

Preprocessing for Optimization of Probabilistic-Logic Models for Sequence Analysis

Henning Christiansen and Ole Torp Lassen

Research group PLIS: Programming, Logic and Intelligent Systems
Department of Communication, Business and Information Technologies
Roskilde University, P.O.Box 260, DK-4000 Roskilde, Denmark
E-mail: {henning, otl}@ruc.dk

Abstract. A class of probabilistic-logic models is considered, which increases the expressibility from HMM's and SCFG's regular and context-free languages to, in principle, Turing complete languages. In general, such models are computationally far too complex for direct use, so optimization by pruning and approximation are needed. The first steps are taken towards a methodology for optimizing such models by approximations using auxiliary models for preprocessing or splitting them into submodels. Evaluation of such approximating models is challenging as authoritative test data may be sparse. On the other hand, the original complex models may be used for generating artificial evaluation data by efficient sampling, which can be used in the evaluation, although it does not constitute a foolproof test procedure. These models and evaluation processes are illustrated in the PRISM system developed by other authors, and we discuss their applicability and limitations.

1 Introduction

Models for data analysis are often based on probability theory which provides a firm theoretical basis and a catalogue of well-understood computational methods, which may be exact or approximative. Hidden Markov Models (HMM) and Stochastic Context-Free Grammars (SCFG) are well-known and popular techniques for analysis of genomic sequences in biology and other comprehensive sequential data sets. Both models consist of a logical and a probabilistic part, where the logical part of a HMM is a finite state automaton and for a SCFG, a Context-Free Grammar; see [7] for overview and detailed references. The logical part defines a space of possible analysis results, and the probabilistic part assigns a probability to each such with the understanding that high probability is a good or close-to-actual-truth result. There exist efficient Viterbi algorithms for HMM's that can find the state path of highest probability for a given sequence in linear time, whereas SCFG requires cubic time to identify a best parse tree.

There is a strong interest in the automated extraction of information from genomic and other biological sequence data due to demanding applications in medicine, biological sciences, food industry, etc..., and there are basically two complementary ways to approach this problem. Firstly, the computational power

can be increased (e.g., by huge clusters of computers). Secondly, and this is the direction that we pursue, more powerful and detailed models of higher formal expressibility can be developed as a basis for novel and more sophisticated analyses. This may increase computational complexity drastically, and such new models must be accompanied by optimizations in order to gain practical value.

Our main goal with the present work is to promote the application of new strong models defined in declarative languages such as extensions to PROLOG. This provides the familiar advantages of declarative programming; it is known that standard models such as HMM and SCFG are embedded in natural ways, they can be extended and combined in a very flexible manner, and long-distance context-sensitive dependencies can be modelled using logical variables and arbitrary auxiliary data-structures and predicates. While this extends up to in principle Turing complete languages, even the cubic complexity that arises for a model that incorporate elements of a SCFG is problematic for long sequences. As a working example, we experiment with optimising and approximating an essentially context-free model to reduce its overall complexity. The method under investigation quite likely extends to even more complex problems but our current experiments does not go beyond context-free features.

While most of our constructions do not reflect the actual domain, our primary interest is on applications in computational biology, and our analysis takes into account the particular restrictions imposed by this. We focus currently on models that can be expressed in the PRISM system [16, 17], but our general framework is independent of the particular formalism used.

We suggest that it would be advantageous for probabilistic-logic models to be developed by initially producing models that represent all the available knowledge about the phenomena in focus as faithfully as possible, exploiting the advantages of a powerful modelling language, and without being limited by implementation issues. Such ideal and declarative models may be used for initial testing of very small data sequences, and as representation of intellectual knowledge. Furthermore, they can be used for generating artificial samples of sequences-with-annotations which may be useful for investigating the models and for testing as we suggest below. Finally, these models can serve as specifications for, and standard of, the development of other, approximating models which can be used for efficient analysis of real data. The work described here represents the early steps taken towards this overall end, focussing on general methodological efficiency rather than on domain specific accuracy.

These efforts are part of a larger research project concerned with logic-statistical models that involves computer scientists, biologists, bioinformaticists, developers of competitive software for bioinformatic applications, and a leading company producing biological cultures to the food industry world-wide (the LoSt project [14]). Another big challenge in this project, that is not touched upon in the present paper, is to learn how to apply these models for real biological problems.

We consider in the present paper approximations based on preprocessing. For example, an efficiently implemented HMM may be used as a preprocessor

to provide “steering marks” for a more advanced analysis. In fact, this is quite similar to tagging and period separation in natural language analysis.

Section 2 defines what we mean by a probabilistic-logic annotation model and states our assumptions about them; section 3 explains how such models can be defined and executed in PRISM as this is the context for the present experiments. In section 4, we define pre-annotation models and explain how they can be utilized for optimized execution for prediction. Section 5 describes the particular problems involved in evaluating the quality of an approximating analysis independently of any specific domain of application. Section 6 describes an experiment that defines a simplified model for analysis of genomic sequences together with an approximating model; we consider also the suitability for these examples of evaluation based on sampling and suggest some improvements of the basic principle. Finally follows a short review of related work and conclusions.

2 Probabilistic-Logic Annotation Models

We consider probabilistic models that describe relationships between observed data and annotations that capture hidden information or “semantics” embedded in the data, and the intention is to use such models for computing the most probable annotations for given sequences.

In our motivating applications, S will be a sequence of letters, but other sorts of data structures may fit with the formal definitions as well; for simplicity of usage, we continue to refer to the S argument as a sequence. A is an annotation represented as a list identifying, say, the start and end positions for genes in S or a more detailed level of introns and exons etc., depending on the biological researcher’s choice. An annotation may also represent an entire parse tree in case of a model that capture sophisticated features such as secondary or, in a proper setting, tertiary RNA-structures. In section 6 we will describe a model of a particular kind of secondary RNA structure.

Definition 1. *An annotation model $m = \langle L, P \rangle$ consists of a logical part L which is a set of ground atoms of a predicate $m(A, S)$ and a probabilistic part P which is a probability distribution over L , i.e., $0 \leq P(C) \leq 1$ for any $C \in L$ and $\sum_{C \in L} P(C) = 1$. For given S , we let $P(S) = \sum_A P(A, S)$.*

Whenever $m(A, S) \in L$, we say that A is an annotation of S ; when, for no $m(A', S) \in L$ that $P(m(A', S)) > P(m(A, S))$, we say that A is a best annotation of S .

We refer to the process of finding a best annotation for given sequences as *prediction*.

The use of probabilistic models for prediction is based on an assumption that there is a correlation between high probability and good score with standard measurements such as precision and recall; however, the latter is hypothetical when no authoritative test data are available, which is typically the case in computational biology.

3 Probabilistic-Logic Models in the PRISM System

PRISM is a powerful system developed by other authors [16, 17] for working with a particular sort of probabilistic-logic models, based on an extension to Prolog with discrete random variables, called multi-valued switches. Consider as an example the following declaration,

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

It creates a switch, which means that whenever `msw(letter,X)` is called within the execution of a program, a value among `[a,c,g,t]` is assigned to `X`, understood as the outcome of a random variable which is independent of any other such variable, including other calls to the same switch. Switch declarations can also be parametrized as shown in the example below.

As shown by [15], an assignment of probabilities to the possible outcomes of each switch induces a probability distribution over program's least Herbrand model, and thus fits with our definition above; (a program needs to satisfy a few natural restrictions for this to be true, but this is of no concern here).

Example 1. We define here a simple HMM as a PRISM program; it has internal states `s1` and `s2` and a special `end` state; `s1` is also the start state. Each state emits probabilistically one of the same four letters, but potentially with different probabilities. We extend it to an annotation model that records a history of states that were visited during the creation of a given sequence.

```
values(letter(_state), [a,c,g,t]).
values(next_state(_state), [s0,s1,end]).
```

```
hmm(A,S):- hmm(s1,A,S).
hmm(end, [], []).
hmm(Q, [Q|Qs], [L|S]):-
    msw(letter(Q),L), msw(next_state(Q), Q1),
    hmm(Q1,Qs,S).
```

Probabilities can be set explicitly or by means of learning algorithms built into PRISM; we ignore these details here and assume simply that some fixed probabilities are given. As an example, PRISM's semantics assigns the following probability to the atom `hmm([s1,s2], [a,a])`,

$$P(\text{msw}(\text{letter}(s1), a)) \times P(\text{msw}(\text{next_state}(s1), s2)) \times \\ P(\text{msw}(\text{letter}(s2), a)) \times P(\text{msw}(\text{next_state}(s2), \text{end})).$$

Stochastic context-free grammars can be defined in an equally concise way as the HMM in example 1 above by having a switch for each nonterminal whose values represent the rules for that nonterminal. It is straightforward to extend a grammar so that it builds annotations in the shape of parse trees or mark-ups of specific subsequences. PRISM includes devices for calculating probabilities of

a given atom as well as generalized Viterbi algorithms that make it possible to obtain the best annotation for given sequence. The time complexity for these varies with the inherent complexity of the given model as well as the degrees of freedom in the specific queries; up to certain sequence lengths, Viterbi computations for HMM’s are linear in the length of the sequence and for SCFG’s cubic which is known to be theoretically best (however involving a substantial constant factor).

Programs with annotations as the one shown in example 1 above should not be given directly to PRISM for Viterbi computations, as it will lead to a combinatorial explosion. We will not go into explaining why this is the case, but simply indicate that it is possible to run Viterbi in linear time for HMM programs with annotations taken out, and then reconstruct the annotations from a proof tree produced by PRISM; [5] explains a tool based on automatic program annotations so that the PRISM model developer does not have to care about these subtleties.

4 Optimization by Pre-Annotations

A central notion in our approach is that of a pre-annotation, which restricts the class of possible annotations for a given sequence.

Definition 2. *Given an annotation model $m = \langle L, P \rangle$, a pre-annotation model is a model $m^{pre} = \langle L^{pre}, P^{pre} \rangle$ equipped with a projection function π such that for any $m(A, S) \in L$, there is a unique pre-annotation A^{pre} with $m^{pre}(A^{pre}, S) \in L^{pre}$ and $\pi(A) = A^{pre}$.*

A best annotation of S given pre-annotation A^{pre} , is an annotation A with $\pi(A) = A^{pre}$ for which there is no other A' with $\pi(A') = A^{pre}$ and $P(m(A', S)) > P(m(A, S))$.

In practice, a pre-annotation can be given as an additional argument to the predicate of a model or as a partially instantiated argument.

Pre-annotations may provide ways to improve the time complexity of prediction. This may be achieved when the time complexities of both

- calculating a best pre-annotation A^{pre} and
- calculating a best annotation A given A^{pre}

are strictly less than the time complexity of calculating the best annotation A^* for S in m . However, for this to be useful, the possible deviations between A and A^* as well as between $P(A, S)$ and $P(A^*, S)$ must not be too large; we return later to the question of how to characterize and estimate this accuracy.

We can make analogies to natural language processing techniques for showing different kinds of pre-annotations. Tagging means to assign most likely word classes and inflections to each word in a text, such that the parser can start from there, analyzing sentence structures without bothering about these details. Another preprocessing is identification of the beginning and end of periods. For

example, some occurrences of “.” indicates end of a period while others serve other purposes, e.g., as in “e.g.”. It is interesting to observe that HMM’s are often used for tagging and period segmentation; see, e.g., [9].

Splitting a sequence into fixed subsequences that are then analyzed separately can be an effective way to reduce time complexity.

Example 2. Sequence analysis using SCFG is known in general to be of cubic complexity, and as we noted above, this can be obtained in PRISM. Often a SCFG has a natural distinction of subsequences covered by a subset of rules; let us refer to those as periods and assume they are limited by an upper length k .

Assuming now that we have an HMM which can produce reliable preannotations that splits the sequence into (proposed) periods of lengths at most k . While analysis of a sequence of length n with the SCFG takes time $\mathcal{O}(n^3)$, the use of the HMM to fix the periods which are then analyzed separately by the SCFG, reducing the time to $\mathcal{O}(n/k \cdot k^3) = \mathcal{O}(n)$ since, in all practical cases, k is much smaller than n .

In other words, suppose we can represent a complex model as a SCFG of the form (probabilities omitted):

$$S \rightarrow A S \mid B S \mid \epsilon$$

$A \rightarrow$ production rules for A -periods

...

$B \rightarrow$ production rules for B -periods

...

such that only the A -rules involves features requiring context-free analysis, while the rules for B -periods in principle could be stated as a HMM instead. Then M can simply be reformulated as the interaction of three interacting submodels:

1. a HMM for S of states A and B each initiating the corresponding submodel,
2. a SCFG for A -type periods, and
3. a HMM for B -type periods.

This sort of two-stage analysis is straightforward to implement in PRISM using two successive calls to Viterbi predicates.

5 Estimating the Quality of an Approximating Model

5.1 Difficulties

Testing of tools for biological sequence analysis is a particularly difficult task, as independent test data with trusted annotations are rare, and thus standard tests based on recall and precision measurements are generally not possible.

This problem comes from the fact that it is very expensive and time consuming to perform the necessary laboratory work in order to produce a correct

annotation of a single sequence. Furthermore, the sparse trusted annotated data are often used as training data for the model in question, which also makes them dubious as test data. But when data are available, we can refer to the role of thumb of splitting known annotated data into training and validation data in order to avoid over-fitting.

As hinted in the introduction, we suggest that when comparing a complex model, m^c , that we may call canonical, and its approximating (and assumed efficiently executable) model m^a , we can use m^c for generating artificial, annotated test data for m^a .

In this way, we abstract away the real world and may compare the two models in an objective way. However, this sort of probabilistic sampling of pairs, $\langle ann, seq \rangle$, of annotation and sequence introduces another problem, namely that ann is not necessarily the annotation of highest probability for seq . In our experiments, we have often seen that even the approximating model can generate annotations of a higher probability (measured in m^c for comparison). The hard truth is that we have no way of checking whether a generated annotation really is a good one or how far it is from a best one. With sampling in this very big outcome space, it is in practice impossible to hit the same sequence more than once, so it will be an illusion to refer to the law of big numbers to get a picture of the distribution for the annotations of a given sequence. To make things even more difficult, we have also observed that there is very little correlation between the length of a sequence and the probability distribution for the annotations of a given sequence. Even within the same model, some sequences have a single obvious best annotation while others are highly ambiguous and can present multitudes of almost equally good ones, with correspondingly small probability mass left for each. In other words, it seems it is not possible to produce a meaningful aggregation of collections of sampled data and their probabilities.

5.2 Considering Generated Samples for Evaluation

While samples with annotations generated from a canonical model cannot take the role of affirmative test data, we will investigate what we are able to conclude from such testing.

Sampling has the advantage that one can start the computer, have thousands of samples generated and tested with the approximating model. Let us consider a single sample $\langle ann^{samp}, seq \rangle$ generated from the canonical model m^c . We can run seq through the approximative model in question, m^a , and obtain its best annotation ann^{approx} as described above.

To compare the two, we can measure their probabilities $P^{samp} = P^c(ann^{samp}, seq)$ and $P^{approx} = P^c(ann^{approx}, seq)$; notice that we measure both in the distribution of the canonical model in order obtain measurements in the same scale. A ratio P^{approx}/P^{samp} close to one indicate that the quality of the two annotations are similar.

As we have discussed, using precision and recall is not always possible in a general setting, so we may instead utilize a subjective measurement of the similarity between the two annotations.

In our experiment that follows, we apply a simple principle for measuring similarity that may apply independently of actual sort of annotations in questions. Each annotation is mapped into a sequence of symbols of the same length as the sequence, indicating the findings of interest, and similarity is defined by the fraction of all such symbol that are identical for the two sequences. When the nested structures are important, the symbol sequence may indicate the structure using brackets; for example, two tree structures may be mapped into “[-(-){-(-)}]” and “[-(-){(-)-}]” that are identical in 8 of 12 character, i.e., a similarity measure of 0.667. When only the classification of particular subsequences is interesting, e.g., distinguishing between genes and non-genes, this sort of measurement still gives score to annotation that differs slightly in the begin and end positions of the subsequences. So the similarity between “nnnnnnnggggggggggnn” and “nnnnnnnggggggggggn” is 0.95.

6 A Simple Genefinder in PRISM and its Evaluation by Sampling

We show here an example of a pair of canonical and approximative model that follows the pattern of example 2. The canonical model m^c is a simplified SCFG that resembles the sort of models that one will expect for genomic sequences for prokaryotic organisms. It distinguishes between subsequences considered as coding (genes) and non-coding; the non-coding parts are modelled basically by an HMM whereas rules for the coding parts include description of a particular kind of secondary RNA-structure called *hairpins* see figure 1, these hairpins do not manifest themselves in the actual genome but in the mRNA produced by the genes; so a good match with such structures could be perceived as indicating the likeliness of a gene. Because hairpins are inherent nested structures, a SCFG is necessary to describe them.

An approximating model m^a is comprised by an HMM that fixes the boundaries between coding and noncoding regions, and then applies different submodels of m^c for the each of the two kinds of subsequences.

As seen in example 2, m^c requires cubic times whereas m^a is linear. The difference in accuracy between the models can be explained as follows: if m^c is applied (hypothetically!) for prediction, it has the degree of freedom to move the coding/non-coding boundaries in order to get the optimal hairpin structure, whereas m^a fixes these boundaries first from a shallow analysis, and then finds the best analyses for the subsequences irrespective of their context.

We indicate here the details of the models and investigate how far we can get in an evaluation using the sampling strategy.

6.1 Overview of the models

The canonical model is described in two levels, a top level that is structured as a HMM with two states, for coding and non-coding regions; instead of emitting a single letter as a HMM, a call is made to a the relevant submodel. The submodel

for noncoding regions is another HMM and the one for coding regions is a SCFG involving the hairpin structures indicated above.

A gene is initiated by a start codon and terminated by a stop codon¹, which is one of $\{\langle a, t, g \rangle, \langle g, t, g \rangle, \langle t, t, g \rangle\}$ and of $\{\langle t, a, a \rangle, \langle t, g, a \rangle, \langle t, a, g \rangle\}$, respectively; but there is no guarantee that a start-stop codon pair actually indicates a coding region. Furthermore, the possible lengths of coding regions are multiples of 3 as to fit with a codon structure. The hairpin structures in the coding regions of this model represent foldings in RNA that occur due to an attraction between the molecules represented by a and t , and between c and g . So for example, the subsequence $agata\dots tatct$ may describe a hairpin, where the outermost 5+5 letters form a stem and those indicated by the dots form a loop at the top; see illustration below; for simplicity we do not allow nested (cacti-like) hairpin structures, although these occur frequently in nature.

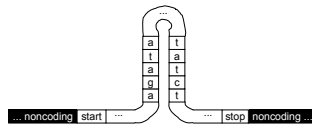


Fig. 1. *Hairpins* or *hairpin loops* consists of a *stem* of mutually abstracted letters and a *loop* of unpaired letters

The first phase of the approximating model m^a uses a HMM that includes start and stop codons, the multiple of 3 condition for coding regions and with potentially different distributions of letters between coding and non-coding regions. As indicated above (in section 4), this model is used for fixing a proposal for the boundaries of coding/non-coding and then the two submodels inherited from m^c takes over.

6.2 Evaluation by Sampling and some Improvements of the Method

When generating samples with m^c , we have made an *ad hoc* improvement of the annotations which is possible for models such as m^c due to its clear subdivision into subsequences: for each such subsequence, we run a Viterbi computation with the relevant submodel and put this best sub-parse into the annotation instead of the sampled one. Be aware that this is different from the approximating model,

¹ A *codon* corresponds to a triplet of three letters that, depending on context, may code for an amino acid, or serve a “control purpose” as start and stop codon.

as the sequences as well as boundary parts of the annotations are created from the unaltered m^c . We can measure, in the total probability, a clear improvement with this trick so our tests reported below may be a bit more accurate in sorting out bad annotations produced by m^c . However, the problem that sampled annotations are suboptimal is decreased but not eliminated.

We conducted four experiments, whose results are summarized in Fig. 2 and 3. Everywhere we measured the probabilities in log space, so the ratio is represented as a difference with 0 representing identity, and also the Hamming-like similarity measure (referred to as match percentages in the figures) which takes into account the deep syntactic structures of the individual parses, as indicated above. The scatterplots in Fig. 2 correlates these two measures; a dot represents a probability ratio together with match percentage; notice that dots to the left of the zero line represent cases where the m^a found a more probable annotations than the one given by the improved m^c sampling.

Experiment a). Firstly we used uniform probabilities for all switches in the model and no selection among the samples. As could be expected, the combined plot indicates a diffuse distribution with the larger part of the results indicating that m^a provides the best annotations measured in probability. Thus we have no clear indication of how good m^a is compared with the objectively best samples (which, by nature of the setting, are unknown), and we may thus also doubt the value of the results where m^a provides a results close to the sampled one. However, the combined scatter plot is a bit misleading as dots may coincide, and the detailed plots indicate that this is the case, as most measurements are close to, or spot on, the ideal 0 resp. 100% marks.

Experiments b), c) and d). We changed the basic experiment in two directions in order to see if we could get different results when the models are made more realistic by 1) training the models using the 100 shortest annotated genes from E. Coli K12, whose lengths are between 50 and 178 letters; and/or 2) we constrained generated samples to those satisfying the inherent length constraint implied by the training data.

From Fig. 2 we see that training has a profoundly positive effect on the similarity of parses, and that constraining the samples to comply with the inherent length constraints of the domain of application affects the probability quotients of the individual analyses similarly. Thus experiment d) represents a much higher degree of correlation with far less occurrences of m^a suggesting annotations with higher probabilities than those provided by improved sampling; this may tempt us to have more confidence to all sampled annotations and thus more confidence to an approximated annotation close to the sampled one. Both Fig. 2 and 3 indicate here that most approximated and sampled annotations correlates closely. Even with these deviations, the two measures correlates perfectly in the vast majority of cases, coinciding in (0, 100%) coordinates, as shown in Fig 3. All in all this indicates that especially with the precautions taken in experiment d), we may trust to the approximating model produce reasonably reliable results.

These experiments showed that the sampling based tests provide a clear indication of the quality of an approximating model; it is also clear that the method works best for models with biased probabilities (so both m^a as predictor as m^c as generator are better to distinguish between good and bad, so to speak). Throwing away samples that do not respect the inherent constraints that also are expected in actual data to be analyzed, removes irrelevant observations from the statistics expressed in the diagrams; this also contributes to the improved reliability of the tests. The more critical issues of this testing method is the lack of quantitative summaries based on firm statistical considerations of how good the approximation is. In the completely general setting with no specific domain of application and sufficiently annotated data, we doubt that such quantitative measures can be devised.

7 Related Work

Bayesian networks, HMM's, and SCFG's are traditional methods for sequence analysis that can be seen as instances of probabilistic-logic models; while their flexibility for modelling and formal expressibility are far below the models we are aiming at (PRISM and similar), there exists a plethora of efficient algorithms and implemented systems; see [7] for background and overview. These provide a catalogue of possible preprocessors to be used within our approach

More general and powerful formalisms have been suggested as extensions to logic programs or equally expressive formalisms within the last 15 years, we may mention PRISM [16, 17] that we have exemplified, Stochastic Logic Programs [13], Stochastic functional Programs [10], and Relational Bayesian Networks [8].

There is a growing interest in such models for bioinformatical applications. In particular [3, 1] discuss applications to Systems Biology, but also various kinds of sequence analysis have been studied (see [6] for an excellent survey.

We may refer to [4] as a precursor of the present work, where similar ideas are applied for a comparative test of three different genefinder programs [2, 11, 12]. A detailed PRISM model was built for parts of human genomic sequences, it was trained with known data, and then the trained model was used for producing artificial genomic sequences (complemented with manual editing for the parts not covered by the developed model). These data were used as test data, and the quality of the genefinders was evaluated with precision and recall measures. In this work, preprocessing analogously to what has been described in the present paper was used to produce auxiliary annotations for speeding up training.

8 Conclusion and Future Work

We advocate the use of probabilistic-logic models based on logic programs (or similarly expressive languages) as the basis for analysis of biological sequence data; due to flexibility and generality, such models are candidates for providing

better and more detailed finds than the currently most used methods, however, this is still to be proved.

We intend that such models should be developed without consideration about performance in order to document in a formal way and as faithfully as possible, the available knowledge about the phenomena being modelled in what we called a canonical model.

Using preprocessors, e.g., based on existing and efficiently implemented technologies, as a way to reach realistic execution times, we intend to get the best of both worlds, flexibility and sophistication of the probabilistic-logic models combined with feasible execution times.

Implementation involving preprocessing to fix certain choices before a detailed analysis takes places, necessarily affects the accuracy, and we have discussed the possible ways of testing such implementation in cases when no authoritative test data is available. In the lack of such, and despite certain problems, the best thing we can do seems to be to employ the possibility of using the complex model for generation of annotated test data, and then compare with the annotations produced by the implemented approximative models. While comparing probabilities for the two annotations can give some indication, we found it best to use subjectively defined measures that compares the similarity between the sequences in a straightforward, syntactic fashion.

We noticed that the sampling method provide the best indications when the model has biased probabilities, as it is easier to distinguish between good and bad annotations. It also increases the chance that the annotations produced under sampling are of a reasonable quality. We noticed also the advantage of applying inherent constraints that are difficult to capture in probabilistic models, to sort out the relevant samples and run the comparison tests on those only. These constraints may typically concern the length of particular kinds of substrings, where sampling will produce annotated sequences that do not reflect nature (or the possible training data).

We intend to extend the methodology with a more firm statistically basis for evaluation of the measurements produced by the sampling principle.

Acknowledgement: This work is supported by the project “Logic-statistic modelling and analysis of biological sequence data” funded by the NABIIT program under the Danish Strategic Research Council, and the CONTROL project, funded by Danish Natural Science Research Council.

References

1. Marenglen Biba, Stefano Ferilli, Nicola Di Mauro, and Teresa Maria Altomare Basile. A hybrid symbolic-statistical approach to modeling metabolic networks. In Bruno Apolloni, Robert J. Howlett, and Lakhmi C. Jain, editors, *KES (1)*, volume 4692 of *Lecture Notes in Computer Science*, pages 132–139. Springer, 2007.
2. Chris Burge and Samuel Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997.

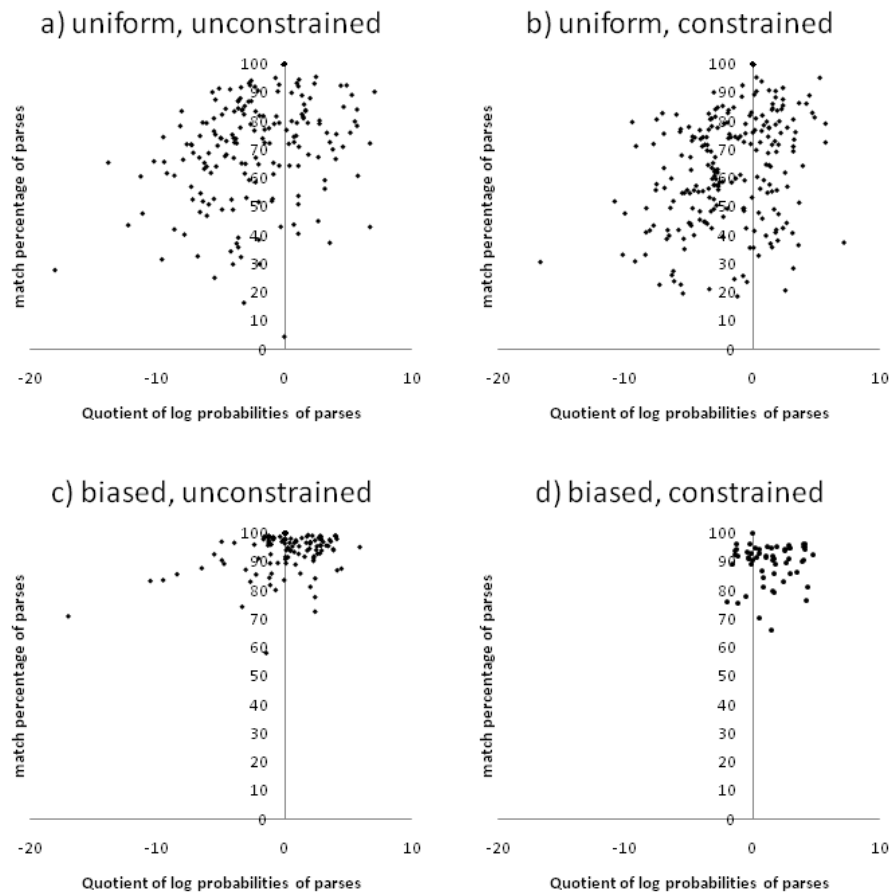


Fig. 2. Representation of the sampling distributions of the four conducted experiment. Along the X-axis are logarithmic probability quotients of the canonical vs. approximative analyses. A quotient of 0 represents identity. Along the Y-axis are the corresponding percentages of Hamming-like similarity between the two analyses (match percentages). Dots to the left of the Y-axis represent cases where the approximative analysis resulted in a more probable annotations than the one resulting from the improved canonical sampling. In *a)* is shown the plot for uniform models and unconstrained sampling. *b)* is the result of constraining samples to comply with the length range of the 100 shortest genes of E.Coli. In *c)* the models were trained using EM-learning on the aforementioned data from E.Coli. Finally, the plot in *d)* represents the analyses of samples from the trained model complying with the length constraints inherited from the training data. The plots in *a)* and *b)* illustrates the chaos of uniform parameters. The effect of training the models is easily observable in *c)*, causing the approximative and canonical analyses to agree to a much higher degree than in *a)* or *b)*. In *d)* we see that forcing domain-specific constraints on the sampling increases the correlation between analysis and probability.

run	probability quotient = 0	match percentage = 100
a)	823/1000	822/1000
b)	839/1000	837/1000
c)	867/1000	865/1000
d)	947/1000	946/1000

Fig. 3. The degree of perfect correlation between canonical and approximative analyses according to the probability-quotient measure and the match-percentage measure respectively. This represents the number of dots that coincides in the coordinates (0,100%) of the scatter plots in Fig 2. Individually, both training and constraining increases the degree of correlation in both measures, but combining training and constraining results in a profound increase of about 10% in both measures

3. Jianzhong Chen, Stephen Muggleton, and Jose Santos. Abductive stochastic logic programs for metabolic network inhibition learning. In Paolo Frasconi, Kristian Kersting, and Koji Tsuda, editors, *MLG*, 2007.
4. Henning Christiansen and Christina Mackeprang Dahmcke. A machine learning approach to test data generation: A case study in evaluation of gene finders. In Petra Perner, editor, *MLDM*, volume 4571 of *Lecture Notes in Computer Science*, pages 742–755. Springer, 2007.
5. Henning Christiansen and John Gallagher. Mode-based slicing and its applications, 2009. submitted.
6. De Raedt, L., Frasconi, P., Kersting, K., and Muggleton, S. (Eds.). *Probabilistic Inductive Logic Programming*. Springer, 2008.
7. Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.
8. Manfred Jaeger. Relational bayesian networks. In Dan Geiger and Prakash P. Shenoy, editors, *UAI*, pages 266–273. Morgan Kaufmann, 1997.
9. Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.
10. Daphne Koller, David A. McAllester, and Avi Pfeffer. Effective bayesian inference for stochastic programs. In *AAAI/IAAI*, pages 740–747, 1997.
11. Anders Krogh. Using database matches with for HMMGene for automated gene detection in Drosophila. *Genome Research*, 10(4):523–528, 2000.
12. A. Lukashin and M. Borodovsky. Genemark.hmm: new solutions for gene finding. *Nucleic Acids Research*, 26(4):1107–1115, 1998.
13. Stephen Muggleton. Learning from positive data. In Stephen Muggleton, editor, *Inductive Logic Programming Workshop*, volume 1314 of *Lecture Notes in Computer Science*, pages 358–376. Springer, 1996.
14. LoSt on the Web. <http://lost.ruc.dk>.
15. Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *ICLP*, pages 715–729, 1995.
16. Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *J. Artif. Intell. Res. (JAIR)*, 15:391–454, 2001.
17. Taisuke Sato and Yoshitaka Kameya. Statistical abduction with tabulation. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond*, volume 2408 of *Lecture Notes in Computer Science*, pages 567–587. Springer, 2002.