinson, A., Finn, R., Griffiths-Jones, S. and Eddy, S. e. a. (2009) Rfam: updates to the RNA families database. *Nucleic Acids Res* **37**:D136–D140.

[11] Chomsky, N. (1959) On certain formal properties of grammars. *Information and Control* **2**:137–167 Also in 'Readings in mathematical psychology', R.D. Luce, R. Bush, and E. Galanter (eds.), New York: Wiley, pp. 125-155, 1965.

[12] Baker, J. K. (1979) Trainable grammars for speech recognition. in *Proceedings of Spring Conference of the Acoustical Society of America* pp 547–550.

[13] Abney, S., Mcallester, D. and Pereira, F. (1999) Relating probabilistic grammars and automata. in *In Proceedings of ACP99* (Morgan Kaufmann), pp 542–549.

[14] Durbin, R., Eddy, S., Krogh, A. and Mitcheson, G. (1998) *Biological Sequence Analysis* (Cambridge University Press).

[15] Krogh, A., Brown, M., Mian, I. S., Sjlander, K. and Haussler, D. (1994) Hidden markov models in computational biology : Applications to protein modeling. *Journal of Molecular Biology* **235**:1501–1531.

[16] Ghahramani, Z. and Jordan, M. I. (1996) Factorial hidden markov models. *Machine Learning* **29**:245–273.

[17] Saul, L. K. and Jordan, M. I. (1999) Mixed memory markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Machine Learning* **37**:75–87.

[18] Bobbio, A., Horváth, A., Scarpa, M. and Telek, M. (2003) Acyclic discrete phase type distributions: properties and a parameter estimation algorithm. *Perform. Eval* **54**:1–32.

[19] Rivas, E., Lang, R. and Eddy, S. R. (2012) A range of complex probabilistic models for RNA secondary structure prediction that includes the nearest-neighbor model and more. *RNA* **18**:193–212.

[20] van Emden, M. and Kowalsk, R. (1976) The Semantics of Predicate Logics as a Programming Language. *J. ACM* **23**:733–742.

[21] Sato, T. and Kameya, Y. (1997) PRISM: A Language for Symbolic-Statistical modeling. in *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI97)* pp 1330–1339.

[22] Sato, T. (2009) Generative modeling by PRISM. in *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009*, Lecture Notes in Computer Science, eds Hill, P. M. and Warren, D. S. (Springer) Vol. 5649, pp 24–35.

[23] Mørk, S. and Holmes, I. (2012) Evaluating bacterial gene-finding hmm structures as probabilistic logic programs. *Bioinformatics* **28**:636–42.

[24] Christiansen, H., Have, C. T., Lassen, O. T. and Petit, M. (2011) Taming the Zoo of Discrete HMM Subspecies & some of their Relatives. in *Proceedings of the first International Work-Conference on Linguistics, Biology and Computer Science: Interplays* (IOS Press).

[25] Sato, T. and Kameya, Y. (2001) Parameter Learning of Logic Programs for Symbolic-Statistical Modeling. *J. Artif. Intell. Res.* **15**:391–454.

[26] Sato, T., fa Zhou, N., Kameya, Y. and Izumi, Y. (2010) PRISM Users Manual (Version 2.0).

[27] Chomsky, N. (1956) Three models for the description of language. *Information Theory, IRE Transactions on* **2**:113–124.

[28] Nederhof, M. and Satta, G. (2006) Probabilistic parsing strategies. *Journal of the ACM (JACM)* **53**:406–436.

[29] Christiansen, H. and Gallagher, J. P. (2009) Non-discriminating Arguments and their Uses. in *Logic Programming. 25th International Conference, ICLP 2009*, eds Hill, P. M. and Warren, D. S. (Springer-Verlag), pp 55–69.

[30] Needleman, S. and Wunsch, C. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48**:443–453.

[31] Smith, T. and Waterman, M. (1981) Identification of common molecular subsequences. *Journal of Molecular Biology* **147**:195–197.

[32] Gotoh, O. (1982) An improved algorithm for matching biological sequences. *Journal of Molecular Biology* **162**:705–708.

[33] Bishop, M. and Thompson, E. (1986) Maximum likelihood alignment of DNA sequences. *Journal of Molecular Biology* **190**:159–165.

[34] Feng, D. and Doolitle, R. (1987) Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution* **25**:351–360.

[35] Thompson, J., Higgins, D. and Gibson, T. (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequnce weighting, position specific gap penalties and weight matrix choice. *NAR* **22**:4673–4680.

[36] Barton, G. and Sternberg, M. (1987) A strategy for the rapid multiple alignment of protein sequences. *Journal of Molecular Biology* **198**:327–337.

[37] Sankoff, D. and Cedergren, R. (1983) Simultaneous comparison of three or more sequences related by a tree. in *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison*, eds Sankoff, D. and Kruskal, J. (Addison-Wesley), pp 253–264.

[38] Hein, J. (1989) A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. *Molecular biology and Evolution* **6**:649–668.

[39] Thorne, J., Kishino, H. and Felsenstein, J. (1992) Inching toward reality: an improved likelihood model of sequence evolution. *Methods in Enzymology* **34**:3–16.

[40] Altschul, S., Gish, W., Miller, W., Myers, E. and Lipman, D. (1990) Basic local alignment search tool. *Journal of Molecular Biology* **215**:403–410.

[41] Sankoff, D. (1985) Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM J Appl Math* **45**:810–825.

[42] Havgaard, J., Lyngsø, R., Stormo, G. and Gorodkin, J. (2005) Pairwise local structural alignment of RNA sequences with sequence similarity less than 40%. *Bioinformatics* **21**:1815–1824.

[43] Sanjuán, R. and Bordería, A. V. (2011) Interplay between RNA Structure and Protein Evolution in HIV-1. *Molecular Biology and Evolution* **28**:1333–1338.

[44] Washietl, S., Pedersen, J. S., Korbel, J. O., Stocsits, C., Gruber, A. R., Hackermüller, J., Hertel, J., Lindemeyer, M., Reiche, K., Tanzer, A., Ucla, C., Wyss, C., Antonarakis, S. E., Denoeud, F., Lagarde, J., Drenkow, J., Kapranov, P., Gingeras, T. R., Guigó, R., Snyder, M., Gerstein, M. B., Reymond, A., Hofacker, I. L. and Stadler, P. F. (2007) Structured RNAs in the ENCODE selected regions of the human genome. *Genome Research* **17**:852–864.

[45] Meyer, I. M. and Miklós, I. (2005) Statistical evidence for conserved, local secondary structure in the coding regions of eukaryotic mRNAs and pre-mRNAs. *Nucleic Acids Research* **33**:6338–6348.

[46] Itzkovitz, S., Hodis, E. and Segal, E. (2010) Overlapping codes within protein-coding sequences. *Genome Research* **20**:1582–1589.

[47] Katz, L. and C.B., B. (2003) Widespread selection for local rna secondary structure in coding regions of bacterial genes. *Genome Research* **13**:2042–2051.

[48] Kudla, G., Murray, A. W., Tollervey, D. and Plotkin, J. B. (2009) Coding-Sequence Determinants of Gene Expression in *Escherichia coli*. *Science* **324**:255–258.

[49] Supek, F. and Šmuc, T. (July 2010) On relevance of Codon Usage to Expression of Synthetic and Natural Genes in *Escherichia coli*. *Genetics* **185**:1129–1134.

[50] Tuller, T., Waldman, Y. Y., Kupiec, M. and Ruppin, E. (2010) Translation efficiency is determined by both codon bias and folding energy. *Proceedings of the National Academy of Sciences* **107**:3645–3650.

[51] Sharp, P. M., Emery, L. R. and Zeng, K. (2010) Forces that influence the evolution of codon bias. *Philosophical Transactions of the Royal Society B: Biological Sciences* **365**:1203–1212.

[52] Tsao, D., Shabalina, S. A., Gauthier, J., Dokholyan, N. V. and Diatchenko, L. (2011) Disruptive mRNA folding increases translational efficiency of catechol-O-methyltransferase variant. *Nucleic Acids Research* **doi**:10.1093/nar/gkr165.

[53] Saunders, R. and Deane, C. M. (2010) Synonymous codon usage influences the local protein structure observed. *Nucleic Acids Research* **38**:6719–6728.

[54] Pedersen, J. S., Meyer, I. M., Forsberg, R., Simmonds, P. and Hein, J. (2004) A comparative method for finding and folding RNA secondary structures within protein-coding regions. *Nucleic Acids Res.* **32**:4925–4936.

[55] Pedersen, J. S. and Hein, J. (2003) Gene finding with a hidden Markov model of genome structure and evolution. *Bioinformatics* **19**:219–227.

[56] Knies, J. L., Dang, K. K., Vision, T. J., Hoffman, N. G., Swanstrom, R. and Burch, C. L. (2008) Compensatory Evolution in RNA Secondary Structures Increases Substitution Rate Variation among Sites. *Molecular Biology and Evolution* **25**:1778–1787.

[57] McCauley, S. and Hein, J. (2006) Using hidden Markov models and observed evolution to annotate viral genomes. *Bioinformatics* **22**:1308–1316.

[58] Frellsen, J., Moltke, I., Thiim, M., Mardia, K. V., Ferkinghoff-Borg, J. and Hamelryck, T. (2009) A probabilistic model of RNA conformational space. *PLoS Comput. Biol.* **5**:e1000406.

[59] Sauthoff, G., Janssen, S. and Giegerich, R. (2011) Bellmans GAP - a declarative language for dynamic programming. in *Proceedings of 13th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming, PPDP '11* (ACM).

[60] Havgaard, J., Torarinsson, E. and Gorodkin, J. (2007) Fast pairwise structural RNA alignments by pruning of the dynamical programming matrix. *PLoS Comput. Biol.* **3**:18961908.

[61] Rosenkrantz, S. and II, P. L. (1970) Deterministic left corner parsing. in *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata* pp 139–152.

## Tables and Figures

| model | Free param | Learn time | graph size | log Likelihood |
|---|---|---|---|---|
| rnahmm | 21 | 0.327±0.01 | 60412±593.836 | -25667.19±173.673 |
| mmrnahmm | 60 | 0.398±0.058 | 60412±593.836 | -25284.028±172.03 |
| mmrnahmm2 | 454 | 0.41±0.076 | 62612±598.237 | -23414.728±146.826 |
| rnascfg | 50 | 18.487±0.083 | 4752810±53114.662 | -25468.216±188.989 |
| mmrnascfg | 119 | 21.658±0.358 | 5701716±63731.83 | -24593.381±179.637 |

Table 1: Average learn statistics for the models trained on test set 1 (intra-family partition scheme)). Notice the difference between HMM based and SCFG based models.

| model | 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|---|
| rnahmm | 0.019±0.003 | 0.026±0.005 | 0.185±0.089 | 2.778±2.218 | 45.908±47.053 |
| mmrnahmm | 0.03±0.0 | 0.04±0.009 | 0.276±0.12 | 2.907±2.056 | 47.054±48.884 |
| mmrnahmm2 | 0.041±0.005 | 0.068±0.016 | 0.367±0.127 | 3.64±2.755 | 59.625±63.158 |
| rnascfg | 0.023±0.005 | 0.056±0.011 | 0.127±0.016 | 0.318±0.06 | 0.609±0.046 |
| mmrnascfg | 0.02±0.0 | 0.052±0.004 | 0.139±0.003 | 0.31±0.011 | 0.612±0.013 |
| rnafold | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 | 0.0±0.0 |

Table 2: Time complexity results of the different models measured with time -v as User time (in seconds) on test set 1 (intra-family partition scheme). Notice the large difference between the HMM based models, the SCFG based models and RNAfold.

| model | 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|---|
| rnahmm | 51986±5664 | 68502±9845 | 259133±96557 | 2671779±1923772 | 42460770±42077102 |
| mmrnahmm | 53211±2560 | 69994±8174 | 258304±98781 | 2675957±1918706 | 42459194±42073200 |
| mmrnahmm2 | 57130±5040 | 78891±8377 | 306829±123025 | 3312350±2414708 | 52227776±52956070 |
| rnascfg | 56696±7103 | 82773±4949 | 123677±3474 | 203259±4956 | 333770±16234 |
| mmrnascfg | 63184±3310 | 93307±8158 | 144011±10051 | 236210±6781 | 409488±5882 |
| rnafold | 5259±26 | 5270±18 | 5315±44 | 5342±35 | 5302±68 |

Table 3: Memory complexity results of the different models measured with time -v as Resident Set Size (in kbytes) for decoding of test set 1 (intra-family partition scheme).

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| IRE | 33 | 0.406 | 0.297 | 0.338 |
| JUMPstart | 43 | 0.093 | 0.107 | 0.097 |
| Prion_pknot | 80 | 0.247 | 0.353 | 0.288 |
| mini-ykkC | 50 | 0.912 | 0.811 | 0.853 |
| Histone3 | 48 | 0.876 | 0.913 | 0.893 |
| SAM-SAH | 46 | 0.685 | 0.528 | 0.597 |
| Acido-Lenti-1 | 55 | 0.883 | 0.756 | 0.806 |
| glnA | 61 | 0.829 | 0.421 | 0.586 |
| PK-IAV | 32 | 0.05 | 0.05 | 0.047 |
| total | 448 | 0.554 | 0.471 | 0.501 |

Table 4: Prediction performance of *rnahmm* on test set 1 (intra-family partition scheme). Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| IRE | 33 | 0.5 | 0.4 | 0.438 |
| JUMPstart | 43 | 0.111 | 0.14 | 0.12 |
| Prion_pknot | 80 | 0.336 | 0.385 | 0.353 |
| mini-ykkC | 50 | 0.873 | 0.778 | 0.819 |
| Histone3 | 48 | 0.922 | 0.91 | 0.914 |
| SAM-SAH | 46 | 0.63 | 0.498 | 0.556 |
| Acido-Lenti-1 | 55 | 0.902 | 0.828 | 0.858 |
| glnA | 61 | 0.213 | 0.152 | 0.173 |
| PK-IAV | 32 | 0.078 | 0.094 | 0.083 |
| total | 448 | 0.507 | 0.465 | 0.479 |

Table 5: Prediction performance of *mmrnahmm* on test set 1 (intra-family partition scheme) Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| IRE | 33 | 0.875 | 0.791 | 0.822 |
| JUMPstart | 43 | 0.385 | 0.391 | 0.384 |
| Prion_pknot | 80 | 0.552 | 0.567 | 0.553 |
| mini-ykkC | 50 | 0.898 | 0.883 | 0.887 |
| Histone3 | 48 | 0.81 | 0.884 | 0.841 |
| SAM-SAH | 46 | 0.865 | 0.87 | 0.863 |
| Acido-Lenti-1 | 55 | 0.936 | 0.837 | 0.88 |
| glnA | 61 | 0.669 | 0.655 | 0.656 |
| PK-IAV | 32 | 0.471 | 0.492 | 0.479 |
| total | 448 | 0.718 | 0.708 | 0.707 |

Table 6: Prediction performance of *mmrnahmm*2 on test set 1 (intra-family partition scheme) Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| IRE | 33 | 0.664 | 0.525 | 0.574 |
| JUMPstart | 43 | 0.225 | 0.362 | 0.278 |
| Prion_pknot | 80 | 0.112 | 0.149 | 0.121 |
| mini-ykkC | 50 | 0.631 | 0.659 | 0.639 |
| Histone3 | 48 | 0.657 | 0.854 | 0.741 |
| SAM-SAH | 46 | 0.693 | 0.539 | 0.606 |
| Acido-Lenti-1 | 55 | 0.787 | 0.82 | 0.799 |
| glnA | 61 | 0.704 | 0.632 | 0.654 |
| PK-IAV | 32 | 0.18 | 0.25 | 0.207 |
| total | 448 | 0.517 | 0.532 | 0.513 |

Table 7: Prediction performance of *rnascfg* test set 1 (intra-family partition scheme) Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| IRE | 33 | 0.575 | 0.54 | 0.548 |
| JUMPstart | 43 | 0.282 | 0.386 | 0.323 |
| Prion_pknot | 80 | 0.233 | 0.26 | 0.239 |
| mini-ykkC | 50 | 0.905 | 0.889 | 0.895 |
| Histone3 | 48 | 0.818 | 0.955 | 0.879 |
| SAM-SAH | 46 | 0.838 | 0.546 | 0.671 |
| Acido-Lenti-1 | 55 | 0.92 | 0.903 | 0.908 |
| glnA | 61 | 0.682 | 0.643 | 0.655 |
| PK-IAV | 32 | 0.376 | 0.378 | 0.372 |
| total | 448 | 0.625 | 0.611 | 0.61 |

Table 8: Prediction performance of *mmrnascfg* on test set 1 (intra-family partition scheme) Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| IRE | 33 | 0.746 | 0.755 | 0.745 |
| JUMPstart | 43 | 0.308 | 0.53 | 0.398 |
| Prion_pknot | 80 | 0.023 | 0.046 | 0.024 |
| mini-ykkC | 50 | 0.807 | 0.828 | 0.814 |
| Histone3 | 48 | 0.694 | 0.896 | 0.777 |
| SAM-SAH | 46 | 0.778 | 0.931 | 0.847 |
| Acido-Lenti-1 | 55 | 0.907 | 0.907 | 0.905 |
| glnA | 61 | 0.709 | 0.764 | 0.732 |
| PK-IAV | 32 | 0.013 | 0.031 | 0.014 |
| total | 448 | 0.554 | 0.632 | 0.584 |

Table 9: Prediction performance of RNAfold on test set 1 (intra-family partition scheme) Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| pan | 3 | 0.925 | 1.0 | 0.96 |
| ffh | 2 | 0.875 | 0.909 | 0.889 |
| Prion_pknot | 80 | 0.185 | 0.282 | 0.221 |
| Bacteria_large_SRP | 2 | 0.0 | 0.0 | -0.006 |
| Hammerhead_3 | 4 | 0.804 | 0.858 | 0.828 |
| JUMPstart | 43 | 0.093 | 0.107 | 0.097 |
| mini-ykkC | 50 | 0.75 | 0.583 | 0.655 |
| glnA | 61 | 0.094 | 0.06 | 0.074 |
| PK-IAV | 32 | 0.05 | 0.05 | 0.047 |
| IRE | 33 | 0.379 | 0.276 | 0.315 |
| Acido-Lenti-1 | 55 | 0.835 | 0.696 | 0.751 |
| HIV_FE | 6 | 0.876 | 0.965 | 0.912 |
| Histone3 | 48 | 0.746 | 0.788 | 0.764 |
| SAM-SAH | 46 | 0.719 | 0.571 | 0.634 |
| total | 465 | 0.524 | 0.51 | 0.51 |

Table 10: Prediction performance of *rnahmm* on test set 2 (inter-family partition scheme). Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| pan | 3 | 0.333 | 0.242 | 0.278 |
| ffh | 2 | 0.25 | 0.273 | 0.251 |
| Prion_pknot | 80 | 0.203 | 0.294 | 0.235 |
| Bacteria_large_SRP | 2 | 0.0 | 0.0 | -0.01 |
| Hammerhead_3 | 4 | 0.825 | 0.858 | 0.839 |
| JUMPstart | 43 | 0.104 | 0.121 | 0.109 |
| mini-ykkC | 50 | 0.732 | 0.542 | 0.623 |
| glnA | 61 | 0.088 | 0.046 | 0.062 |
| PK-IAV | 32 | 0.0 | 0.0 | -0.002 |
| IRE | 33 | 0.397 | 0.269 | 0.321 |
| Acido-Lenti-1 | 55 | 0.727 | 0.581 | 0.642 |
| HIV_FE | 6 | 0.904 | 0.965 | 0.933 |
| Histone3 | 48 | 0.748 | 0.736 | 0.734 |
| SAM-SAH | 46 | 0.718 | 0.55 | 0.622 |
| total | 465 | 0.431 | 0.391 | 0.403 |

Table 11: Prediction performance of *mmrnahmm* on test set 2 (inter-family partition scheme). Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| pan | 3 | 0.333 | 0.242 | 0.273 |
| ffh | 2 | 0.875 | 0.591 | 0.715 |
| Prion_pknot | 80 | 0.065 | 0.077 | 0.063 |
| Bacteria_large_SRP | 2 | 0.0 | 0.0 | -0.006 |
| Hammerhead_3 | 4 | 0.789 | 0.594 | 0.669 |
| JUMPstart | 43 | 0.074 | 0.083 | 0.073 |
| mini-ykkC | 50 | 0.504 | 0.403 | 0.443 |
| glnA | 61 | 0.267 | 0.125 | 0.178 |
| PK-IAV | 32 | 0.0 | 0.0 | -0.003 |
| IRE | 33 | 0.44 | 0.289 | 0.344 |
| Acido-Lenti-1 | 55 | 0.861 | 0.622 | 0.721 |
| HIV_FE | 6 | 0.788 | 0.783 | 0.782 |
| Histone3 | 48 | 0.57 | 0.651 | 0.6 |
| SAM-SAH | 46 | 0.457 | 0.345 | 0.392 |
| total | 465 | 0.43 | 0.343 | 0.374 |

Table 12: Prediction performance of *mmrnahmm*2 on test set 2 (inter-family partition scheme). Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| pan | 3 | 0.333 | 0.242 | 0.276 |
| ffh | 2 | 0.881 | 1.0 | 0.938 |
| Prion_pknot | 80 | 0.052 | 0.088 | 0.059 |
| Bacteria_large_SRP | 2 | 0.125 | 0.063 | 0.077 |
| Hammerhead_3 | 4 | 0.689 | 0.885 | 0.779 |
| JUMPstart | 43 | 0.121 | 0.195 | 0.146 |
| mini-ykkC | 50 | 0.526 | 0.57 | 0.541 |
| glnA | 61 | 0.516 | 0.555 | 0.529 |
| PK-IAV | 32 | 0.023 | 0.063 | 0.031 |
| IRE | 33 | 0.523 | 0.344 | 0.406 |
| Acido-Lenti-1 | 55 | 0.788 | 0.835 | 0.806 |
| HIV_FE | 6 | 0.64 | 1.0 | 0.796 |
| Histone3 | 48 | 0.536 | 0.813 | 0.65 |
| SAM-SAH | 46 | 0.652 | 0.551 | 0.593 |
| total | 465 | 0.457 | 0.514 | 0.473 |

Table 13: Prediction performance of $rnascfg$ test set 2 (inter-family partition scheme). Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| pan | 3 | 0.333 | 0.182 | 0.236 |
| ffh | 2 | 0.0 | 0.0 | -0.015 |
| Prion_pknot | 80 | 0.064 | 0.1 | 0.073 |
| Bacteria_large_SRP | 2 | 0.0 | 0.0 | -0.01 |
| Hammerhead_3 | 4 | 0.45 | 0.5 | 0.469 |
| JUMPstart | 43 | 0.099 | 0.13 | 0.107 |
| mini-ykkC | 50 | 0.632 | 0.575 | 0.597 |
| glnA | 61 | 0.517 | 0.171 | 0.291 |
| PK-IAV | 32 | 0.0 | 0.0 | -0.005 |
| IRE | 33 | 0.115 | 0.065 | 0.076 |
| Acido-Lenti-1 | 55 | 0.906 | 0.751 | 0.815 |
| HIV_FE | 6 | 0.861 | 1.0 | 0.927 |
| Histone3 | 48 | 0.628 | 0.896 | 0.742 |
| SAM-SAH | 46 | 0.32 | 0.338 | 0.323 |
| total | 465 | 0.352 | 0.336 | 0.33 |

Table 14: Prediction performance of *mmrnascfg* on test set 2 (inter-family partition scheme). Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| pan | 3 | 1.0 | 0.914 | 0.955 |
| ffh | 2 | 0.959 | 1.0 | 0.978 |
| Prion_pknot | 80 | 0.023 | 0.046 | 0.024 |
| Bacteria_large_SRP | 2 | 1.0 | 0.938 | 0.968 |
| Hammerhead_3 | 4 | 0.736 | 1.0 | 0.855 |
| JUMPstart | 43 | 0.308 | 0.53 | 0.398 |
| mini-ykkC | 50 | 0.807 | 0.828 | 0.814 |
| glnA | 61 | 0.709 | 0.764 | 0.732 |
| PK-IAV | 32 | 0.013 | 0.031 | 0.014 |
| IRE | 33 | 0.746 | 0.755 | 0.745 |
| Acido-Lenti-1 | 55 | 0.907 | 0.907 | 0.905 |
| HIV_FE | 6 | 0.623 | 1.0 | 0.785 |
| Histone3 | 48 | 0.694 | 0.896 | 0.777 |
| SAM-SAH | 46 | 0.778 | 0.931 | 0.847 |
| total | 465 | 0.665 | 0.753 | 0.7 |

Table 15: Prediction performance of RNAfold on test set 2 (inter-family partition scheme). Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| pan | 3 | 0.912 | 0.917 | 0.912 |
| ffh | 2 | 0.728 | 0.955 | 0.829 |
| Prion_pknot | 80 | 0.036 | 0.067 | 0.041 |
| Bacteria_large_SRP | 2 | 0.125 | 0.063 | 0.079 |
| Hammerhead_3 | 4 | 0.713 | 0.969 | 0.829 |
| JUMPstart | 43 | 0.413 | 0.712 | 0.534 |
| mini-ykkC | 50 | 0.667 | 0.728 | 0.692 |
| glnA | 61 | 0.745 | 0.749 | 0.741 |
| PK-IAV | 32 | 0.0 | 0.0 | -0.006 |
| IRE | 33 | 0.783 | 0.694 | 0.728 |
| Acido-Lenti-1 | 55 | 0.761 | 0.79 | 0.77 |
| HIV_FE | 6 | 0.693 | 0.965 | 0.811 |
| Histone3 | 48 | 0.671 | 0.917 | 0.776 |
| SAM-SAH | 46 | 0.655 | 0.568 | 0.603 |
| total | 465 | 0.564 | 0.649 | 0.596 |

Table 16: Prediction performance of *rnascfg* test set 3 (augmented inter-family partition scheme). Prediction performance is given as true positive rate (sensitivity) TPR, positive predictive value (specificity) and Mathews correlation coefficient MCC, for each family in the test set. Number of members are also indicated for each family. Total prediction performance (bottom row) is the arithmetic mean of the family wise prediction performance measures.

| fam | members | TPR | PPV | MCC |
|---|---|---|---|---|
| pan | 3 | 1.0 | 1.0 | 1.0 |
| ffh | 2 | 0.783 | 0.955 | 0.861 |
| Prion_pknot | 80 | 0.115 | 0.208 | 0.147 |
| Bacteria_large_SRP | 2 | 0.0 | 0.0 | -0.009 |
| Hammerhead_3 | 4 | 0.887 | 0.969 | 0.924 |
| JUMPstart | 43 | 0.278 | 0.381 | 0.316 |
| mini-ykkC | 50 | 0.89 | 0.91 | 0.897 |
| glnA | 61 | 0.773 | 0.705 | 0.733 |
| PK-IAV | 32 | 0.05 | 0.075 | 0.055 |
| IRE | 33 | 0.697 | 0.587 | 0.628 |
| Acido-Lenti-1 | 55 | 0.899 | 0.908 | 0.901 |
| HIV_FE | 6 | 0.639 | 0.965 | 0.779 |
| Histone3 | 48 | 0.667 | 0.872 | 0.753 |
| SAM-SAH | 46 | 0.733 | 0.559 | 0.634 |
| total | 465 | 0.601 | 0.65 | 0.616 |

Table 17: Prediction performance of *mmrnascfg* test set 3 (augmented inter-family partition scheme).

```
% parameters
values(t(begin),[unpaired(single),push]).
values(t(single),[unpaired(single),push,end]).
values(t(push),[push,unpaired(lbulge),unpaired(loop),pull]).
values(t(loop),[unpaired(loop),pull]).
values(t(pull),[pull,unpaired(rbulge),unpaired(bifurcation)]).
values(t(pullout),[unpaired(single),end]).
values(t(lbulge),[unpaired(lbulge),push]).
values(t(rbulge),[unpaired(rbulge),pull]).
values(t(bifurcation),[unpaired(bifurcation),push]).

values(e(push),[(a,u),(c,g),(g,c),(u,a),(g,u),(u,g)]).
values(e(_),[a,c,g,u]).

model(Seq,Dot_bracket):-
        msw(t(begin),NS),
        r(NS,[],Seq,Dot_bracket).

r(end,_,[],[]).

r(unpaired(X),Stack,[W|R1],[':'|R2]):-
        msw(e(unpaired),W),
        can_empty_stack(Stack,R1),
        msw(t(X),NS),
        r(NS,Stack,R1,R2).

r(push,Tail,[W|R1],['<'|R2]):-
        msw(e(push),(W,S)),
        can_empty_stack([S|Tail],R1),
        msw(t(push),NS),
        r(NS,[S|Tail],R1,R2).

r(pull,[],R1,R2):-
        msw(t(pullout),NS),
        r(NS,[],R1,R2).

r(pull,Stack,[Head|R1],['>'|R2]):-
        get_stack(Stack,Head,Tail),
        msw(t(pull),NS),
        r(NS,Tail,R1,R2).

get_stack([Head|Tail],Head,Tail).
```

Figure 1: PRISM code of a RNA HMM model

```
% parameters:
values(transition(free),[[a,free],[c,free],[g,free],[u,free],[free,free],[stem],[a],[c],[g],[u]]).
values(transition(stem),[
                                    [al,stem,ur],
                                    [cl,stem,gr],
                                    [gl,stem,cr],
                                    [ul,stem,ar],
                                    [gl,stem,ur],
                                    [ul,stem,gr],
                                    [stem,stem],

                                    [al,stem,rb,ur],
                                    [cl,stem,rb,gr],
                                    [gl,stem,rb,cr],
                                    [ul,stem,rb,ar],
                                    [gl,stem,rb,ur],
                                    [ul,stem,rb,gr],

                                    [al,lb,stem,ur],
                                    [cl,lb,stem,gr],
                                    [gl,lb,stem,cr],
                                    [ul,lb,stem,ar],
                                    [gl,lb,stem,ur],
                                    [ul,lb,stem,gr],

                                    [loop]
                                    ]).

values(transition(loop),[[a,loop],[c,loop],[g,loop],[u,loop],[stem,loop],[a],[c],[g],[u]]).
values(transition(lb),[[a,lb],[c,lb],[g,lb],[u,lb],[a],[c],[g],[u]]).
values(transition(rb),[[a,rb],[c,rb],[g,rb],[u,rb],[a],[c],[g],[u]]).

model(L,A):-
        scfg(free,L-[],A-[]).

scfg(LHS,L0-L1,A0-A1):-
        (
        terminal(LHS)->
        annot(LHS,W,A),
            A0=[A|A1],
            L0=[W|L1]
        ;
        msw(transition(LHS),RHS),
        projection(RHS,L0-L1,A0-A1)
        ).

projection([],L-L,A-A).

projection([RHS_head|RHS_tail],L0-L2,A0-A2):-
        scfg(RHS_head,L0-L1,A0-A1),
        projection(RHS_tail,L1-L2,A1-A2)
        .

terminal(a).
terminal(c).
terminal(g).
terminal(u).
terminal(al).
terminal(cl).
terminal(gl).
terminal(ul).
terminal(ar).
terminal(cr).
terminal(gr).
```

Figure 2: PRISM code of a RNA SCFG model

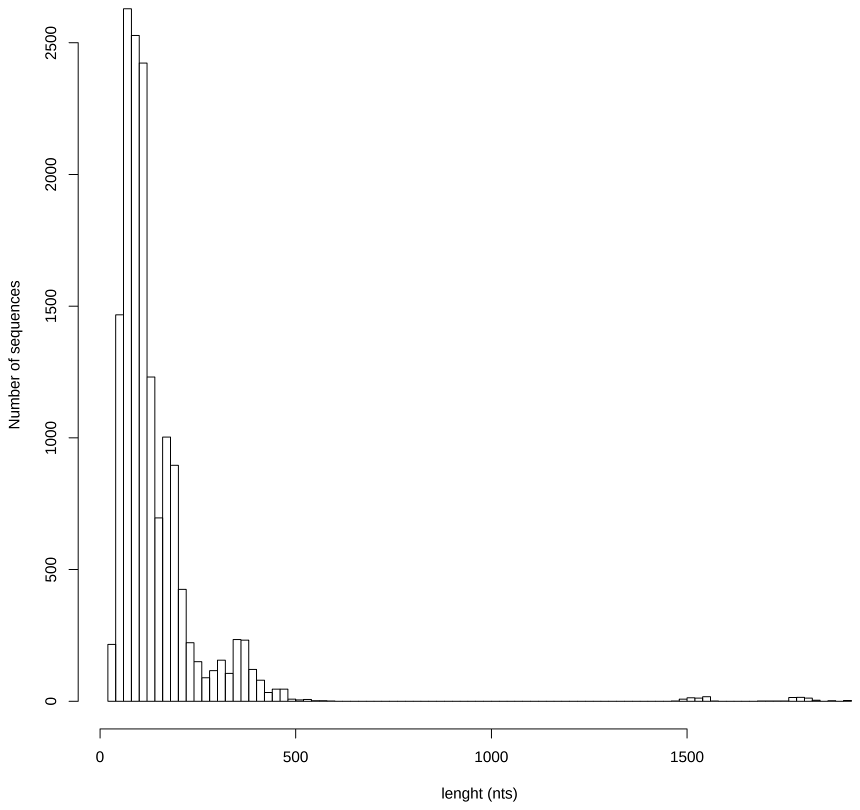


Figure 3: Length distribution of initial Rfam93.90.30 dataset. Notice the large peak of sequences shorter than 100 nts. The small peaks of sequences longer than 1500 nts are large subunit ribosomal RNAs.

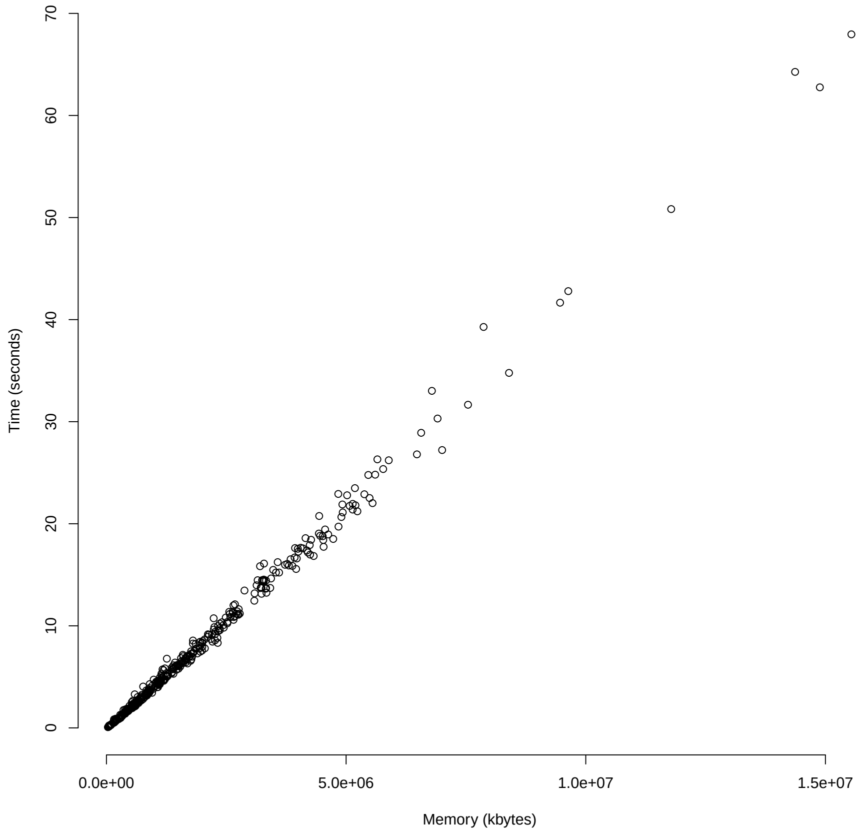


Figure 4: Scatterplot of time and memory complexity of rnahmm.psm decoding of test set 1 (intra-family partition scheme). Notice the somewhat unusual linear relationship for the time and memory complexity.

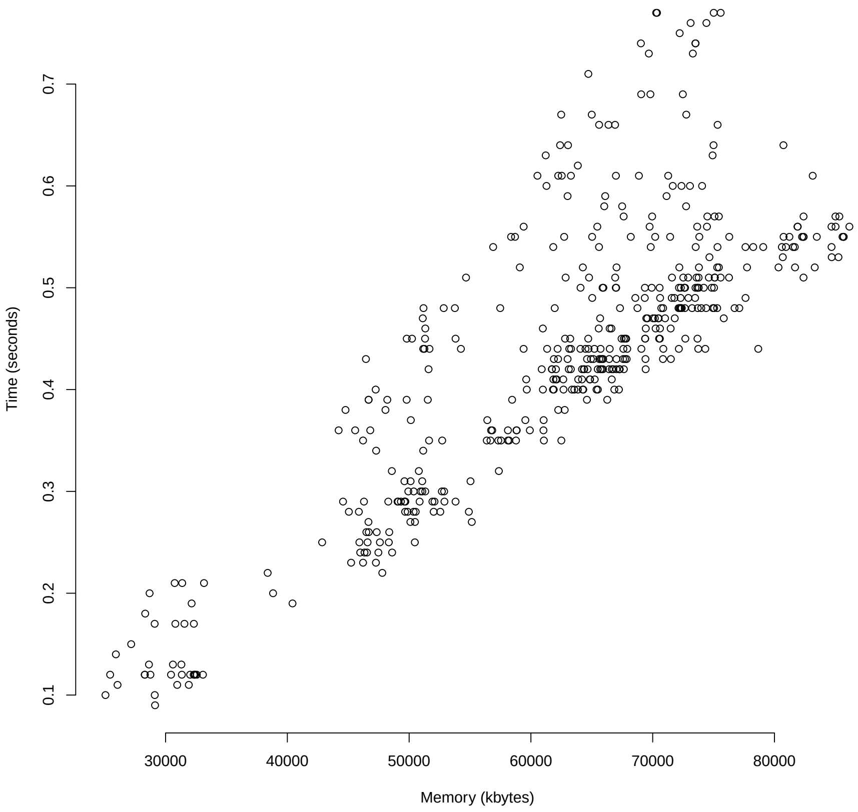


Figure 5: Scatterplot of time and memory complexity of rnahmm.psm decoding of test set 1 (intra-family partition scheme).

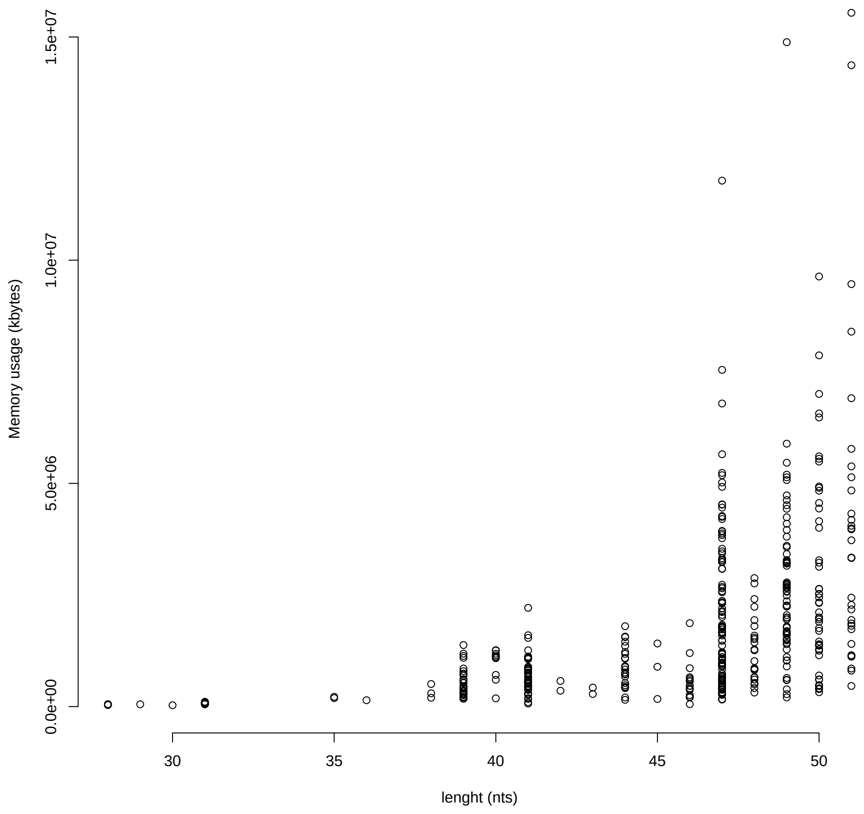


Figure 6: Memory complexity of the rnahmm.psm decoding of the test set 1 (intra-family partition scheme). Notice that the decoding complexity of rnahmm is heavily dependent upon the sequence composition (due to the inefficient tabling procedure).

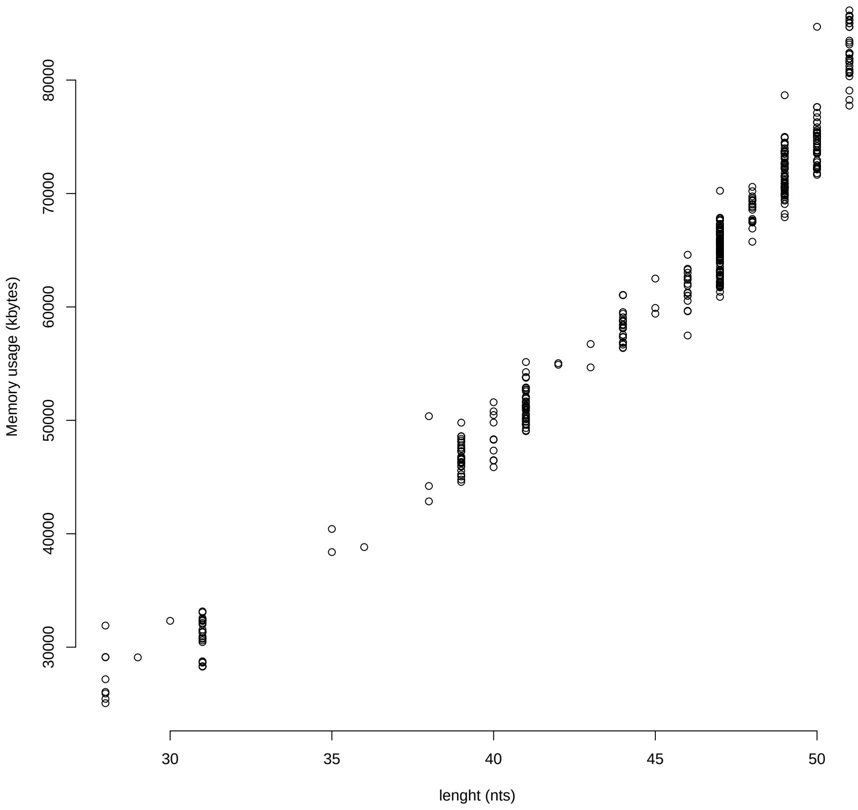


Figure 7: Memory complexity of the rnascfg.psm decoding of the test set 1 (intra-family partition scheme).

```
values(begin, [[match],[x_state],[y_state]]).
values(match,[[m(a,a), match, m(u,u)],
            [m(c,c), match, m(g,g)],
            [m(g,g), match, m(c,c)],
            [m(u,u), match, m(a,a)],
            [m(a,a),match],[m(c,c),match],
            [m(g,g),match],[m(u,u),match],
            [x_state],[y_state],
            [m(a,a)],[m(c,c)],[m(g,g)],[m(u,u)]]).
values(x_state,[[ x(a), x_state, x(u)],
            [ x(c), x_state, x(g)],
            [ x(g), x_state, x(c)],
            [ x(u), x_state, x(a)],
            [x(a),x_state],[x(c),x_state],
            [x(g),x_state],[x(t),x_state],
            [x(a),match],[x(c),match],
            [x(g),match],[x(t),match],
            [x(a)],[x(c)],[x(g)],[x(u)]]).
values(y_state,[[ y(a), y_state, y(u)],
            [ y(c), y_state, y(g)],
            [ y(g), y_state, y(c)],
            [ y(u), y_state, y(a)],
            [y(a),y_state],[y(c),y_state],
            [y(g),y_state],[y(u),y_state],
            [y(a),match],[y(c),match],
            [y(g),match],[y(u),match],
            [y(a)],[y(c)],[y(g)],[y(u)]]).

model(L1,L2):- scfg(begin, L1-[], L2-[]).

scfg(LHS, L10-L11, L20-L21):-
            nonterminal(LHS)->
            msw(LHS,RHS),
            projection(RHS,L10-L11, L20-L21).

scfg(m(X,Y), L10-L11,L20-L21):-
            L10=[X|L11],L20=[Y|L21].

scfg(x(X), L10-L11,L20-L21):-
            L10=[X|L11],L20=L12.

scfg(y(Y), L10-L11,L20-L21):-
            L10=L11,L20=[Y|L21].

projection([],L1-L1, L2-L2).

projection([H|T], L10-L11, L20-L21):-
            scfg(H, L10-L12, L20-L22),
            projection(T, L12-L11, L22-L21).

nonterminal(begin). nonterminal(match).
nonterminal(x_state). nonterminal(y_state).

terminal(m(a,a)). terminal(m(c,c)).
terminal(m(g,g)). terminal(m(u,u)).

terminal(x(a)). terminal(x(c)).
terminal(x(g)). terminal(x(u)).
terminal(y(a)). terminal(y(c)).
terminal(y(g)). terminal(y(u)).
```

Figure 8: PRISM code of a pair-scfg model for pairwise alignment

```
values(t(begin),[unpaired(5)]).
values(t(unpaired),[push(1),unpaired(5)]).
values(t(push(1)),[loop(1),push(1)]).
values(t(loop(1)),[pseudo,loop(1)]).
values(t(pseudo),[loop(2),pseudo]).
values(t(loop(2)),[pull,loop(2)]).
values(t(pull),[pull]).
values(t(unpaired,stay),[knot,unpaired(3)]).
values(t(knot,stay),[knot]).
values(t(knot,go),[loop(2)]).
values(t(unpaired,go),[end,unpaired(3)]).
values(e(_),[a,c,g,u]).

model(Seq,Dot_bracket):-
            msw(t(begin),Next),
            r(Next,[],[],Seq,Dot_bracket).

r(unpaired(5),Stack,Knot,[W|R1],['.'|R2]):-
            msw(e(x),W),
            msw(t(unpaired),Next),
            r(Next,Stack,Knot,R1,R2).

r(unpaired(3),[],[],[W|R1],['.'|R2]):-
            msw(e(x),W),
            msw(t(unpaired,go),Next),
            r(Next,[],[],R1,R2).

r(unpaired(3),Stack,Knot,[W|R1],['.'|R2]):-
            msw(e(x),W),
            msw(t(unpaired,stay),Next),
            r(Next,Stack,Knot,R1,R2).

r(push(I),Tail,Knot,[W|R1],['<'|R2]):-
            msw(e(x),W),
        msw(t(push(I)),Next),
            r(Next,[W|Tail],Knot,R1,R2).

r(loop(I),Stack,Knot,[W|R1],['.'|R2]):-
            msw(e(x),W),
            msw(t(loop(I)),Next),
            r(Next,Stack,Knot,R1,R2).

r(pseudo,Stack,Knot,[W|R1],['i'|R2]):-
            msw(e(x),W),
            msw(t(pseudo),Next),
            r(Next,Stack,[W|Knot],R1,R2).

r(pull,[],Knot,R1,R2):-
        r(unpaired(3),[],Knot,R1,R2).

r(pull,Stack,Knot,[W|R1],['>'|R2]):-
            get_stack(Stack,Head,Tail),
            msw(e(Head),W),
            msw(t(pull),Next),
            r(Next,Tail,Knot,R1,R2).

r(knot,Stack,[],[W|R1],['.'|R2]):-
            msw(e(x),W),
        msw(t(knot,go),Next),
            r(Next,Stack,[],R1,R2).

r(knot,Stack,Knot,[W|R1],['j'|R2]):-
            get_stack(Knot,Head,Tail),
            msw(e(Head),W),
            msw(t(knot,stay),Next),
            r(Next,Stack,Tail,R1,R2).

r(end,[],[],[],[]).

get_stack([Head|Tail],Head,Tail).
```

(a)

```
values(t(unpaired),[push(1),unpaired(5)]).
values(t(push(1)),[loop(1),push(1)]).
values(t(loop(1)),[kissing,loop(1)]).
values(t(kissing),[loop(2),kissing]).
values(t(loop(2)),[pull,loop(2)]).
values(t(unpaired,stay),[push(2),unpaired(3)]).
values(t(push(2)),[loop(3),push(2)]).
values(t(loop(3)),[kissed,loop(3)]).
values(t(unpaired,go),[end,unpaired(3)]).
values(e(_),[a,c,g,u]).

model(Dot_bracket,Seq):-
            r(unpaired(5),[],[],Dot_bracket,Seq).

r(unpaired(5),Stack,Kiss,['.'|R1],[W|R2]):-
            msw(e(x),W),
        msw(t(unpaired),Next),
            r(Next,Stack,Kiss,R1,R2).

r(unpaired(3),[],[],['.'|R1],[W|R2]):-
            msw(e(x),W),
        msw(t(unpaired,go),Next),
            r(Next,[],[],R1,R2).

r(unpaired(3),Stack,Kiss,['.'|R1],[W|R2]):-
            msw(e(x),W),
        msw(t(unpaired,stay),Next),
            r(Next,Stack,Kiss,R1,R2).

r(push(I),Tail,Kiss,['<'|R1],[W|R2]):-
            msw(e(x),W),
        msw(t(push(I)),Next),
            r(Next,[W|Tail],Kiss,R1,R2).

r(loop(I),Stack,Kiss,['.'|R1],[W|R2]):-
            msw(e(x),W),
        msw(t(loop(I)),Next),
            r(Next,Stack,Kiss,R1,R2).

r(kissing,Stack,Kiss,['i'|R1],[W|R2]):-
            msw(e(x),W),
            msw(t(kissing),Next),
            r(Next,Stack,[W|Kiss],R1,R2).

r(pull,[],Kiss,R1,R2):-
           r(unpaired(3),[],Kiss,R1,R2).

r(pull,Stack,Kiss,['>'|R1],[W|R2]):-
            get_stack(Stack,Head,Tail),
            msw(e(Head),W),
            r(pull,Tail,Kiss,R1,R2).

r(kissed,Stack,[],['.'|R1],[W|R2]):-
            msw(e(x),W),
            r(loop(2),Stack,[],R1,R2).

r(kissed,Stack,Kiss,['j'|R1],[W|R2]):-
            get_stack(Kiss,Head,Tail),
            msw(e(Head),W),
            r(kissed,Stack,Tail,R1,R2).

r(end,[],[],[],[]).

get_stack([Head|Tail],Head,Tail).
```

(b)

Figure 9: PRISM code of HMM based pseudoknot and kissing hairpins models

```
% mRNA.psm

% rna-parameters:
values(scfg_t(begin),[b(f),l,b(x)]).
values(scfg_t(x),[b(x),end]).
values(scfg_t(f),[b(f),l,end]).
values(scfg_t(l),[l,b(o)]).
values(scfg_t(o),[b(o),l,r]).
values(scfg_t(r),[r,b(o)]).

% mm.psm parameters:
values(hmm_t(start),[state(i),state(1)]).
values(hmm_t(i,_,_),[state(i)]).
values(hmm_t(1,_,_),[state(2)]).
values(hmm_t(2,_,_),[state(3)]).
values(hmm_t(3,_,_),[state(1)]).

values(hmm_e(_),[a,c,g,t]).

values(scfg_e(_),[a,c,g,t]).

values(agree(_,_),[a,c,g,t]).

model(O):-
      msw(hmm_t(start),S),
      msw(scfg_t(begin),T),
      p(T,[],S,(b,b),O).

p(end,_,_,_,[]).

p(r,[],state(S),(Y,Z),R):-
      msw(scfg_t(f),NNT),
      p(NNT,[],state(S),(Y,Z),R).

p(Nt,St,state(S),(Y,Z),[W|R]):-
      scfg(Nt,St,ENt,XNt),
      hmm(S,(Y,Z),ES),
      msw(hmm_e(ES),S_w),
      msw(scfg_e(ENt),Nt_w),
      msw(agree(S_w,Nt_w),W),
      msw(hmm_t(S,Y,Z),NS),
      msw(scfg_t(XNt),NNT),
      stack(Nt,St,W,NSt),
      p(NNT,NSt,NS,(Z,W),R).

scfg(b(T),St,b,T).
scfg(r,[],b,f).
scfg(r,[H|_],H,r).
scfg(l,_,l,l).

stack(l,St,W,[W|St]).
stack(r,[],_,[]).
stack(r,[_|T],_,T).
stack(b(_),St,_,St).

hmm(1,(Y,Z),(1,1,1)).
hmm(2,(Y,Z),(2,1,Z)).
hmm(3,(Y,Z),(3,Y,Z)).

hmm(i,(Y,Z),(i,i,i)).
```

Figure 10: PRISM code of an HMM based mRNA model

# Appendix:
# PRISM Source Code for Additional Models

## pair_hmm.psm

```
% parameters:
values(emission(indel),[a,c,g,t]).
values(emission(substitution),[(a,a),(a,c),(a,g),(a,t),
                               (c,a),(c,c),(c,g),(c,t),
                               (g,a),(g,c),(g,g),(g,t),
                               (t,a),(t,c),(t,g),(t,t)]).

values(transition(begin),[substitution,insert,delete]).
values(transition(substitution),[substitution,insert,delete,end]).
values(transition(insert),[substitution,insert,end]).
values(transition(delete),[substitution,delete,end]).

% recursion:
recursion(substitution,[Xi | R1],[Yi | R2]):-
   msw(emission(substitution),(Xi,Yi)),
   msw(transition(substitution),Next_state),
   recursion(Next_state,R1,R2).

recursion(insert,[Xi | R1],Seq2):-
   msw(emission(indel),Xi),
   msw(transition(insert),Next_state),
   recursion(Next_state,R1,Seq2).

recursion(delete,Seq1,[Yi | R2]):-
   msw(emission(indel),Yi),
   msw(transition(delete),Next_state),
   recursion(Next_state,Seq1,R2).

% initiation:
model(Seq1,Seq2):-
   msw(transition(begin),Next_state),
   recursion(Next_state,Seq1,Seq2).

% termination:
recursion(end,[],[]).
```

## triple_hmm.psm

```
values(trans(begin),[xyz,xy,xz,yz,x,y,z]).
values(trans(xyz),[xyz,xy,xz,yz,x,y,z,end]).
values(trans(xy),[xyz,xy,xz,yz,x,y,end]).
values(trans(xz),[xyz,xy,xz,yz,x,z,end]).
values(trans(yz),[xyz,xy,xz,yz,y,z,end]).
values(trans(x),[xyz,xy,xz,x,end]).
values(trans(y),[xyz,xy,yz,y,end]).
values(trans(z),[xyz,xz,yz,z,end]).

values(triple_emit,[(a,a,a),(a,a,c),(a,a,g),(a,a,t),
                    (a,c,a),(a,c,c),(a,c,g),(a,c,t),
                    (a,g,a),(a,g,c), (a,g,g), (a,g,t),
                    (a,t,a),(a,t,c),(a,t,g),(a,t,t),
                    (c,a,a),(c,a,c),(c,a,g),(c,a,t),
                    (c,c,a),(c,c,c),(c,c,g),(c,c,t),
                    (c,g,a),(c,g,c),(c,g,g),(c,g,t),
                    (c,t,a),(c,t,c),(c,t,g),(c,t,t),
                    (g,a,a),(g,a,c),(g,a,g),(g,a,t),
                    (g,c,a),(g,c,c),(g,c,g),(g,c,t),
                    (g,g,a),(g,g,c),(g,g,g),(g,g,t),
                    (g,t,a),(g,t,c),(g,t,g),(g,t,t),
                    (t,a,a),(t,a,c),(t,a,g),(t,a,t),
                    (t,c,a),(t,c,c),(t,c,g),(t,c,t),
                    (t,g,a),(t,g,c),(t,g,g),(t,g,t),
                    (t,t,a),(t,t,c),(t,t,g),(t,t,t)]).

values(double_emit,[(a,a),(a,c),(a,g),(a,t),
                    (c,a),(c,c),(c,g),(c,t),
                    (g,a),(g,c),(g,g),(g,t),
                    (t,a),(t,c),(t,g),(t,t)]).

values(single_emit,[a,c,g,t]).

model(Q1,Q2,Q3):-
   msw(trans(begin),S),
   parse(S,Q1,Q2,Q3).

parse(xyz,[L1 | Q1], [L2 | Q2], [L3 | Q3]):-
   msw(triple_emit,(L1,L2,L3)),
   msw(trans(xyz),S),
   parse(S,Q1,Q2,Q3).

parse(xy,[L1 | Q1], [L2 | Q2],Q3):-
   msw(double_emit,(L1,L2)),
   msw(trans(xy),S),
   parse(S,Q1,Q2,Q3).

parse(xz,[L1 | Q1],Q2,[L3 | Q3]):-
   msw(double_emit,(L1,L3)),
   msw(trans(xz),S),
   parse(S,Q1,Q2,Q3).

parse(yz,Q1,[L2 | Q2],[L3 | Q3]):-
   msw(double_emit,(L2,L3)),
   msw(trans(yz),S),
   parse(S,Q1,Q2,Q3).
parse(x, [L1 | Q1],Q2,Q3):-
   msw(single_emit,L1),
   msw(trans(x),S),
   parse(S,Q1,Q2,Q3).

parse(y,Q1,[L2 | Q2],Q3):-
   msw(single_emit,L2),
   msw(trans(y),S),
   parse(S,Q1,Q2,Q3).

parse(z,Q1,Q2,[L3 | Q3]):-
   msw(single_emit,L3),
   msw(trans(z),S),
   parse(S,Q1,Q2,Q3).

parse(end,[],[],[]).
```

## pair_codon_hmm.psm

```
values(trans(begin),[m1,x1,y1]).
values(trans(m1),[m2]).
values(trans(m2),[m3]).
values(trans(m3),[m1,x1,y1,end]).
values(trans(x1),[x2]).
values(trans(x2),[x3]).
values(trans(x3),[m1,x1,end]).
values(trans(y1),[y2]).
values(trans(y2),[y3]).
values(trans(y3),[m1,y1,end]).

values(emit(m1),[(a,a),(a,c),(a,g),(a,t),
                 (c,a),(c,c),(c,g),(c,t),
                 (g,a),(g,c),(g,g),(g,t),
                 (t,a),(t,c),(t,g),(t,t)]).

values(emit(m2),[(a,a),(a,c),(a,g),(a,t),
                 (c,a),(c,c),(c,g),(c,t),
                 (g,a),(g,c),(g,g),(g,t),
                 (t,a),(t,c),(t,g),(t,t)]).

values(emit(m3),[(a,a),(a,c),(a,g),(a,t),
                 (c,a),(c,c),(c,g),(c,t),
                 (g,a),(g,c),(g,g),(g,t),
                 (t,a),(t,c),(t,g),(t,t)]).

values(emit(x1),[a,c,g,t]).
values(emit(x2),[a,c,g,t]).
values(emit(x3),[a,c,g,t]).

values(emit(y1),[a,c,g,t]).
values(emit(y2),[a,c,g,t]).
values(emit(y3),[a,c,g,t]).

model(Q1,Q2):-
   msw(trans(begin),S),
   parse(S,Q1,Q2).


parse(m1,[L1 | Q1], [L2 | Q2]):-
   msw(emit(m1),(L1,L2)),
   msw(trans(m1),S),
   parse(S,Q1,Q2).

parse(x1,[L1 | Q1], Q2):-
   msw(emit(x1),L1),
   msw(trans(x1),S),
   parse(S,Q1,Q2).

parse(y1,Q1,[L2 | Q2]):-
   msw(emit(y1),L2),
   msw(trans(y1),S),
   parse(S,Q1,Q2).
parse(m2,[L1 | Q1], [L2 | Q2]):-
   msw(emit(m2),(L1,L2)),
   msw(trans(m2),S),
   parse(S,Q1,Q2).

parse(x2,[L1 | Q1], Q2):-
   msw(emit(x2),L1),
   msw(trans(x2),S),
   parse(S,Q1,Q2).

parse(y2,Q1,[L2 | Q2]):-
   msw(emit(y2),L2),
   msw(trans(y2),S),
   parse(S,Q1,Q2).

parse(m3,[L1 | Q1], [L2 | Q2]):-
   msw(emit(m3),(L1,L2)),
   msw(trans(m3),S),
   parse(S,Q1,Q2).

parse(x3,[L1 | Q1], Q2):-
   msw(emit(x3),L1),
   msw(trans(x3),S),
   parse(S,Q1,Q2).

parse(y3,Q1,[L2 | Q2]):-
   msw(emit(y3),L2),
   msw(trans(y2),S),
   parse(S,Q1,Q2).

parse(end,[],[]).
```

## scfg.psm

```
values(transition(free),[[n,free],[stem,free],[n]]).
values(transition(stem),[[l,stem,r],[l,leftbulge,stem,r],[l,stem,rightbulge,r],[loop]]).
values(transition(loop),[[n,loop],[n]]).
values(transition(leftbulge),[[n,leftbulge],[n]]).
values(transition(rightbulge),[[n,rightbulge],[n]]).

values(emission(l),['<']).
values(emission(r),['>']).
values(emission(n),['.']).

model(L):-
   scfg(free,L-[]).

scfg(LHS,L0-L1):-
   (
   preterminal(LHS)->
   msw(emission(LHS),Symbol),
   L0=[Symbol|L1]
   ;
   msw(transition(LHS),RHS),
   projection(RHS,L0-L1)
   ).

projection([],L-L).

projection([RHS_head | RHS_tail],L0-L2):-
   scfg(RHS_head,L0-L1),
   projection(RHS_tail,L1-L2)
        .

preterminal(r).
preterminal(l).
preterminal(n).
```

## mm_scfg_hmm.psm

```
% parameters:
values(scfg_t(begin),[b(f),l]).
values(scfg_t(f,_,_),[b(f),l,end]).
values(scfg_t(l,_,_),[l,b(o)]).
values(scfg_t(o,_,_),[b(o),l,r]).
values(scfg_t(r,_,_),[r,b(o)]).

values(scfg_e(_,_,_),[a,c,g,t]).

% initiation:
model(O):-
   msw(scfg_t(begin),T),
   p(T,[],(b,b),O).

% termination:
p(end,_,_,[]).

% stem termination:
p(r,[],(X,Y),R):-
   msw(scfg_t(f,X,Y),NNT),
   p(NNT,[],(X,Y),R).

% recursion:
p(Nt,St,(X,Y),[W | R]):-
   scfg(Nt,St,ENt,XNt),
   msw(scfg_e(ENt,X,Y),W),
   msw(scfg_t(XNt,X,Y),NNT),
   stack(Nt,St,W,NSt),
   p(NNT,NSt,(Y,W),R).

scfg(b(T),St,b,T).
scfg(r,[H|_],H,r).
scfg(l,_,l,l).

stack(l,St,W,[W|St]).
stack(r,[],_,[]).
stack(r,[_|T],_,T).
stack(b(_),St,_,St).
```

## pair_scfg.psm

```
values(transition(free),[[(n,n),free],[stem,free],[(n,n)]]).
values(transition(stem),[
                        [(a,a),stem,(t,t)],
                        [(a,c),stem,(t,g)],
                        [(a,g),stem,(t,c)],
                        [(a,t),stem,(t,a)],

                        [(c,a),stem,(g,t)],
                        [(c,c),stem,(g,g)],
                        [(c,g),stem,(g,c)],
                        [(c,t),stem,(g,a)],

                        [(g,a),stem,(c,t)],
                        [(g,c),stem,(c,g)],
                        [(g,g),stem,(c,c)],
                        [(g,t),stem,(c,a)],

                        [(t,a),stem,(a,t)],
                        [(t,c),stem,(a,g)],
                        [(t,g),stem,(a,c)],
                        [(t,t),stem,(a,a)],

                        [leftbulge,stem,(n,n)],
                        [(n,n),stem,rightbulge],
                        [loop]]).

values(transition(loop),[[(n,n),loop],[(n,n)]]).
values(transition(leftbulge),[[(n,n),leftbulge],[(n,n)]]).
values(transition(rightbulge),[[(n,n),rightbulge],[(n,n)]]).

values(transition(b),[m,i,d]).
values(transition(m),[m,i,d]).
values(transition(i),[m,i,d]).
values(transition(d),[m,i,d]).

values(emission(a),[a,c,g,t]).
values(emission(c),[a,c,g,t]).
values(emission(g),[a,c,g,t]).
values(emission(t),[a,c,g,t]).
values(emission(n),[a,c,g,t]).

model(L1,L2):-
   scfg(b,free,L1-[],L2-[]).

scfg(b,LHS,L10-L11,L20-L21):-
   msw(transition(b),S),
   scfg(S,LHS,L10-L11,L20-L21).

scfg(m,LHS,L10-L11,L20-L21):-
   (
   preterminal(LHS)->
   get_tuple(LHS,(X,Y)),
   msw(emission(X),Symbol_1),
   msw(emission(Y),Symbol_2),
   L10=[Symbol_1|L11],
   L20=[Symbol_2|L21]
   ;
   msw(transition(m),S),
   msw(transition(LHS),RHS),
   projection(S,RHS,L10-L11,L20-L21)
   ).

scfg(i,LHS,L10-L11,L20-L21):-
   (
   preterminal(LHS)->
   get_tuple(LHS,(X,Y)),
   msw(emission(Y),Symbol_2),
   L10=L11,
   L20=[Symbol_2|L21]
   ;
   msw(transition(i),S),
   msw(transition(LHS),RHS),
   projection(S,RHS,L10-L11,L20-L21)
   ).

scfg(d,LHS,L10-L11,L20-L21):-
   (
   preterminal(LHS)->
   get_tuple(LHS,(X,Y)),
   msw(emission(X),Symbol_1),
   L10=[Symbol_1|L11],
   L20=L21
   ;
   msw(transition(d),S),
   msw(transition(LHS),RHS),
   projection(S,RHS,L10-L11,L20-L21)
   ).

get_tuple((X,Y),(X,Y)).

projection(_,[],L1-L1,L2-L2).

projection(S,[RHS_head | RHS_tail],L10-L12,L20-L22):-
   scfg(S,RHS_head,L10-L11,L20-L21),
   projection(S,RHS_tail,L11-L12,L21-L22)
         .
preterminal((a,a)).
preterminal((a,c)).
preterminal((a,g)).
preterminal((a,t)).
preterminal((c,a)).
preterminal((c,c)).
preterminal((c,g)).
preterminal((c,t)).
preterminal((g,a)).
preterminal((g,c)).
preterminal((g,g)).
preterminal((g,t)).
preterminal((t,a)).
preterminal((t,c)).
preterminal((t,g)).
preterminal((t,t)).
preterminal((n,n)).

% enforcing canonical stem base-pairing:
:-fix_sw(emission(a),[0.999997,0.000001,0.000001,0.000001]).
:-fix_sw(emission(c),[0.000001,0.999997,0.000001,0.000001]).
:-fix_sw(emission(g),[0.000001,0.000001,0.999997,0.000001]).
:-fix_sw(emission(t),[0.000001,0.000001,0.000001,0.999997]).
```

## adph_scfg_hmm.psm

```
% parameters:
values(t(begin),[unpaired(5)]).
values(t(5),[push]).
values(t(push),[unpaired(loop)]).
values(t(loop),[pull]).
values(t(pullout),[unpaired(3)]).
values(t(pull),[pull]).
values(t(3),[end]).
values(e(unpaired),['.']).
values(e(push),[('<','>')]).
values(adph,[s,g]).

% initiation:
model(Obs):-
        msw(t(begin),NS),
        r(NS,adph(1),[],Obs).

% termination:
r(end,_,_,[]).

% recursions:
r(unpaired(X),adph(S),Stack,[W | R]):-
   msw(e(unpaired),W),
   msw(adph,SG),
   adph(S,SG,A),
   r(unpaired(X),A,Stack,R).

r(unpaired(X),next,Stack,[W | R]):-
   msw(e(unpaired),W),
   msw(t(X),NS),
   r(NS,adph(1),Stack,R).

r(push,adph(S),Tail,[Left | R]):-
   msw(e(push),(Left,Right)),
   msw(adph,SG),
   adph(S,SG,A),
   r(push,A,[Right|Tail],R).

r(push,next,Tail,[Left | R]):-
   msw(e(push),(Left,Right)),
   msw(t(push),NS),
   r(NS,adph(1),[Right|Tail],R).

r(pull,A,[],R):-
   msw(t(pullout),NS),
   r(NS,A,[],R).

r(pull,A,Stack,[Head | R]):-
   get_stack(Stack,Head,Tail),
   msw(t(pull),NS),
   r(NS,A,Tail,R).

get_stack([Head | Tail],Head,Tail).

adph(1,s,adph(1)).
adph(1,g,adph(2)).
adph(2,s,adph(2)).
adph(2,g,adph(3)).
adph(3,s,adph(3)).
adph(3,g,next).
```

## factorial_hmm.psm

```
% parameters:
values(emission(_,_),[a,c,g,t]).

values(transition(_),[state(1),state(2),end]).

% recursion:
recursion(state(S1),state(S2),[Xi | Rest_of_sequence]):-
   msw(emission(S1,S2),Xi),
   msw(transition(S1),Next_state_S1),
   msw(transition(S1),Next_state_S2),
   recursion(Next_state_S1,Next_state_S2,Rest_of_sequence).

% initiation:
model(Sequence):-
   msw(transition(begin),Next_state_S1),
   msw(transition(begin),Next_state_S2),
   recursion(Next_state_S1,Next_state_S2,Sequence).

% termination:
recursion(end,_,[]).
recursion(_,end,[]).
```

## mm_frames.psm

```
% parameters:
values(transition(_,_,_,i),[1,2,3,i]).
values(transition(_,_,_,1),[1,2,3,i]).
values(transition(_,_,_,2),[1,2,3,i]).
values(transition(_,_,_,3),[1,2,3,i]).

values(transition(_,_,_,j),[4,5,6,j]).

values(transition(_,_,_,4),[4,5,6,j]).
values(transition(_,_,_,5),[4,5,6,j]).
values(transition(_,_,_,6),[4,5,6,j]).

values(continue,[yes,no]).

values(emission(_,_,_,_),[a,c,g,t]).

% initiation:
model(O):-
   recursion(yes,state(i,i,i,j,j,j),x,y,O).

%termination:
recursion(no,_,_,_,[]).

%recursion:
recursion(yes,state(F1,F2,F3,F4,F5,F6),P2,P1,[X | [Y| [ Z | R]]]):-
   msw(emission(P2,P1,F1,F4),X),
   msw(emission(P1,X,F2,F5),Y),
   msw(emission(X,Y,F3,F6),Z),
   msw(transition(P2,P1,X,F1),NF1),
   msw(transition(P2,P1,X,F4),NF4),
   msw(transition(P1,X,Y,F2),NF2),
   msw(transition(P1,X,Y,F5),NF5),
   msw(transition(X,Y,Z,F3),NF3),
   msw(transition(X,Y,Z,F6),NF6),
   msw(continue,C),
   recursion(C,state(NF1,NF2,NF3,NF4,NF5,NF6),Y,Z,R).
```

## factorial_scfg_hmm.psm

```
values(t(scfg),[b(f),l]).
values(t(f),[b(f),l,end]).
values(t(l),[l,b(lb),b(o)]).
values(t(lb),[b(lb),l]).
values(t(o),[b(o),r]).
values(t(r),[r,b(rb)]).
values(t(rb),[b(rb),r]).

values(e(_,_),[a,c,g,t]).

model(O):-
   msw(t(scfg),Nt_1),
   msw(t(scfg),Nt_2),
   p(Nt_1,[],Nt_2,[],O).

p(end,_,_,_,[]).
p(_,_,end,_,[]).

p(Nt_1,St_1,Nt_2,St_2,[W | R]):-
   scfg(Nt_1,St_1,Pt_1,TNt_1),
   scfg(Nt_2,St_2,Pt_2,TNt_2),
   msw(e(Pt_1,Pt_2),W),
   msw(t(TNt_1),NNT_1),
   msw(t(TNt_2),NNT_2),
   stack(Nt_1,St_1,W,NSt_1),
   stack(Nt_2,St_2,W,NSt_2),
   p(NNT_1,NSt_1,NNT_2,NSt_2,R).

scfg(b(T),L,b,T).
scfg(r,[],b,f).
scfg(r,[H|T],H,r).
scfg(l,St,l,l).

stack(l,St,W,[W | St]).
stack(r,[],W,[]).
stack(r,[H | T],W,T).
stack(b(T),St,W,St).
```

## scfg_mrna.psm

```
% SCFG parameters:
values(transition(free),[[n,free],[stem,free],[n]]).
values(transition(stem),[[a,stem,t],
                         [c,stem,g],
                         [g,stem,c],
                         [t,stem,a],
                         [leftbulge,stem,n],
                         [n,stem,rightbulge],
                         [loop]]).
values(transition(loop),[[n,loop],[n]]).
values(transition(leftbulge),[[n,leftbulge],[n]]).
values(transition(rightbulge),[[n,rightbulge],[n]]).

% HMM parameters:
values(transition(i),[(a,t,g),(c,t,g),(t,t,g),i]).

values(transition((t,a,a)),[i]).
values(transition((t,a,g)),[i]).
values(transition((t,g,a)),[i]).

values(transition((_,_,_)),[(a,a,a),(c,a,a),(g,a,a),(t,a,a),
                             (a,c,a),(c,c,a),(g,c,a),(t,c,a),
                             (a,g,a),(c,g,a),(g,g,a),(t,g,a),
                             (a,t,a),(c,t,a),(g,t,a),(t,t,a),

                             (a,a,c),(c,a,c),(g,a,c),(t,a,c),
                             (a,c,c),(c,c,c),(g,c,c),(t,c,c),
                             (a,g,c),(c,g,c),(g,g,c),(t,g,c),
                             (a,t,c),(c,t,c),(g,t,c),(t,t,c),

                             (a,a,g),(c,a,g),(g,a,g),(t,a,g),
                             (a,c,g),(c,c,g),(g,c,g),(t,c,g),
                             (a,g,g),(c,g,g),(g,g,g),(t,g,g),
                             (a,t,g),(c,t,g),(g,t,g),(t,t,g),

                             (a,a,t),(c,a,t),(g,a,t),(t,a,t),
                             (a,c,t),(c,c,t),(g,c,t),(t,c,t),
                             (a,g,t),(c,g,t),(g,g,t),(t,g,t),
                             (a,t,t),(c,t,t),(g,t,t),(t,t,t)]).

% joint parameters:
values(emission(_,_),[a,c,g,t]).

% parser:
model(L):-
   scfg(1,(a,t,g),i,free,L-[]).

scfg(State,Tuple,V,LHS,L0-L1):-
   (
   nonterminal(LHS),
   msw(transition(LHS),RHS),
   projection(State,Tuple,V,RHS,L0-L1)
   ;
   preterminal(LHS)->
   msw(emission(LHS,V),W),
   L0=[W|L1]
   ).
```

```
projection(State,Tuple,V,[],L-L).

projection(1,(X,Y,Z),V,[RHS_head | RHS_tail],L0-L2):-
   (
   preterminal(RHS_head)->
   scfg(2,(X,Y,Z),Y,RHS_head,L0-L1),
   projection(2,(X,Y,Z),Y,RHS_tail,L1-L2)
   ;
   scfg(1,(X,Y,Z),V,RHS_head,L0-L1),
   projection(1,(X,Y,Z),V,RHS_tail,L1-L2)
   ).

projection(2,(X,Y,Z),V,[RHS_head | RHS_tail],L0-L2):-
   (
   preterminal(RHS_head)->
   scfg(3,(X,Y,Z),Y,RHS_head,L0-L1),
   projection(3,(X,Y,Z),Y,RHS_tail,L1-L2)
   ;
   scfg(2,(X,Y,Z),V,RHS_head,L0-L1),
   projection(2,(X,Y,Z),V,RHS_tail,L1-L2)
   ).

projection(3,(X,Y,Z),V,[RHS_head | RHS_tail],L0-L2):-
   (
   preterminal(RHS_head)->
   msw(transition((X,Y,Z)),New),
   scfg(1,New,Z,RHS_head,L0-L1),
   projection(1,New,Z,RHS_tail,L1-L2)
   ;
   scfg(3,(X,Y,Z),V,RHS_head,L0-L1),
   projection(3,(X,Y,Z),V,RHS_tail,L1-L2)
   ).

projection(1,i,V,[RHS_head | RHS_tail],L0-L2):-
   (
   preterminal(RHS_head)->
   msw(transition(i),New),
   scfg(1,New,i,RHS_head,L0-L1),
   projection(1,New,i,RHS_tail,L1-L2)
   ;
   scfg(1,i,V,RHS_head,L0-L1),
   projection(1,i,V,RHS_tail,L1-L2)
   ).

nonterminal(free).
nonterminal(stem).
nonterminal(loop).
nonterminal(leftbulge).
nonterminal(rightbulge).

preterminal(a).
preterminal(c).
preterminal(g).
preterminal(t).
preterminal(n).
```

## repeat_hmm.psm

```
% parameters:
values(transition(1),[state(1),push]).
values(transition(push),[push,state(3)]).
values(transition(3),[state(3),pop]).
values(transition(5),[state(5),end]).

values(emission(_),[a,c,g,t]).

% init
model(O):-
   model(1,state(1),[],O).

% recursion:
model(I,state(H),S,[X | R]):-
   msw(emission(H),X),
   msw(transition(H),N),
   model(I,N,S,R).

model(I,push,S,[X | R]):-
   I\=10,
   msw(emission(push),X),
   msw(transition(push),N),
   J is I+1,
   model(J,N,[X | S],R).

% repeat limit
model(10,push,S,R):-
   model(0,state(3),S,R).

model(I,pop,S,R):-
   model(I,empty_stack,S,[],R).

model(I,empty_stack,[],Q,R):-
   model(I,empty_queue,Q,R).

% stack emptying:
model(I,empty_stack,S,Q,R):-
   get_head(S,X,T),
   model(I,empty_stack,T,[X | Q],R).

model(I,empty_queue,[],R):-
   model(I,state(5),[],R).

model(I,empty_queue,Queue,[X | R]):-
   get_head(Queue,X,Tail),
   model(I,empty_queue,Tail,R).

get_head([Head | Tail],Head,Tail).

% termination
model(_,end,[],[]).
```