Compositionality in probabilistic logic modelling for biological sequence analysis

Ole Torp Lassen

NOVEMBER 2011 ROSKILDE UNIVERSITY COMPUTER SCIENCE RESEARCH REPORT #135

Ole Torp Lassen

Computer Science
Department of Communication,
Business and Information Technologies
Roskilde University
P. O. Box 260
DK–4000 Roskilde
Denmark

Telephone: +45 4674 3839
Telefax: +45 4674 3072
Internet: http://www.ruc.dk/dat_en/
E-mail: datalogi@ruc.dk

ISSN 0109–9779

Research reports are available electronically from:

http://www.ruc.dk/dat_en/research/reports/

Compositionality in probabilistic logic modelling for biological sequence analysis

by

Ole Torp Lassen

A dissertation presented to the faculties
of Roskilde University in partial fulfillment of
the requirement for the PhD degree

Assessment committee:
Troels Andreasen, CBIT, Roskilde University (chairman)
Veronica Dahl, Simon Fraser University, Canada
Anders Krogh, University of Copenhagen

Supervisor: Henning Christiansen, Roskilde University

Department of Communication, Business and Information Technologies
Roskilde University, Denmark

Submitted 29. July 2011

Contents

Abstract

The dissertation concerns the efficient application of probabilistic logic programming – in particular the PRISM system – to problems involving biological sequence data. Specifically, I consider DNA-feature-annotation in general and gene-finding in bacterial DNA in particular. A compositional framework for complex sequence annotation, *Bayesian Annotation Networks*, inspired form traditional Bayesian Networks is developed. In this new framework, *probabilistic annotation models* assumes the role of conditional probability tables for quantifying dependencies between individual DNA-features and how they are analysed. I also develop approximative methods for inference with the framework and apply it to several non-trivial DNA-annotation tasks and show that computational complexity can be constrained to the complexity of the most complex constituent. The inherent modularity of the approach allows for easy experimentation with alternative combinations of putative signals for gene-finding in DNA and also allows local dependencies to be exploited for training and prediction. The dissertation is divided in three main parts: Part I introduces the different scientific domains involved in the interdisciplinary study – molecular biology, sequence analysis, machine-learning – and also provides a formal introduction to the PRISM system, that is used for example programs throughout the dissertation. Part II presents the proposed methodology for approximation and optimization through problem decomposition as a means for increasing efficiency of complex annotation tasks. The proposed modelling paradigm, *Bayesian Annotation Networks*, is introduced and formally defined as the main original contribution of the dissertation, and two examples of employing the paradigm to problems from bioinformatics are presented and their respective strengths and weaknesses are discussed. Finally, in Part lll, I present related and future work and present my conclusions.

Resumé

Denne afhandling omhandler effektiv anvendelse af probabilsitisk logikprogrammering – og især PRISM systemet – til analyse af DNA-sekvenser. Jeg fokuserer på annotering af egenskaber ved, og aspekter af DNA i almindelighed og gen-annottering i særdeleshed Der udvikles og dokumenteres et kompositionelt system, kaldet *Bayesian Annotation Networks*, til brug for sammensatte analyse- og annotterings-problemer. I dette system, der kan ses som en variant af traditionelle Bayesianske netværk, kvantificeres de forskellige afhængigheds-relationer mellem individuelle aspekter v.h.a. *probabilistiske annotations modeller* i stedet for de traditionelle betingede sandsynligheds fordelinger. Der udvikles yderligere approksimerende algoritmer for inferens med BAN'er som eksemplificeres på flere ikke-trivielle DNA-annoterings problemer. Det demonstreres at den samlede kompleksitet kan begrænses til kompleksiteten af den mest kompleske konstituent. Systemets modulære opbygning egner sig umiddelabrt godt til experimenter med potentielle signaler og kombinationer af samme med henblik på for eksempel detektion af gener i DNA. Samme modlaritet muliggør yderligere udnyttelese af lokale afhængigheder mellem del-analyser i forbindlese med inferens. Afhandlingen har tre hoved-dele: Part I introducerer de videnskabelige domæner der vedrører afhandlingen – molekylær biologi, formel sprogteori, machine-learning samt en formel introduktion til PRISM systemet, der anvendes til imlementering af eksempler igennem hele afhandlingen. Part II præsenterer den foreslåede metodologi til approksimereing og optimering via problem-dekomposition som indgangsvinkel til effektiv behandling af sammensatte annotaterings-problemer Afhandliongens hovedbiddrag, *Bayesian Annotation Networks*, introduceres og defineres, og to eksempler p andvendelse beskrives og diskuteres. Til slut, i Part III, opsummeres relateret forskning og mulige retninger for videre arbejde, samt afhandlingens konklussion.

Acknowledgements

The present work is partly the result of my close collaboration with a number of people involved in the LoSt projekt (Logic Statistical Modelling for Biological Sequence Analysis). It has been a challenging and fun experience. First of all, I would like to thank my supervisor Henning Christiansen for inspiration, support and patience over the period. I would also like to thank my colleagues in the project group at Roskilde Christian Theil Have, Matthieu Petit, Ole Skovgaard, Søren Mørk, Ana Capatana for useful discussions, hard work and team spirit. I would also like to thank Klaus Skaalum Lassen, Ursula Krogh, Sine Zamback, Jens Ulrik Hansen, Chistian Theil Have and Henning Christiansen for proofreading and commenting on the thesis in various versions of completion. Also Anders Krogh, Manfred Jaeger, Nicos Angelopoulos, Yoshitaka Kameya, Luc de Raedt and Maurice Bruynooghe have provided useful support and feedback on various occasions throughout the work. In addition, I would thank my computer science colleagues at Roskilde University especially Sine Zamback, Jens Ulrik Hansen, Mai Ajspur for their friendship and all the fun times we had.

Finally, I would extend my thanks to my girlfriend Birgitte Krogh for putting up with me for the past six months where I have admittedly been somewhat preoccupied around the house.

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.

Chapter 1

Introduction

1.1 Project aims and purposes

The thesis proposed in the dissertation concerns the efficient application of probabilistic logic programming to biological sequence analysis. In particular, it is the aim to research the extent to which PRISM, a probabilistic extension to the logic programming language Prolog, is suited for developing competitively efficient programs for automatic detection of genes in bacterial DNA.

Recently developed technologies for extremely efficient automatic sequencing of genomic material (see [70] for an early review) have caused a veritable explosion in the amount of raw DNA to be analysed and annotated, and the need for efficient and accurate automatic methods for DNA annotation is more pressing than ever.

Automatic methods for predicting and annotating likely genes in DNA have been researched at least since Roger Staden [71] in the early 1980s designed a computer program to recognise the distinct structural patterns formed by tRNA genes in DNA. In general, the best current programs for gene-finding, i.e., *genefinders*, [41, 6, 63], are able to detect up to about 90% of the known genes in well studied bacterial genomes like Escherichia Coli. About 10%, however, prove to be extremely resilient to automatic detection. Furthermore, this accuracy is relative to a set of accepted reference genes for that organism that is almost certanly not complete. Even in extremely well studied reference genomes, like that of Escherichia Coli, there are likely to be genes that have not yet been identified and annotated, and thus missing from the reference. The real proportion of genes that are correctly detected by state of the art genefinders could therefore be significantly smaller than 90%. This is supported by recent research [75] indicating, that

a large number of primarily short genes are missing from current reference databases of bacterial genomes.

These observations all contribute to the growing need for considering new ways of regarding the problem of efficient and accurate DNA annotation in general and gene finding in particular.

For the past 15 years there has been a growing interest and activity in the research of how to combine logic programming, probability theory and machine learning into unified modelling languages and formalisms and several probabilistic programming languages that extends classical Prolog with the capability of probabilistic inference have emerged, see for example [26] for an excellent survey. The inherited declarative strengths of such systems may prove to be useful for rapid prototyping and experimental model design, not only for DNA-annotation but for complex sequence analysis in general.

1.2 The LoSt-project

The research documented here has been carried out as an integral part of the LoSt-project, "Logic-statistic modelling and analysis of biological sequence data" [52], funded by the NABIIT program under the Danish Strategic Research Council.

The project has involved a number of people from both academia and industry to varying degrees including: *Henning Christiansen*, *John Peter Gallagher*, *Ole Skovgaard*, *Mathieu Petit*, *Ana Capatana*, *Søren Mørk*, and *Christian Theil Have* from Roskilde University. *Manfred Jaeger* from Ålborg University, *Taisuke Sato* and *Yoshitaka Kameya* from Tokyo Institute of Technology, and *Anders Krogh* from University of Copenhagen as well as specialists from industrial companies *CLC-bio* and *Christian A. Hansen* have also been involved in the project as a whole.

The bulk of the present work is the result of my close collaboration with *Henning Christiansen*, *Matthieu Petit* and *Christian Theil Have* at Roskilde University. The synergy in this group of people has been of such a vibrant and cross-fertilizing nature, that it is practically impossible to discern any single responsibility for most of the published work (which is also why we consistently denote authorship of co-authored publications alphabetically rather than in order of contribution). In each case, one or two persons had primary responsibility but in all cases each co-author contributed extensively to the content either directly or by continuous discussion and experimentation.

The overall goal of the LoSt project is to experiment with one probabilistic logic programming language in particular, namely the Prolog-based PRISM system [64, 65], as a platform for developing efficient, accurate and flexible tools for DNA-annotation. Being declarative, PRISM and similar languages cater for rapid prototyping and clear and concise programs especially by

- increased expressive power - programs in these languages are generally shorter and more concise than implementations in procedural languages,
- executable problem specifications - once a problem has been properly specified, that specification functions as a working implementation for the solution of the problem,
- clear and consistent semantics based on classical 1^{st} order logic.

We believe that these strengths allow for systematic experimentation with - and comparison of - novel or putative gene-signals in DNA and benefit the development of new and powerful DNA-annotation systems. There are however also challenges with regard to efficiency of computation when using Prolog and PRISM, and the overall research question of the LoSt project can be formulated as follows:

How to implement clear, flexible and efficient systems using PRISM for accurate DNA-annotation?

Three possible directions of research for the LoSt project were originally identified:

- Can proper pre-processing of data increase efficiency for example by identifying regions of special interest and thus avoid unnecessary analysis?
- Can the underlying implementation of a PRISM-program be optimized, for example based on automatic program analysis?
- Can we apply automatic program analysis to devise an automatic system for source to source transformation of PRISM programs (to for example C++)?

My chosen focus was on pre-processing and in particular on compositional aspects of DNA-analysis, where a complex overall task is optimized or approximated by identifying and negotiating constituent subtasks and, in turn, integrating their analytical results. The motivating research question of my research within the Lost-project can thus be formulated as:

Can we establish a system of compositionality for probabilistic annotation programs in PRISM that retains the strengths of declarative programming but keeps computational complexity low enough for practical application?

Compositionality of sequence analysis

I present in this work a general, systematic and very flexible methodology, *Bayesian Annotation Networks*, for integrating annotations from separate and independent sub-models and attempt to evaluate their relative importance to the overall analysis. The main methodology is based on two publications in particular:

(2009) *Preprocessing for optimization of probabilistic-logic models for sequence analysis* [21], authored by Henning Christiansen and myself.

(2011) *Bayesian Annotation Networks for Complex Sequence Analysis* [18], authored by Henning Christiansen, Christian Theil Have, Matthieu Petit and myself.

The first one [21] concerns early experiments with a systematic method for decomposing a complex sequence model into simpler constituents. We first attempt to identify the different analytical tasks as being either simple or demanding in terms of computational complexity. We then decompose the overall model accordingly, i.e., into a specialised sub-model for each class of tasks. By means of yet another sub-model, called a *chopper*, for simple pre-analysis of the data-sequence, we attempt to distinguish corresponding types of sub-sequences: those, that require the complex analysis and those, that can make do with the less sophisticated analysis. Each sub-sequence is then submitted to the sophisticated annotation model of its type, and the individual sub-annotations combined to produce an annotation of the original sequence. That is, we follow an algorithm along these lines:

1. Apply the pre-processor to distinguish sub-sequences according to type.

2. Submit each sub-sequence to complex annotation-analysis according to type.

3. Append the annotations.

I am responsible for the general idea, prototype program, the experiments were formulated in dialogue with Søren Mørk and carried out by me while planning, analysis of results, and the published paper itself represents joint work between Henning Christiansen and myself. This work – published also in part in [20] (2008) and [42] (2008) – forms the substrate of chapter 8 of the dissertation.

We gained at least one important insight in particular from these early experiments. Namely, that when we, like this, distinguish first and perform complex analysis later, then the complex analysis obviously has no influence what so ever on the distinction. In many cases, where the preprocessor represents a very simple model without the sophistication of the specialised annotation-model, this corresponds to letting the blind lead the seeing. Therefore, if we want the sophisticated annotation programs to influence the distinction and classification of sub-sequences, we are required to annotate first and distinguish later. This let me to think of the overall model in terms of a tree-like structure (a directed acyclic graph, really) of sub-models, each with their special responsibility be it chopping, feature-analysis, or integrating, and each depending on the annotations of one or more of the others. During my stay in Leuven, I enthusiastically applied myself to the task of programming a prototype system, consisting of a chopper, a program for annotating DNA-conservation (in a manner discussed with Ole Skovgaard and also Ana Capatana prior to my departure), and several parsers for various existing databases and gene-finder results. I also included a simple format for specifying order of executing of sub-programs, passing the results to dependent sub-models and presenting different annotations for visual inspection. Safely returned to Roskilde, the general merits of the approach were recognised by the rest of the LoSt-group and it quickly became the main focus for both me, Christian Theil Have and Matthieu Petit. Other than for visual inspection, however, I had no idea of how to consistently integrate the different annotations of a data-sequence until Henning Christiansen pointed out the structural and logical similarity to classical Bayesian Networks, unfamiliar to me at the time. From this evolved the general methodology of organizing specialized sub-models according to their interdependencies in a topology, that we now call a *Bayesian Annotation Network* (BAN). Matthieu Petit replaced almost all of my initial programs by clever new versions and we all contributed to a growing library of specialised annotation models. My simple format for specifying execution order was replaced by a sophisticated framework, called the *LoSt framework*, for specifying topologies, keeping track of already computed annotations and coordinating probabilities thanks especially to the efforts of Christian Theil

Have. BAN's are the topic of the second paper [18], that I had the main responsibility for though Christian Theil Have contributed significantly to the experimental part of the paper. Both Christian Theil Have and Henning Christiansen contributed to the discussion concerning BAN-evaluation while tentative speculations towards an information theoretical approach to evaluation is my responsibility. I describe general methodology in detail in chapter 6, and the experimental part forms the basis of chapter 7.

Other topics

A number of other publications also resulted from the collaborative work in the LoSt-project, that fell outside the focus of the dissertation:

(2009) *A Constraint Model for Constrained Hidden Markov Models* [15]
(2010) *Inference with constrained hidden Markov models in PRISM* [16]
(2010) *The Viterbi Algorithm expressed in Constraint Handling Rules* [17]
(2011) *Taming the Zoo of Discrete HMM Subspecies & Some of their Relatives* [19]

all authored by Henning Christiansen, Christian Theil Have, Matthieu Petit and myself.

1.3 Dissertation overview

In the course of this very interdisciplinary work, I will make reference to concepts from molecular biology, formal language theory probability theory and machine learning as well as to PRISM and Prolog in general. The rest of Part I of the dissertation is therefore devoted to a short general introduction to each of these domains along with the related terminologies and conventions adopted in the dissertation. Part II will describe in detail the proposed methodology that evolves around the central idea of formulating a complex annotation task in terms of *Bayesian annotation networks* of annotations, I also document and discuss experiments with the proposed methodology. In Part III, I make reference to future and related work before I sum up my conclusions.

A word on examples, figures and Illustrations

Unless otherwise stated, all examples, figures, and illustrations in this dissertation are of my own devise and to my knowledge not subject to any other intellectual ownership.

Part I

Background

Chapter 2

Biological Sequence Data

In this chapter, I introduce the biological terminology employed throughout the dissertation.
To begin an understanding of the challenges involved in *DNA-annotation* and *gene-finding* (defined below, in section 2.6), it is necessary to also understand the intricate biological relationship between DNA (section 2.3) and genes (section 2.1). To this end, section 2.1 introduces the overall relationships between *Cells, Proteins, Genes* and *DNA*, while the remainder of the chapter provides a more formal terminological definition of individual concepts and processes.

The material is based entirely on existing literature in the field, in particular [3, 22, 77]) and, while going into sufficient detail to form a basis for the thesis presentation and discussion, the reader is respectfully referred to such sources for a deeper and more thorough study of the domain than what is supported here. The illustration of molecular constituents of DNA and their relationships in figure 2.2 (p. 15) is adapted from [3].

2.1 The central "dogma" of molecular biology

Among the most important agents in the chemical processes occurring in all biological organisms are the *proteins*. Proteins are polypeptides consisting of sequences of smaller molecules called *amino acids*.

Proteins are vital for the proper functioning of any living organism and they are produced in the cells, (see [3] or similar for a thorough treatment of cells in biology). Each cell in every organism contains, among other things, the DNA (deoxyribonucleic acids) of the organism. DNA is a macromolecule consisting of two *strands*, each representing a sequence of *nucleic-*

acid molecules (also-called a *poly-nucleotide*). The two sequences are in fact each other's complements (see section 2.3), and encode the *genome* of the organism, i.e., all the hereditary information in the organism – including the genes for all the necessary proteins.
This relationship – sometimes called "the central dogma" of biology – between genes, i.e., DNA and nucleotides, on one side and *gene-products*, e.g., amino acids and proteins, on the other, is defined in terms of two essential intracellular molecular processes, *transcription* (section 2.4) and *translation* (section 2.5), as also sketched in figure 2.1:

1. The gene for the protein is transcribed to a poly-nucleotide called *messenger RNA* or *mRNA*, by a macro-molecule called *RNA-polymerase*.

2. The mRNA is then translated to the polypeptide that makes up the protein, by another macro-molecule, the *ribosome*.

Apart from proteins, *functional RNA* of various types are also considered gene-products. These serve functional purposes in the cell directly after being transcribed and are therefore never translated. Thus, the *ribosome* consists, for example, to a large extent of functional RNA, so-called *ribosomal RNA* or *rRNA*.

A gene that is transcribed and translated in a cell, is said to be *expressed* in the cell, and the combined process from gene to gene product is referred to as *gene expression*. While all cells in the organism contain the entire genome, individual cells specialise according to their specific type and function to *regulate* what genes are expressed in them.

Prokaryotic vs. Eukaryotic organisms

There is an important distinction between two mutually exclusive groups of organisms, namely *eukaryotes* and *prokaryotes*. The eukaryotes encompass, among others, plants, fungi and animals. Primarily bacteriae belongs to the prokaryotic group, but also another sub-group of micro-organisms called *Archaea* belong here. The distinction between eukaryotes and prokaryotes is relevant to DNA annotation for several reasons:

- The number of genes is significantly lower in prokaryotes than in eukaryotes. For example about 4.600 genes is thought to make up the genome of the bacteria Escherichia Coli, while the eukaryotic human genome is estimated to involve about 20.000 protein coding genes.
- The amount of DNA is bigger in eukaryotic genomes than in prokaryotic ones. Escherichia Coli's genome consists of about 4.6 million bp

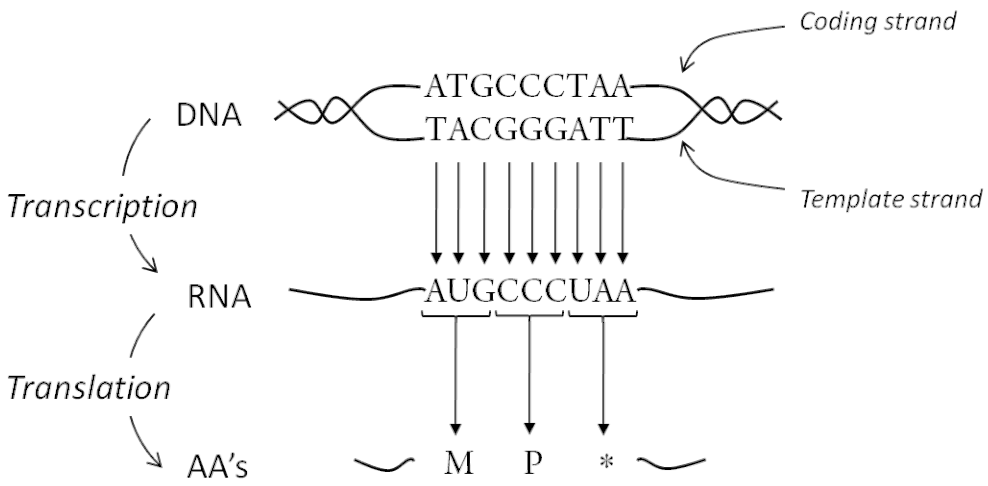


Figure 2.1: Sketch of "the central dogma" of biology. The individual nucleotides (section 2.2) of the template strand of the double-stranded DNA-molecule are first transcribed to RNA by the polymerase (not depicted). The single stranded RNA poly-nucleotide is identical to the coding strand of the DNA, except that all `A`'s are transcribed as `U`'s rather than as `T`'s), as described in section 2.4. The ribosome (not depicted) then translates the RNA – three consecutive nucleotides (a codon) at a time – to a sequence of amino-acids according to a particular genetic code (figure 2.4 on page 22). Translation is described in detail section 2.5.

(*basepairs*, see section 2.2 below) and the largest bacteria genome known has about 9.9 million bp (*Solibacter Usitatus (Ellin6076)*). In contrast, the human genome has about 3.200 million bp and the largest known genome, belonging to an amoeba called *Polychaos Dubium*, has 670,000 millions bp.

- The average gene length also differs between prokaryotes, about 900 bp, and eukaryotes, about 1300 bp, see [78].
- The protein coding genes of eukaryotic organisms only take up about 10% of its entire genome, while in prokaryotic organisms, they take up about 90% of the genome.
- In eukaryotic organisms, the RNA transcript of the DNA is called *pre-mRNA* and undergoes a special process called *splicing* before translation takes place. This process does not occur in prokaryotes. During splicing, entire subsections, referred to as *introns*, are cut out of the pre-mRNA and the remaining subsections, the *exons*, are linked to-

gether in sequence to form the *mature mRNA*, that is then translated to amino-acids. Because the same pre-mRNA may be spliced in several different ways, known as *alternative splicing*, it may define several different genes.

This work focuses on annotation of prokaryotic genomes.

2.2 Nucleotides

DNA (see below) is built from four distinct molecules, adenosine, cytidine, thymidine and guanosine, collectively referred to as *nucleic acids* or *nucleotides*. From a molecular perspective they all consist of three components: A *base*, a central *sugar ring* and a *phosphate*, respectively shown in figure 2.2 in green, blue and yellow. One of four distinct nucleic *bases* (figure 2.2 a)), adenine, cytosine, thymine and guanine, is joined to the first of five carbon atoms in a sugar ring (*ribose*). The carbon atoms in the ribose-ring are conventionally numbered according to the order in which they are positioned in a clockwise direction, i.e., as *1'* through *5'*, counting from the base-connection. Finally, the sugar ring is joined to a phosphate at 5', forming one of four distinct complex nucleotide molecules like the one shown in figure 2.2 b). For convenient reference, individual nucleotides are represented as capital letters A, C, T and G, referring to their respective bases.

Base-pairing

The bases of individual nucleotides also form pairwise bonds, in a process called *base-pairing*. Specifically, T binds to A, and C binds to G, as shown in figure 2.2 c).[1] The pairs *T-A* and *C-G* are called *base-pairs* and T and A are said to be each others *complements*, as are C and G. Base-pairing plays a crucial role in many important processes involving DNA and RNA, including the pairing of the two strands of *DNA*, *transcription* and *translation*, described in sections 2.3, 2.4 and 2.5 below.

[1] This kind of base-pairing, i.e., between A-T and C-G, is called Watson-Crick base-pairing after James D. Watson and Francis Crick who proposed it along with the double helical structure of DNA in 1953 (see [76]). Other kinds do exist, primarily between nucleotides in RNA. They are arguably less important to the general discussion presented in this work and not treated in further detail here.

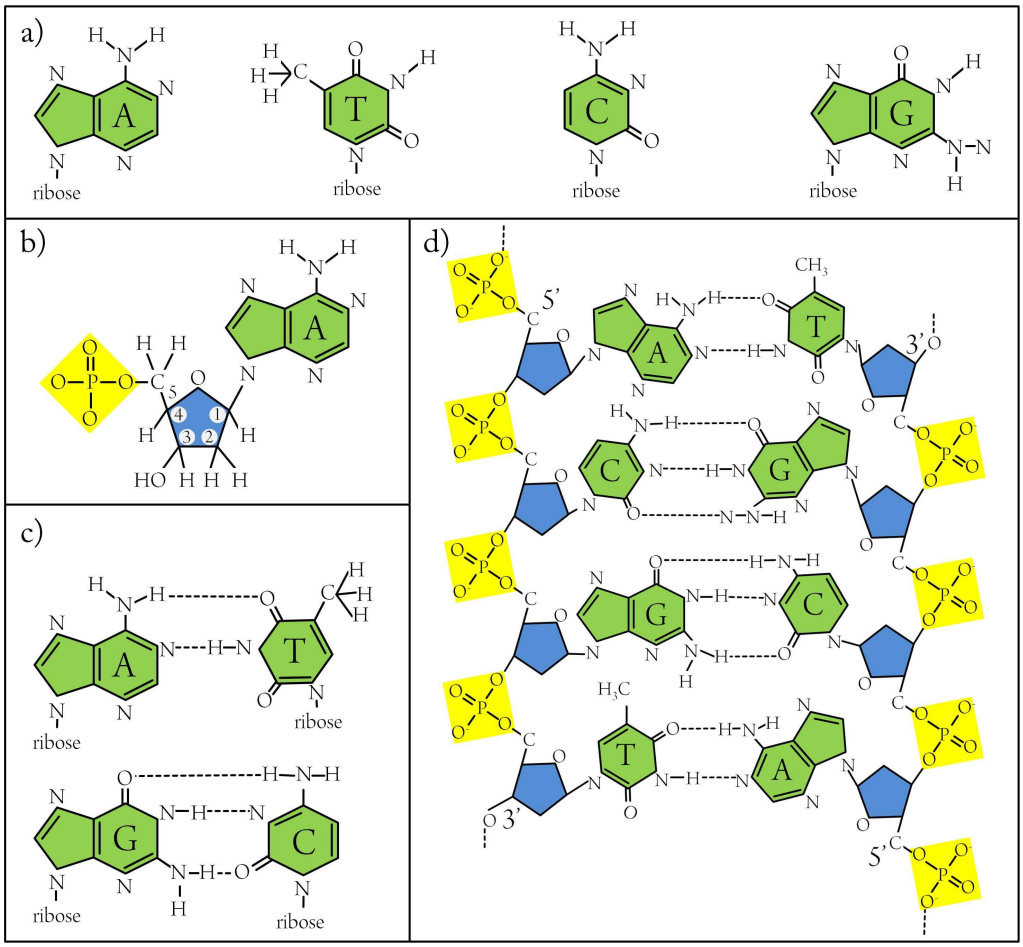


Figure 2.2: The components of DNA and how they combine (adapted from [3]). In **a)** individual *nucleosides adenosine*, *cytidine*, *thymidine* and *guanosine* are shown in green. Each combine with ribose (blue) and phosphate (yellow) to form one of four *nucleotides* or *nucleic acids*, named *adenine* **b)**, *cytosine*, *thymine* and *guanine*. The central ribose-molecule in DNA consists of a 5-ring with oxygen and four carbon atoms. The carbon atoms in the ring are conventionally numbered clockwise from the oxygen i.e., as *1'*, *2'*, *3'* and 4' indicated in **b)**. The nucleoside binds at *1'* and the phosphate binds to a fifth carbon atom of the ribose, numbered *5'*. Base-pairing **c)** is defined as hydrogen bonds between nucleoside components of *A*-*T* and *C*-*G* respectively and does not involve the other components. Base-pairing holds together the two *reversed-complementary* strands of DNA (dotted lines between left and right chains in **d)**). Polynucleotides are chains of nucleotides that are formed when the alcohol *HO* at *3'* in the ribose of one nucleotide binds to the phosphate of another. The direction of polynucleotides are identified by the orientation of the third and fifth carbon atom in the ribose as *5'-3'* vs *3'-5'*, also indicated in **d)**

2.3 DNA

DNA (deoxyribonucleic acid) molecules are *double-stranded* macromolecules consisting of two polynucleotides, called *strands*. Each strand consists of a chain of nucleotides, linked together via bonds between the phosphate of one nucleotide and the *3'* carbon atom of the sugar-ring of another, shown as the chains of yellow and blue components in fig. 2.2 d) (p. 15). One end of each strand is conventionally referred to as the *5'-end* because it terminates with the phosphate at the fifth carbon atom of a nucleotide sugar ring. The other end is called the *3'-end* of the strand, because it terminates with the third carbon atom of a nucleotide sugar ring. Because of the way the DNA was created, *DNA-synthesis* (see for example [3]), the two strands are each others *reversed complements*, in that, regarding them, nucleotide for nucleotide, in opposite directions, i.e., *5'-3'* vs *3'-5'*, they consist of complementary nucleotides, i.e., an A on one strand corresponds to a T on the other and vice versa and similarly for C and G. The two strands are joined together like this – in opposite directions – via base-pairing between individual nucleotides from each strand, as shown in figure 2.2 d).

2.4 Transcription

DNA-transcription involves an *enzyme*, a kind of protein, called *RNA polymerase*. The RNA polymerase may bind to a pair of *promoter sequences* on a strand of DNA. The promoters are short segments of nucleotides that complements two reactive sites in the RNA polymerase. When the RNA polymerase binds to promoters on one of the DNA-strands it begins transcribing that strand in the $3' - 5'$ direction. Along the way, the polymerase *denatures* the DNA, i.e., it splits the base-pairing that holds the two strands together, just enough to allow interaction with the individual bases on the strand that it transcribes. During transcription, free nucleotides available in the cell are paired to bases that have been laid bare on the DNA and linked together to form a growing poly-nucleotide called *RNA* (ribonucleic acids). Transcription stops once another pattern, called a *termination signal*, is transcribed.

The resulting RNA is the reversed complement of the DNA segment, that was transcribed, except that in stead of transcribing A to its DNA complement, T, a slightly different version of that nucleotide, *uracil* – conventionally represented as U, is used.

The strand of the segment of DNA that was transcribed is called the *template*-strand to distinguish it from the *coding* strand of the DNA, where

the segment is identical to the newly produced RNA (Us in the RNA for Ts in the DNA). The template strand is also sometimes called the *reverse* or the *anti-sense* strand and the coding strand is sometimes also called the *direct* or the *sense* strand. Depending on promoter location, either strand may be undergo transcription.

RNA

A *RNA*-sequence is a single-stranded transcription of a gene (see 2.1. It is identical to the direct strand of the *DNA* of that gene with Us instead of Ts. For the present discussion it is useful to distinguish between two main types of RNA:

- *Messenger-RNA* or just *mRNA*, transcribed from protein-coding genes that are later *translated* to proteins (see below).
- Different kinds of *functional RNA* that serve various crucial functions in the cell directly as is and thus never translated.

Functional RNA, structure is function

Very little is known about the functions served by a vast host of functional RNA. It is however generally believed that functional RNA relies heavily on its *structure* for serving its proper functioning in the cell. This is because molecular function relies on interaction between reactive regions of individual molecules and the molecular structure must allow easy contact between such regions for interaction to occur.
Due to base-pairing and the exertion of other biochemical forces, a RNA strand inadvertently fold up on itself, forming particular structures, depending in part, on its nucleic acid sequence. *RNA-structures* can be described in three different levels of abstraction conventionally referred to as *primary*, *secondary* and *tertiary RNA-structure*, respectively:

- the *primary RNA-structure* is the linear sequence of nucleotides making up the RNA.
- the *secondary RNA-structure* refers to a 2-dimensional structure resulting from base-pairing of complementary regions of the strand while leaving other regions *un-paired*. Secondary RNA structures are classified in several different general groups (see figure 2.3 *Right* on page 19 for some of them). One of the most important of these general structures is the hairpin loop (see below).

- the *tertiary RNA-structure:* The secondary RNA-structures is "flat", but RNA-molecules are not. Individual nucleotides in a sequence "fit" into its neighbours in such a way that each orient itself slightly differently than the one before it, causing the entire strand to twist and curl. There are also other biochemical forces of attraction and repulsion that cause a 3-dimensional structure that is particular to each type of RNA and referred to as its *tertiary* structure ([77]).

Arguably, knowing the tertiary-structure of a piece of functional RNA, enables narrowing in on its possible functions. Tertiary-structure is constrained by its secondary structure, that is in turn constrained by its primary structure, therefore a great deal of recent and current research is concerned with predicting secondary and tertiary structures from the primary structure of a RNA sequence, which can again be assessed via its DNA.

"Hairpins" and "Loops"

Hairpin loops, or just *hairpins*, refers to a generic type of secondary RNA-structure that consists of two base-pairing regions forming a single *stem* supporting a small *loop* of un-paired nucleotides, figure 2.3 a). Hairpins are in a sense the basic secondary structure, because other structures can be seen as special cases or combinations of hairpins.

2.5 Translation

Translation refers to a process that constructs a protein according to the information in a specific mRNA. Translation occurs when the *ribosome*, a complex molecule consisting of both nucleotides and amino-acids ([3]), binds to a pattern of nucleotides in the mRNA called a *ribosomal bindings-site* or just *RBS*.

The ribosome translates nucleotides in non-intersecting sets of three, so-called *codons*, starting a few nucleotides from the RBS, with a so-called *start-codon*. Each codon, including the start-codon, corresponds to one of 20 amino-acids, which are produced in interaction between the ribosome and so-called *transfer RNA* molecules or *tRNA*, see right part of figure 2.3. Each type of tRNA has one of the 20 amino acids attached to its *3'*-end. In the opposite portion of the tRNA molecule, there is a reactive site of three specific nucleotides complement to one of the codons in the RNA, the

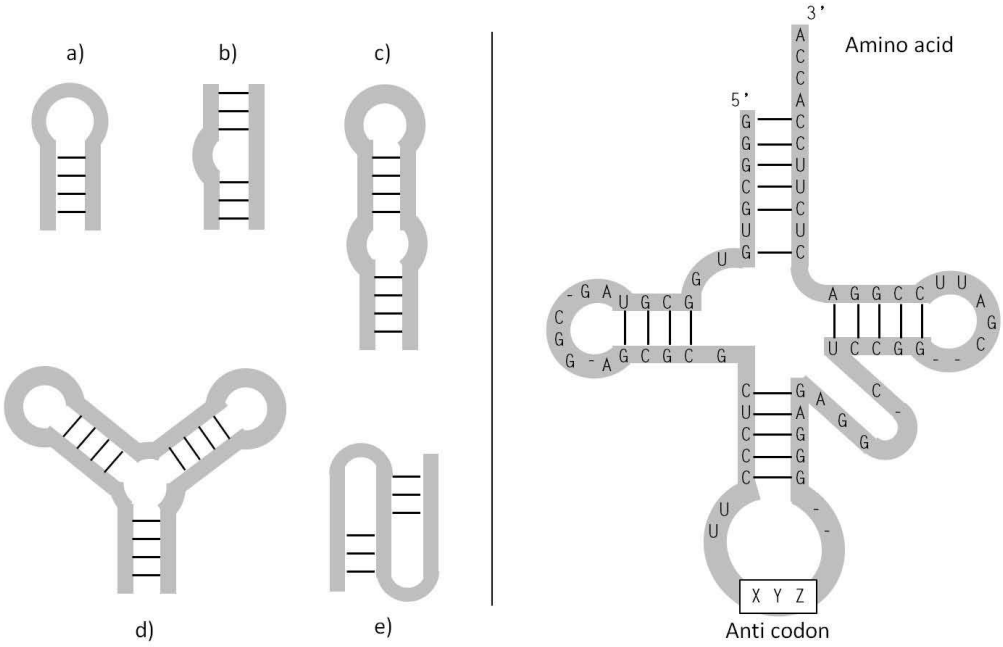


Figure 2.3: *(Left)* Secondary structure of DNA and RNA. The grey line represents the sequence of nucleotides of the tRNA molecule. The black lines indicate base-pairing that forms the secondary structure from the primary. *Hairpin-loops* consist of a stem of base-paired nucleotides supporting a loop **a)** of unpaired nucleotides. Other generic structures include *bulge loop* in **b)**, *inner loop* in **c)**, *multistem loop* in **d)** and *pseudo-knot* in **e)**, that involve crossing base-pair relations. *(Right)* The particular secondary structure of tRNA, after [71]. Also indicated in the figure are the reactive sites of the tRNA described in section 2.5.

so-called *anti-codon*. When the anti-codon binds – via base-pairing – to the codon, that is being translated, it causes the attached amino acid to be transferred from the tRNA to the growing polypeptide under construction by the ribosome.

Translation stops when one of three specific stop-codons are encountered, causing the polypeptide, the now finished protein, to be detached from the ribosome.

Reading frames

There are three different offsets at which the sequence of codons may begin in the DNA (0, 1 or 2 bases) and since DNA can be transcribed in both

directions, a translated region is said to be in one of six *reading frames* of the DNA.

Open reading frames (*ORF*s)

A start-codon followed by a sequence of codons and terminated by a stop-codon is referred to as an *open reading frame* or *ORF*, regardless of whether it is actually translated or not.

Genetic code

Since there are 64 different codons and only 20 amino-acids, most amino-acids are encoded by several codons. The *genetic code*, i.e., which codons are considered start-codons by the ribosome and what amino-acids each codon corresponds to, differs slightly from organism to organism. Escherichia Coli, the bacterial organism that is used for the experiments in this work, follows a genetic code table conventionally numbered *table 11*, and showed in figure 2.4(p. 22). Common in all tables is that stop-codons terminate translation and are not translated themselves.[2] Start codons may occur any number of times during translation and, apart from initiating translation, they are also translated to specific amino acids themselves.

Amino-acids

As mentioned in section 2.1 above, amino-acids are the basic building blocks of proteins, similar to the way in which nucleic acids are the basic building blocks of DNA and RNA. Like nucleotides can be linked to each other to form poly-nucleotides – of which DNA and RNA are prominent examples, amino-acids can also be linked to each other in what is called polypeptides – of which proteins are examples. There are in total 22 different *standard amino-acids* of which 2 are usually left out of the general discussion.[3] The remaining 20 are listed in the bottom half of figure 2.4.

[2] There are rare exceptions where stop-codons occur internally in a transcribed gene. These are considered so rare, that they do not affect the general trend, but are just extremely rare exceptions to the rule.

[3] When stop-codons are overridden during translation, and thus occur internally in a gene, two additional amino acids result, namely *Selenocysteine* and *Pyrrolysine*. As already mentioned, these circumstances are extremely rare and not important for this introduction. Apart from the standard amino acids there exist a great number of non-standard amino-acids, that are not relevant to our purposes and therefore not described further.

Each individual amino acid has particular biochemical properties and characteristics that dictates its behaviour under different circumstances. *Hydrophobic* amino acids, for example, repels water, as opposed to *hydrophilic* amino-acids, that are attracted to water. Other groupings of amino acids can be distinguished according to their electrical charge and various other factors. Each protein is defined by the corresponding sequence of amino-acids, also here referred to as the primary structure of the protein and the properties of individual amino acids in a protein dictates, how proteins, similarly to functional RNA, fold up on themselves to form complex secondary and tertiary protein structures, and how they react and interact with their surroundings.

2.6 Genome annotation and Gene-finding

In this work, *genome-* and *DNA-annotation* refers to the identification and marking of aspects, features and properties in a sequence of DNA that could be relevant for identifying genes in that sequence. Thus, the ultimate goal of *DNA-annotation* w.r.t. some DNA, would be the annotation of all genes and other functional features in it. Interesting features in this regard informally include the DNA-sequence itself, the genes in it and 'everything in between'. We might express gene-finding as a binary classification problem, that seeks to classify each open reading frame as either belonging to the *positive* class, i.e, for actual genes or the *negative* class, i.e., non-genes.

Traditional measures for accuracy are defined in terms of *true positives*, *false positives*, *true negatives* and *false negatives*, in this case:

TP: number of actual genes correctly annotated.
FP: number of non-gene regions wrongly annotated as genes.
TN: number of correctly annotated non-gene regions.
FN: number of actual genes wrongly annotated as non-gene regions.

In these terms, *Sensitivity*, *Specificity* and *Accuracy* are defined as follows:

$$Sensitivity(SN) = \frac{TP}{TP + FN} \tag{2.1}$$

$$Specificity(SP) = \frac{TP}{TP + FP} \tag{2.2}$$

$$Accuracy(AC) = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

Informally, *Sensitivity* concerns the proportion of actual genes correctly annotated as such, *Specificity* concerns the proportion of correctly annotated

The Genetic Code:

TTT	F	Phe	TCT	S	Ser	TAT	Y	Tyr	TGT	C	Cys
TTC	F	Phe	TCC	S	Ser	TAC	Y	Tyr	TGC	C	Cys
TTA	L	Leu	TCA	S	Ser	TAA	*	Ter	TGA	*	Ter
TTG	L	Leu i	TCG	S	Ser	TAG	*	Ter	TGG	W	Trp
CTT	L	Leu	CCT	P	Pro	CAT	H	His	CGT	R	Arg
CTC	L	Leu	CCC	P	Pro	CAC	H	His	CGC	R	Arg
CTA	L	Leu	CCA	P	Pro	CAA	Q	Gln	CGA	R	Arg
CTG	L	Leu i	CCG	P	Pro	CAG	Q	Gln	CGG	R	Arg
ATT	I	Ile i	ACT	T	Thr	AAT	N	Asn	AGT	S	Ser
ATC	I	Ile i	ACC	T	Thr	AAC	N	Asn	AGC	S	Ser
ATA	I	Ile i	ACA	T	Thr	AAA	K	Lys	AGA	R	Arg
ATG	M	Met i	ACG	T	Thr	AAG	K	Lys	AGG	R	Arg
GTT	V	Val	GCT	A	Ala	GAT	D	Asp	GGT	G	Gly
GTC	V	Val	GCC	A	Ala	GAC	D	Asp	GGC	G	Gly
GTA	V	Val	GCA	A	Ala	GAA	E	Glu	GGA	G	Gly
GTG	V	Val i	GCG	A	Ala	GAG	E	Glu	GGG	G	Gly

Aminoacid single- and triple-letter symbols:

Arginine	R	Arg	Glutamine	Q	Gln	Lysine	K	Lys
Threonine	T	Thr	Asparagine	N	Asn	Glycine	G	Gly
Methionine	M	Met	Tryptophan	W	Trp	Asparticacid	D	Asp
Histidine	H	His	Phenylalanine	F	Phe	Tyrosine	Y	Tyr
Cysteine	C	Cys	Isoleucine	I	Ile	Valine	V	Val
Proline	P	Pro	Leucine	L	Leu	*Stop codon*	*	Ter

Figure 2.4: The genetic code shows how individual codons are translated by the ribosome in a large group of organisms including all bacteriae. The table is organised according to the three codon positions and each codon is followed by the amino-acid that it translates to, represented by both a single- and a triple-letter symbol. Possible start-codons in this table are marked with the lower case letter *i*. It can be seen that all start-codons translate into amino-acids in addition to serving as translation initiation signals. Contrarily, stop codons, with single and triple letter symbols "*" and "Ter", do not translate into amino-acids but only signal translation termination. Several codons, varying primarily in the third codon position, i.e., within an inner "box" of the table, translate to the same amino-acid.

genes to nongenes annotated as genes and *Accuracy* concerns the proportion of correct annotations (genes or non-genes) to wrong ones.

An important problem with using these measures for gene-finding is that, in general, we do not know the *correct* answers. For lack of better alternatives, the measures are still used, but it is important to keep in mind that they refer to the extent of current domain knowledge. So an open reading frame that is known to represent a gene will be counted as belonging to the positive class. Only the annotation that annotates the ORF as a gene from the correct start to the correct end, according to this knowledge, will be counted as a *true positive*. Consequently, an annotation that gets either end wrong is therefore counted as a *false positive*, even though some (or all) of the ORF may actually be *part* of a gene. Similarly, only an ORF annotated as a non-gene, that is in fact known to represent a gene, represents a *false negative*, and a *true negative* otherwise. Given that our knowledge about genes is not complete, undiscovered genes, i.e., genes that we do not know of, will be among be among the false positives, if they are detected by the gene-finder.

Intrinsic versus Extrinsic methods for gene-finding

Genes can be identified and verified by experiments in biochemical laboratories. Nowadays, however, *gene-finding* typically refers to the application of computational methods for identifying, or more precisely, *predicting* possible genes in DNA. A distinction is usually made between *intrinsic* and *extrinsic* gene-finders, depending on what information is considered for gene-prediction. I will return to these matters in more detail later, and just present the distinction informally here.

Intrinsic methods consider only the DNA that is being analysed, i.e., the distribution of features like promoters, start- and stop-codons, codon sequence, and termination sites, see figure 2.5 (p. 25). They do, of course, require knowledge about the *likely* distribution of those features in genes. Intrinsic methods are quite efficient in circumstances where little or no knowledge is available about the genome under investigation. The primary weakness of intrinsic methods is that these signals are all quite weak. Finding, especially short, undetected genes in well-studied genomes is extremely hard and may require increased intrinsic knowledge about genes, beyond what has already been mentioned.

Extrinsic methods relies primarily on similarity to verified genes of closely related organisms, *homologs*, for gene-prediction. The primary weakness of extrinsic methods is that they require extensive annotation of other genomes, and they are limited to finding genes that are similar to known

genes in homologous genomes. While gene-finders that incorporate all available information are seeing recent improvement (see [8] for a review of eukaryotic genefinders), there is a danger of consistently missing entire groups of genes that differ from the already discovered genes in one way or another.

In this work, a framework that supports both approaches is developed.

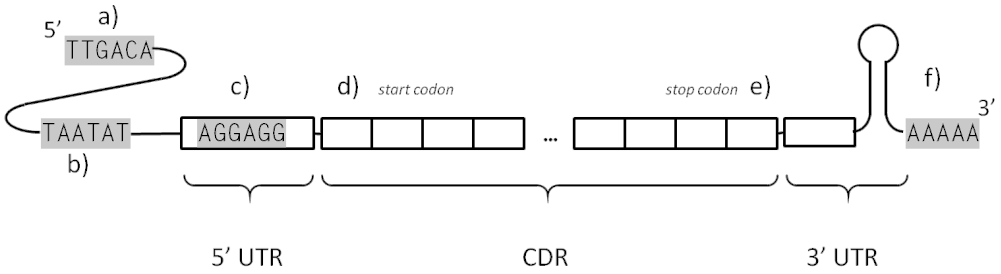


Figure 2.5: Some intrinsic signals for transcription and translation in prokaryotic DNA. The diagram shows the coding strand of a gene and its immediate surroundings in DNA . Shown in frames are the transcribed parts, i.e., **c)**, **d)** and **e)**, whereas the remaining parts, **a)**, **b)** and **f)**, interact with reactive sites in the polymerase for guiding the transcription process. The sequences, $TTGACA$ and $TAATAT$, at **a)** and **b)**, represent the two promoters, referred to as the *-35 box* and *-10 box* (or *TATA-box*), that occur approximately 35 nucleotides and 10 nucleotides before transcription start. Both can be present, but the *-35 box* is often missing and the *-10 box* may also be absent. Even if present, the sequences shown here are only so-called *consensus motifs*, i.e., sequences of most frequent nucleotides for each position. Commonly, actual promoters only have four or five of the six symbols matching the consensus. In c), is the 5' untranslated region, $5'UTR$, so-called because it is not translated by the ribosome. It includes a reactive site for the ribosome to bind to, a *ribosomal bindings site*. The ribosomal binding site shown here is the so-called *Shine-Dalgarno-sequence* (after John Shine and Lynn Dalgarno who proposed it) that most often occur in prokaryotes (but never in eukaryotes). Like for the promoters, the sequence $AGGAGG$ is also only a consensus motif and may vary significantly from gene to gene. From **d)** to **e)** is the *coding region*, CDR that determines the gene product. It starts with one of several possible start-codons and proceeds with a varying number of codons (from around twenty and up to several thousands) and terminated by one of a few stop codons. After the CDR follows the *3' untranslated region*, $3'UTR$, ending with the transcription termination signal at **f)**. The termination signal shown here consist of a strong hairpin followed by a sequence of A''s that is transcribed to a sequence of relatively weakly bound U's in the RNA. The energy by which the stem forms is thought to literally tear the growing poly-nucleotide from the U's, releasing the finished RNA.

Chapter 3

Formal Language Theory

Biological sequence data as defined in section 2 consists of sequences of symbols. Regarded as such, DNA and RNA are not much different from natural language or programming languages and many tools and methods developed for these domains can be applied with little or no adaptation to the biological domain. In this chapter, I include a selection of general definitions of formal languages, grammars and automata that have been applied both in compiler theory and natural language modelling (see for instance [1, 2, 68, 36]).

3.1 String languages

I will use the terms *sequence* or *symbol sequence* interchangeably to refer to a finite string over some set of symbols called an *alphabet* denoted Σ.

Definition 1 (Alphabet) *An* alphabet*, denoted Σ, is any set of symbols.*

Definition 2 (String) *A* string *over an alphabet Σ is defined recursively as follows:*

- *the empty string, denoted ϵ, is a string over any alphabet*
- *if ω is a string over Σ and $\alpha \in \Sigma$, then $\omega\alpha$ is a string over Σ*
- *nothing else is a string over Σ.*

We adopt the following short-hands:

- *Σ^+ denotes the set of all non-empty strings over Σ, and*
- *Σ^* denotes the set of all strings over Σ, i.e., $\Sigma^+ \cup \{\epsilon\}$*
- *For a string ω, $|\omega|$ denotes the number of symbols in that string.*

Definition 3 (Concatenation) *The* concatenation *of x and y, where x and y are strings over Σ, denotes the string consisting of the symbols of x followed by the symbols of y and can be written both as xy and $x \bullet y$.*

Definition 4 (Language over Σ) *A* language *L over Σ is a possibly infinite set $L \subseteq \Sigma^*$ of strings over Σ.*

3.2 Languages and Grammars

Specific languages can be denoted by *grammars* as defined as follows.

Definition 5 (Grammar) *A* rewriting grammar *is a quadruple $G = \langle N, \Sigma, R, S\rangle$ where*

- *N is a finite set of nonterminal symbols,*
- *Σ is a finite set of terminal symbols, disjoint from N,*
- *R is a finite set of rewriting rules (α, β), where*
  - *$\alpha \in (N \cup \Sigma)^+$ and $\beta \in (N \cup \Sigma)^*$*
  - *$S \in N$ is a distinguished start-symbol.*

We adopt the conventions that

- *rules (α, β) may be written $\alpha \to \beta$ and*
- *a set of rules $\alpha \to \beta_1, \alpha \to \beta_2 \ldots \alpha \to \beta_n$ may be written $\alpha \to \beta_1|\beta_2|...|\beta_n$.*

Definition 6 (Rewriting step) *For a grammar, $G = \langle N, \Sigma, R, S\rangle$ and two symbol sequences, $\alpha\beta\gamma$, $\alpha\delta\gamma$ where $\alpha \in (N \cup \Sigma)^+$ and $\beta, \gamma, \delta \in (N \cup \Sigma)^*$, $\alpha\beta\gamma$ can be* rewritten in one step *as $\alpha\delta\gamma$ iff there is a rule $\beta \to \delta \in R$. We write:*

$$\alpha\beta\gamma \Rightarrow \alpha\delta\gamma$$

Definition 7 (Derivation) *For a grammar, $G = \langle N, \Sigma, R, S\rangle$ and symbol sequences, $\alpha, \beta_i, \gamma \in (N \cup \Sigma)^*$, a* derivation *from α to γ is a sequence of rewriting steps,*

$$\alpha \Rightarrow \beta_1, \beta_1 \Rightarrow \ \ldots \ \Rightarrow \gamma$$

We adopt the convention that

- *a derivation of n rewriting steps may be written $\alpha \Rightarrow^n \gamma$ and*
- *a derivation of any number of rewriting steps may be written $\alpha \Rightarrow^* \gamma$.*

In unrestricted grammars there are many ways of deriving a given sequence that are essentially the same, differing only in the order of individual rewriting steps. Different rules of preference exist for different classes of grammars, and the issue will be dealt with as appropriately below in section 3.3 and 3.4, when describing the regular and context-free classes of languages.

Definition 8 (Parse) *The language $L(G)$ denoted by a grammar $G = \langle N, \Sigma, R, S\rangle$ is the set of terminal symbol sequences, ω, that can be derived from the start symbol S, i.e., $\{\omega \in \Sigma^* | S \Rightarrow^* \omega\}$*

– $\xi(\omega)$ *denotes a derivation* $S \Rightarrow^* \omega$*, also called a* parse *of* ω *in* G

Definition 9 (Ambiguity) *For a grammar* $G = \langle N, \Sigma, R, S \rangle$*, a sequence* $\omega \in L(G)$ *is* ambiguous in G *if there exists different parses* $\xi(\omega)$ *and* $\xi'(\omega)$ *for* ω *in* G*.*

A classic classification of languages is the *Chomsky hierarchy of languages* [12], that distinguishes between five classes:

$$\textit{finite} \subset \textit{regular} \subset \textit{context-free} \subset \textit{context-sensitive} \subset \textit{unrestricted}$$

The unrestricted class contains all languages. The remaining classes each pose gradually more restrictive constraints on the rules in their respective grammars and denote subset languages of decreasing expressive power. However, while finite languages are restricted to contain a finite set of strings, the other classes all denote languages of possibly infinitely many strings. In general, the more restricted languages are easier to analyse but also less expressive. For a string ω and a regular language L, it can be established in linear time whether $\omega \in L$ in terms of $|\omega|$. For context-free languages, the task requires almost cubic time in the worst case, [73], whereas context-sensitive languages require exponential time, and it is generally un-decidable whether a sequence belongs to an unrestricted language or not. Grammar-formalisms that are more expressive than context-free grammars but still recognisable in polynomial time, are of great interest, not only to computational linguists but also in general (see [36] for a thorough presentation of current formalisms). Computational complexity of algorithms are traditionally distinguished as either polynomial or exponential in terms of input data size, where polynomial is considered acceptable. For large data sizes this distinction is not sufficient since some polynomial algorithms are too complex to be practical. The present work is restricted to describe in detail only regular and context-free grammars, i.e., up to cubic complexity in terms of input size.

3.3 Regular languages

Definition 10 (Regular grammar) *A* regular language L *is described by a regular rewriting grammar* $G = \langle N, \Sigma, R, S \rangle$ *with rules in* R *of the form* $A \to a$ *or* $A \to aB$ *where:*

- $A, B \in N$ *and*
- $a \in \Sigma^*$

Since only nonterminals can occur in the head of rules and each rule may contain at most one non-terminal, there is only one order in which rules can be rewritten, and the possible ambiguity arising form different orders of rewriting steps does not apply for regular grammars.
Regular languages can also be specified by *regular expressions* defined as follows.

Definition 11 (Regular expression) *A* regular expression ρ *over an alphabet* Σ *and the language* L_ρ denoted *by it are defined recursively as follows:*

- ***the empty string****:* ϵ *is a regular expression denoting the language* $L_\epsilon = \{\epsilon\}$.
- ***unit****:* a*, where* $a \in \Sigma$*, is a regular expression denoting the language* $L_a = \{a\}$.
- ***concatenation****: if* α *and* β *are regular expressions, then so is* $\alpha\beta$*, denoting the language* $L_{\alpha\beta} = \{x \bullet y | x \in L_\alpha \bigwedge y \in L_\beta\}$.
- ***disjunction****: if* α *and* β *are regular expressions, then so is* $\alpha + \beta$*, denoting the language* $L_{\alpha+\beta} = L_\alpha \cup L_\beta$.
- ***Kleene star****: if* α *is a regular expression, then so is* α^**, denoting zero or more concatenations of* α*, i.e., closure over concatenation.*
- ***parentheses*** *may be used for disambiguation, i.e., if* α *is a regular expression, then so is* (α)*, denoting* L_α.
- ***nothing else*** *is a regular expression.*

Any language that can be specified as a regular expression is also a regular language.

Example 1 *Consider the language* L_0 *over the alphabet* $\Sigma = \{0, 1\}$ *of sequences beginning with a* 0 *and containing at least one* 1*. We can specify* L_0 *as the language denoted by the regular expression :*

$$\rho_0 = 00^*1(0|1)^*$$

Example 2 *The language* L_0 *specified by the regular expression* R_0 *from example 1 can be specified also as the grammar* $G_0 = \langle\{S, A\}, \{0, 1\}, R, S)\rangle$*, where* R *consists of the rules:*

$$\begin{aligned} S &\to \quad 0S \\ S &\to \quad 1A \\ A &\to \quad 0A \\ A &\to \quad 1A \\ A &\to \quad \epsilon \end{aligned}$$

Note, that this is an unambiguous grammar, in that for any $\omega \in L(G_0)$ *there exists exactly one parse* $\xi(\omega)$.

For any regular expression there is at least one regular grammar specifying the same language (proved elsewhere; see for example [1]) and vice-versa. Algorithms that transform between regular expressions and equivalent regular grammars are well-known in the field.

Finite State Automata

Yet another way of specifying languages is by way of *automata*. An automaton for a language L is an abstract machine that accepts a sequence ω as input and reports as output, whether $\omega \in L$ and it is said to *recognise* L. For any grammar there is a corresponding automaton that specifies the same language. For regular grammars, the type of automata is called Finite State Automata (FSA). There exist algorithms for transforming regular expressions into equivalent FSA's and vice versa [2] but that is outside the scope of the present work. FSA's are central to the description of their probabilistic extensions *Hidden Markov Models* (chapter 4). There are two equivalent versions of FSA's, referred to as Moore- and Mealy-machines respectively. In both traditions, FSA's are essentially directed graphs where the vertices represent states and the edges represent transitions between states. The difference between the two traditions lies in how they consider emissions:

- in Moore-machines, a set of possible state-emissions is associated with each state,
- in Mealy-machines individual emissions are associated with individual transitions.

Here, I define the Moore-version since that is the preferred tradition in the bioinformatics domain.

Definition 12 (FSA) *A* finite state automaton *over some alphabet* Σ *is a sextuple* $FSA = \langle S, Begin, End, T, \Sigma, E \rangle$*, where:*

- S *is the set of states,*
- $Begin \in S$ *is a distinguished* begin state*,*
- $End \in S$ *is a distinguished* end state*,*
- $T \subseteq S \times S$ *is a set of directed* state transitions*.*
- Σ *is the alphabet of* emittable symbols*,*
- $E \subseteq S \times \Sigma$ *is a set of possible* state emissions*.*

The following conventions are adopted:

- $T(s) = \{t|(s,t) \in T\}$ *denotes the set of states reachable from* $s \in S$ *in one transition.*
- $E(s) = \{\beta|(s,\beta) \in E\}$ *denotes the set of emittable symbols from* $s \in S$*.*
- *Any state,* s*, that have no emittable symbols, i.e.,* $E(s) = \emptyset$*, is called* silent *states.*

Note: Traditionally, the specification of FSA's involve a non-empty set of possible end-states with emissions. I prefer a version with a single *End*-state with no emissions. This version can be formed from the traditional definition by including the new *End*-state in S and adding to T a transition from each of the original *End*-states to this new one.

Definition 13 (Path) *A* path $\mathcal{P}$ *in an* $FSA = \langle S, Begin, End, T, \Sigma, E\rangle$ *is defined as a finite sequence of* n *states* $\pi_0, \ldots, \pi_{n-1} \in S$ *such that* $\pi_0 = Begin, \pi_{n-1} = End$ *and* $(\pi_i, \pi_{i+1}) \in T$*, where* $0 \leq i < n-1$*,*

A path, $\mathcal{P}(\omega)$, deriving ω in a FSA corresponds to a parse, $\xi(\omega)$, in an equivalent grammar.

Two different paths $\mathcal{P}(\omega)$ and $\mathcal{P}'(\omega)$ deriving the same sequence, ω, in an FSA is proof of the ambiguity of ω – just as two different parses $\xi(\omega)$ and $\xi'(\omega)$.

Example 3 *The regular expression* ρ_0 *from example 1 (p. 30) and the equivalent grammar* G_0 *(example 2) can be specified by the unambiguous FSA,* $FSA_0 = \langle S, Begin, End, T, \Sigma, E\rangle$*, where*

- $S = \{s_0, s_1, s_2, s_3, s_4\}$
- $Begin = s_0$
- $End = s_4$
- $T = \{(s_0, s_1), (s_0, s_2), (s_1, s_1), (s_1, s_2), (s_2, s_3), (s_2, s_4), (s_3, s_3), (s_3, s_4)\}$
- $\Sigma = \{0, 1\}$
- $E = \{(s_1, 0), (s_2, 1), (s_3, 0), (s_3, 1))\}$

Any FSA can be represented graphically as shown for FSA_0 in figure 3.1. When representing FSA's graphically, each state $s_i \in S$ is drawn as a uniquely labelled vertex. Here vertices are circles and labels are printed inside the circles. Any accepting state $e \in End$, is traditionally distinguished by a double circle (however when the meaning is clear from the context the double circle can be omitted). Any state s_i in S is also labelled with the set of possible emissions $\{(s_i, \beta) \in E\}$, for the corresponding state in S. For each transition $(s_i, s_j) \in T$, a directed edge is drawn from the vertex corresponding to s_i to the vertex corresponding to s_j. A sequence of symbols

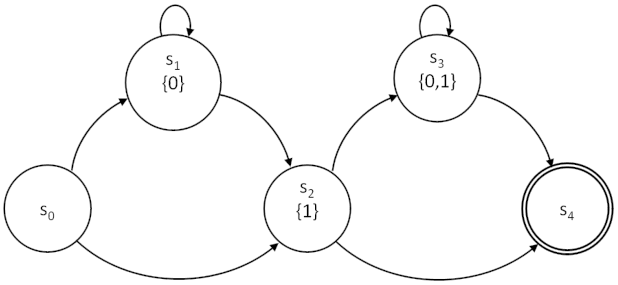


Figure 3.1: Graphical representation of FSA_0

$[\beta_1 \dots \beta_n]$ is accepted as a sentence in the language recognised by the FSA, if a there exists a path $s_0, \pi_1, \ \dots \ , \pi_n, s_i$, such that s_i is a state in *End* and in each nonsilent state π_i along the path, the symbol β_i is among the possible emissions. If no such path is possible, the sequence is rejected.

Example 4 *A grammar, that distinguishes ORFs as defined in section 2.5 in the first direct reading frame of DNA, can be represented as the FSA illustrated in Figure 3.2 below. The model ensures that genes start with a start codon in state sequence $s_1 \rightarrow s_2 \rightarrow s_3$, and that when a stop codon is encountered, in state sequences $s_5 \rightarrow s_8 \rightarrow s_{13}$ or $s_5 \rightarrow s_9 \rightarrow s_{14}$, the ORF portion of the model is terminated and a new recursion must follow or there are no more symbols in the sequence. It also models codon sequences as sequences of nucleotide triplets. Finally it models non-ORF regions simply as a sequence of arbitrary nucleotide-triplets, ensuring that we stay in the same reading frame.*

It is easy to see that this FSA is ambiguous. For an arbitrary ORF-sequence, there exist a path through the FSA starting in s_1, with the start codon and ending with the stop-codon in s_{16} or s_{17}. However, there exists also at least one alternative path, namely the one that starts in the non-ORF state, s_{15}, and stays there until the end of the sequence. Each additional possible start-codon inside the ORF, will give rise to one more path through the FSA, starting with a series of non-ORF triplets before beginning a series of ORF-triplets from that particular start-codon and out.

3.4 Context-free languages

Definition 14 (CFG) *A* Context-free grammar *(CFG) is a grammar $G = \langle N, \Sigma, R, S \rangle$ where the rules in R are of the form $A \rightarrow \beta$ and:*

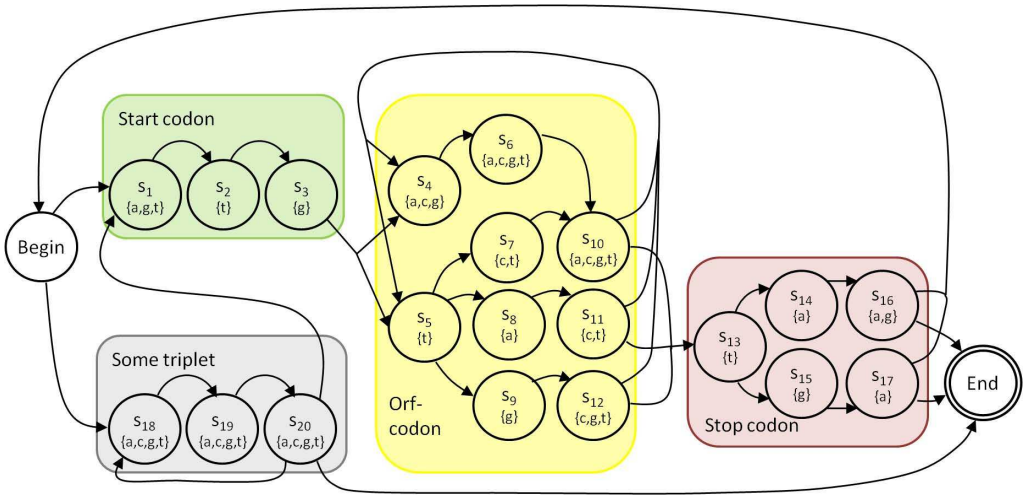


Figure 3.2: An FSA representation for a DNA reading frame with the capability of distinguishing substrings as open reading frames (ORF's). In this specification, an ORF starts with a start-codon in the states in the green portion of the figure. Then follows a sequence of coding codons, in the yellow portion before a stop-codon ends the ORF in the red portion of the figure. Only paths that include states s_{16} or s_{17} specify a stop codon. A non-ORF is simply a series of triplets of nucleotides, as specified in the grey portion of the figure. This FSA accepts any series of ORF's and non-ORFs that can be expressed as a series of nucleotide triplets.

- $A \in N$ *and*
- $\beta \in (N \cup \Sigma)^*$.

In other words, context-free grammars are grammars where the left-hand-sides of rules in R consist of exactly one nonterminal, while the right-hand-sides are unrestricted. Because each right-hand-side may contain more than one non-terminal, a potential ambiguity arises from rewriting them in different orders. Therefore, by convention, in each rewriting-step the left-most nonterminal is rewritten.

For CFGs derivations $\xi(\omega)$ are traditionally represented as equivalent *parse trees* $Tree(\omega)$:

Definition 15 (Parse Tree) *For a CFG,* $G = \langle N, \Sigma, R, S \rangle$ *and a sequence* $\omega = \sigma_1 \sigma_2 \ldots \sigma_n$, $\omega \in L(G)$, *a* parse tree, $Tree_\omega$, *for* ω *is an ordered labelled tree with the following properties:*

- *the root node of* $Tree_\omega$ *is labelled by the start symbol* S
- *each node in* $Tree_\omega$ *is either*

- *the root node of a sub-tree labelled by a nonterminal symbol* $\alpha \in N$ *such that, if* $\beta_1, \beta_2 \ldots \beta_m$ *are the labels of the roots of the immediate descendants of* α *from left to right, then* $\alpha \rightarrow \beta_1\beta_2 \ldots \beta_m$ *is a rule in* R.
- *or a leaf node labelled by a terminal symbol* $\sigma \in \Sigma$ *or* ϵ *such that, when read from left to right, the leaves of* $Tree_\omega$ *form the string* $\omega = \sigma_1\sigma_2 \ldots \sigma_n$.

We write $Tree(\omega)$ *and* $Tree'(\omega)$ *or* $Tree^i(\omega)$ *to distinguish between different parse trees for the same sequence,* ω.

The existence of different parse trees $Tree(\omega)$ and $Tree'(\omega)$ for a sequence ω is proof of the ambiguity of ω, just as different parses $\xi(\omega)$ and $\xi'(\omega)$.

CFGs denote context-free languages (including regular languages). Specifically, CFGs can describe nested constructs like palindromes, which is impossible with a regular grammar. FSA's are not sufficient as recognising automata for context-free grammars, instead the general type of recognising automaton for this class of languages is *pushdown automaton* (defined elsewhere, for example [1, 36]). Consequently, parsing context-free languages is significantly more complex, in the worst case cubic in the length of the string to be analysed (in contrast to linear), [73].

Example 5 *Examples of a context-free language, for which there exists no regular grammar includes the language consisting of sequences of 0's followed by an equal number of 1's. An unambiguous context-free grammar for that language can be expressed as* $G_2 = \langle\{S, A\}, \{0, 1\}, R, S)\rangle$ *with the following rules in* R:

$$\begin{aligned} S &\rightarrow \quad 0A \\ A &\rightarrow \quad 0A1 \\ A &\rightarrow \quad 1 \end{aligned}$$

Example 6 *The secondary RNA structure called* hairpin loop *is another important example of a non-regular context-free structure (see section 2.4). A grammar for hairpin loops can be defined as*

$$G_3 = \langle\{HairpinLoop, Stem, Loop\}, \{a, t, c, g\}, R, S)\rangle$$

with the following rules in R:

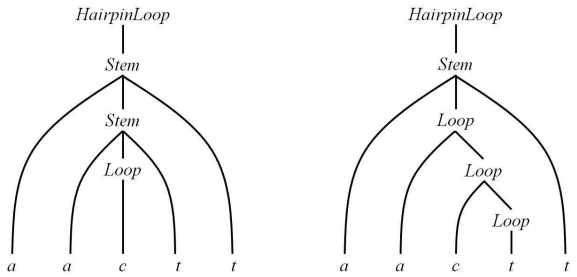


Figure 3.3: Two alternative parsetrees for the sequence *aactt*, given the ambiguous grammar for hairpin loops G_3 from example 6.

$$
\begin{aligned}
\mathit{HairpinLoop} \rightarrow &\quad a\ \mathit{Stem}\ t \mid t\ \mathit{Stem}\ a \mid c\ \mathit{Stem}\ g \mid g\ \mathit{Stem}\ c \\
\mathit{Stem} \rightarrow &\quad a\ \mathit{Stem}\ t \mid t\ \mathit{Stem}\ a \mid c\ \mathit{Stem}\ g \mid g\ \mathit{Stem}\ c \mid \mathit{Loop} \\
\mathit{Loop} \rightarrow &\quad a\ \mathit{Loop} \mid t\ \mathit{Loop} \mid c\ \mathit{Loop} \mid g\ \mathit{Loop} \mid \\
&\quad a \mid t \mid c \mid g
\end{aligned}
$$

This grammar is clearly ambiguous. For some sequence, for example aactt, a stem of at least length one is required as specified in the HairpinLoop-rule. Here, the initial a and final t suffice. But the sub-sequence forming the Stem-part, here act may be interpreted both as a continuation of the stem by one of the nested α Stem β alternatives, figure 3.3 (left), or as one big loop by immediately entering the Loop alternative, figure 3.3 (right).

Chapter 4

Probabilistic Sequence Analysis

As illustrated by the examples, many grammars and their equivalent automata are ambiguous. In compiler theory, that concerns the design and implementation of programming languages, ambiguity is undesirable and avoided altogether. Natural language, however, is inherently highly ambiguous and that is also the case for many grammars specifying biological sequence data. Ambiguity of a sequence in a language means that there exists different derivations for the sequence in the grammar of the language. Probability theory or probability calculus provides a systematic way of making informed choices among sets of possible solutions.
In this chapter I include basic definitions from probability theory and go on to describe probabilistic extensions FSA's and CFG's (HMM's and PCFG's respectively). The remainder of the chapter contains a description of relevant inference tasks with the probabilistic models.

4.1 Probability Theory

Probability theory concerns the probability of events and draws upon both propositional logic and statistics. The following introduction is adapted from several sources including [5, 61, 25, 24].

Propositions and events

We adopt the definitions, equivalences and properties of propositional logic in particular the notions of *propositional variables*, *logical connectives*, *propositions* and *truth*. Specifically, propositions are well-formed sentences formed

from propositional variables, constants and logical connectives. Propositional variables assume one of two basic values *true* or `false` and propositions can be evaluated to be either *true* or `false`. Basic propositional logic can be generalised to concern *multi-valued variables* that range over several possible values rather than *true* and *false*.

Definition 16 (Multi-valued variables) *A* multi-valued variable *is a variable, X, that ranges over a* domain*, $dom(X)$, of possible values and include two variants:*

- *A* discrete variable*, that ranges over a* countable*, possibly infinite set of possibles values.*
- *A* Continuous variable*, that ranges over* all *values in $\mathbb{R}$, possibly delimited by a certain minimum and maximum value.*

I will only be concerned with discrete variables ranging over *finite domains*. As is customary in statistics and probability theory, I use the term *event* instead of *proposition* and in the generalized propositional logic we get the following definition:

Definition 17 (Event) Events *are defined recursively from variables, values and logical connectives, $=, \neg, \wedge, \vee$, as follows:*

- *every variable X is an event.*
- *$X = x_i$ is an event, where X is a variable and x_i is one of its possible values.*
- *if ϕ and ψ are events, then so are*
  $\neg\phi$ (true when ϕ is not and false otherwise).
  $\phi \wedge \psi$ (true when both ϕ and ψ is true and false otherwise).
  $\phi \vee \psi$ (true when either or both of ϕ and ψ is true and false otherwise).

Furthermore, I adopt the conventions:

- Events of the form $X = x_i$, are called *variable instantiations* or just *instantiations*.
- $X = x_i$ may be written instead as x_i when the relation between variable and value is clear form the context.
- $\phi_1 \wedge \phi_2 \wedge \ldots$ may be written instead as $\phi_1, \phi_2, \ldots$.

event	X	Y	y_2	$\neg x_3 \vee y_2$
e_1	x_1	y_1	*false*	*true*
e_2	x_1	y_2	*true*	*true*
e_3	x_1	y_3	*false*	*true*
e_4	x_2	y_1	*false*	*true*
e_5	x_2	y_2	*true*	*true*
e_6	x_2	y_3	*false*	*true*
e_7	x_3	y_1	*false*	*false*
e_8	x_3	y_2	*true*	*true*
e_9	x_3	y_3	*false*	*false*

Figure 4.1: Instantiation table for Multi-valued variables, X and Y, each ranging over three distinct values, $\{x_1, x_2, x_3\}$ and $\{y_1, y_2, y_3\}$, respectively. Also shown are the truth-distributions of propositions $Y = y_2$ and $\neg X = x_3 \vee Y = y_2$.

Atomic events

In a system of n Multi-valued variables, an *atomic event* is a conjunction of instantiations of every variable in the system, i.e.:

$$e = \bigwedge_{i=1}^{n} x_i$$

or equivalently

$$e = x_1, x_2, \ldots, x_n$$

where $x_i \in dom(X_i)$.
The set of all atomic events in a system is specified by an *instantiation table*, as exemplified in figure 4.1 (first three columns). I will write, $e \models \phi$, to indicate that an event, ϕ, is true in an atomic event, e, and say that e *covers* ϕ. Any event, ϕ, can be completely specified by the disjunction of atomic events, e_i, that covers ϕ, i.e.:

$$\phi = \bigvee_{e_i \models \phi} e_i$$

For example as in figure 4.1. for events $Y = y_2$ and $\neg(X = x_3) \vee Y = y_2$. The event $Y = y_2$ is covered by atomic events $\{e_2, e_5, e_8\}$, and the truth of either implies the truth of $Y = y_2$. Similarly, $\neg(X = x_3) \vee Y = y_2$, is covered by atomic events $\{e_1, e_2, e_3, e_4, e_5, e_7, e_9\}$.

event	X	Y	$p(e)$
e_1	x_1	y_1	0.05
e_2	x_1	y_2	0.05
e_3	x_1	y_3	0.10
e_4	x_2	y_1	0.10
e_5	x_2	y_2	0.10
e_6	x_2	y_3	0.10
e_7	x_3	y_1	0.15
e_8	x_3	y_2	0.15
e_9	x_3	y_3	0.20

Figure 4.2: A *probabilistic model* of random variables X and Y from figure 4.1, and the associated *joint probability distribution* over the atomic events of that model.

Probabilities of events

Where propositional logic is concerned with the truths of events, probability calculus is concerned with the *probability* of events. In this context, atomic events, e, are regarded as possible outcomes of a *probabilistic model* and the probability, $p(e)$ of an atomic event is a real value in the range $[0; 1]$ that represents the relative degree of belief in the occurrence of that event; 0 meaning zero belief, and 1 meaning absolute certainty:

Definition 18 (Discrete Probability) *For a countable set of possible atomic events e – called a* sample space *and denoted Ω, we define a probability function $p : \Omega \mapsto [0; 1]$ such that:*

- $\forall e \in \Omega \ 0 \leq p(e) \leq 1$,
- $\sum_{e_i \in \Omega} \ p(e_i) = 1$.

I define here a *probabilistic model* in terms of *random variables* and *joint probability distributions*, basically an instantiation table where each atomic event has a distinct probability, see figure 4.2.

Definition 19 (Random variable) *A (discrete)* random variable *is a tuple, $\langle X, P(X) \rangle$, where:*

- *X is Multi-valued variable with a countable domain $dom(X) = \{x_1, x_2, \ldots\}$.*
- *$P(X) = \langle p(x_1), p(x_2), \ldots \rangle$, is called the* probability distribution *of X, and contains a* probability*, $p(x_i)$ for each element, $x_i \in dom(X)$*

Definition 20 (Probabilistic model) *A* probabilistic model *is a tuple* $\langle m, P(m)\rangle$*, where:*

- $m = \{X_1, \ldots, X_n\}$*, is a set of distinct random variables*
- $P(m)$*, is called the* joint probability distribution *of* M *and is a mapping,* $e \mapsto p(e)$*, from atomic events,* e*, in* m *to probabilities* $p(e)$ *of those events.* $P(m)$ *is sometimes referred to as the probabilistic* parameter *of* m*, denoted* θ_m *or just* θ*.*

Definition 21 (Marginal probability) *The* marginal *or* unconditional *probability of an event,* ϕ*, in a probabilistic model,* m*, is the probability of* ϕ *regardless of all other events in* m*. It is given as the joint probability of the atomic events that covers* ϕ *and since all atomic events of a model are distinct, we have:*

$$p(\phi) = \sum_{e_i \models \phi} p(e_i)$$

The marginal probability of an event of a model is also called the prior probability of that event.
An important concept that captures the possible dependency between events in a probabilistic model is the *conditional probability*, $p(\phi|E)$, of an event ϕ given that a set of events $E = \{\psi_1, \psi_2, \ldots \psi_n\}$, referred to as *evidence*, all occurred:

Definition 22 (Conditional probability) *The conditional probability* $p(\phi|E)$ *is defined from prior/marginal probabilities* $p(\phi)$ *and* $p(E)$*, assuming* $p(E) \neq 0$*, as follows:*

$$p(\phi|E) = \frac{p(\phi \wedge E)}{p(E)}$$

where $p(\phi \wedge E)$ *is the probability that both* ϕ *and all of* $\psi_1 \ldots \psi_n$ *occurs. The* conditional probability distribution

$$P(\phi|E)$$

is the set of probability distributions of ϕ *given all possible instantiations of the variables in* E*. We have that for any combination of values,* $\psi_1, \ldots, \psi_n$*, of the* n *evidence variables in* E*,* $\sum_{\forall x_i} p(x_i|\psi_1, \ldots, psi_n) = 1$*, as illustrated for* $P(X|Y)$ *in figure 4.3.*

X	Y	$p(X\|Y)$
x_1	y_1	0.1667
x_1	y_2	0.1667
x_1	y_3	0.2500
x_2	y_1	0.3333
x_2	y_2	0.3333
x_2	y_3	0.2500
x_3	y_1	0.5000
x_3	y_2	0.5000
x_3	y_3	0.5000

Figure 4.3: *Conditional probability distribution*, $P(X|Y)$, with reference to the variables in figure 4.2. Each entry contains the conditional probability $p(x_i|y_j)$ for particular instantiations of X and Y, calculated as defined in definition 22.

Rules of Probability theory

Conditionally independent events Two events, ϕ, ψ, in a probabilistic model are *conditionally independent* from each other, if the occurrence of one does not affect the probability of the other, i.e., iff:

$$p(\phi|\psi) = p(\phi)$$

or equivalently

$$p(\psi|\phi) = p(\psi)$$

Minus rule The *complement* of an event, ϕ, written here as $\neg\phi$, is the event that ϕ does not occur. We have the rule:

$$p(\neg\phi) = 1 - p(\phi)$$

Product rule The probability, $p(\phi \wedge \psi)$ also written $p(\phi, \psi)$, of the *conjunction* of two events, ϕ and ψ, is the probability that both ϕ AND ψ occur. We have the rule:

$$p(\phi, \psi) = p(\phi)p(\psi|\phi)$$

or equivalently:

$$p(\phi, \psi) = p(\psi)p(\phi|\psi)$$

For independent events this reduces to the product of marginal probabilities:

$$p(\phi_1, \ldots, \phi_n) = \prod_{1=1}^{n} p(\phi_i)$$

Sum rule The probability, $p(\phi \vee \psi)$, of the disjunction of two events, ϕ and ψ, is the probability that either OR both events occur. We have the rule

$$p(\phi \vee \psi) = p(\phi) + p(\psi) - p(\phi, \psi)$$

For a disjunction of mutually exclusive events, that can never occur together, this reduces to the sum of marginal probabilities:

$$p(\phi_i \vee \ldots \vee \psi_n) = \sum_{i=1}^{n} p(\phi_i)$$

Bayes conditioning Given the evidence of $p(\phi, \psi)$ and $p(\psi)$ we can regard the probability of ϕ in the light of positive evidence of ψ, i.e.:

$$p(\phi|\psi) = \frac{p(\phi, \psi)}{p(\psi)}$$

This is called *updating the belief* in ϕ given the evidence ψ.

4.2 Hidden Markov Models

Hidden Markov Models (HMM's) are probabilistic augmentations of finite state automata. This introduction is adapted primarily from [1, 2, 29, 61, 62]. In a HMM, the non-deterministic choices of which transition to make and which symbol to emit in each state, s_i, are each governed by a random variable ranging over the possible alternatives. The languages that are denoted are still regular.

By extending the FSA from definition 12 (p. 31) with conditional probability distributions, P^T and P^E, for transitions and emissions both depending on the state s_i, we get a first-order output HMM, defined as follows:

Definition 23 (HMM) *A first-order output HMM is an octuble* $M = \langle S, Begin, End, T, \Sigma, E, P^T, P^E \rangle$ *where:*

- $\langle S, Begin, End, T, \Sigma, E \rangle$ *is an FSA and*

- $P^T = \{p((s,t))|(s,t) \in T\}$ *is a set of probability distributions over possible transitions from each state and*
- $P^E = \{p((s,\beta))|(s,\beta) \in E\}$ *is a set of probability distributions over possible emissions from each state.*

There exists a wide variety of extensions to the these basic forms of HMM's, please see [19] for a more thorough survey of HMM-varieties.

Probability of a path in a HMM A parse $\xi(\omega)$ of a sequence ω corresponds to a path $\mathcal{P}(\omega)$ (definition 13, page 32) for that sequence in an equivalent FSA. The probability, $p(\xi(\omega))$, of a particular parse, $\xi(\omega)$, deriving the sequence $\omega = \beta_1\beta_2 \ldots \beta_n$ is the joint probability of the particular transitions and emissions in the parse, considering them as independent events, i.e. by the product rule,

$$p(\xi(\omega)) = \prod_{i=1}^{n} p((\pi_{i-1}, \pi_i)) \times \prod_{j=1}^{n-1} p((\pi_j, \beta_j))$$

Probability of a sequence in a HMM In a HMM M, the probability, $p(\omega)$, of some sequence, $\omega \in L^M$, is calculated as the combined probability of all parses $\xi_i(\omega)$ deriving ω in M, i.e., by the sum rule for mutually exclusive events,

$$p(\omega) = \sum_{\forall i} p(\xi_i(\omega))$$

Example 7 *To transform the FSA for reading frames in DNA from figure 3.2 (p. 34) into a first order output HMM, simply associate a transition probability with every edge and an emission probability with every emittable symbol in every state, as indicated in figure 4.4.*

4.3 Probabilistic Context free Grammars

The probabilistic augmentations of context-free grammars are called *probabilistic context-free grammars* (*PCFG*'s), sometimes also *stochastic context free grammars* (*SCFG*'s). This introduction is adapted from [1, 2, 29, 61, 62]. They are formed from context-free grammars by associating a probability distribution P^R with the rewriting rules, $A \rightarrow \beta_1|\beta_2|...|\beta_n$, for each nonterminal, A, thus turning each of them into a random variable ranging over its alternative right-hand-sides, β_j.

$s \in S$	*Emissions* β *from* s	$p((s,\beta))$	*Transitions to* t *from* s	$p((s,t))$
Begin			s_1	0.45
			s_{18}	0.55
s_1	a	0.25	s_2	1.00
	g	0.50		
	t	0.25		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_{20}	a	0.20	s_{18}	0.89
	c	0.15	s_1	0.10
	g	0.30	*End*	0.01
	t	0.35		

Figure 4.4: Sketch of specification of a HMM the basis of the DNA FSA in figure 3.2 on page 34.

Definition 24 (PCFG) *A probabilistic context-free grammar is a quintuple* $G = \langle N, \Sigma, R, S, P^R \rangle$ *where:*

- $G = \langle N, \Sigma, R, S \rangle$ *is a context-free grammar and*
- $P^R = \{p(A_i \to \beta_j) | A_i \to \beta_j \in R\}$ *is a set of probability distributions over alternative rewriting rules for each nonterminal in* G*.*

Probability of a parse in a PCFG For context-free grammars, the probability, $p(\xi(\omega))$, of a particular parse, $\xi(\omega) = \langle A_1 \Rightarrow \beta_1, A_2 \Rightarrow \beta_2 \ldots A_n \Rightarrow \beta_n \rangle$, deriving the sequence ω is the joint probability of the series of rules applied in the parse, considering them as independent events, i.e. by the product rule,

$$p(\xi(\omega)) = \prod_{i=1}^{n} p(A_i \to \beta_i)$$

Probability of a sequence in a PCFG In a PCFG, the probability, $p(\omega)$, of some sequence, $\omega \in L^M$, is the combined probability of all parses, $\xi_i(\omega)$, deriving ω, i.e., by the sum rule for mutually exclusive events,

$$p(\omega) = \sum_{\forall i} p(\xi_i(\omega)).$$

4.4 Probabilistic Inference

Three types of probabilistic inference are introduced here, *sampling*, *prediction* and *parameter estimation*. *Sampling* implements a basic statistical experiment with a probabilistic model. The motivation for employing probability in the first place was to resolve the ambiguity of the sequence models. This is referred to as *prediction* and is the primary task of probabilistic inference. *Parameter estimation* is another important task for probabilistic inference, that seeks to establish the parameter of a probabilistic model from observed data. The following introduction is adapted from [61, 62, 25].

Sampling

Given a probability distribution $P(e_i)$ defined over a finite sample space of atomic events, I will use the term, *sampling*, to refer to a statistical experiment consisting of picking an atomic event e_i from that space with probability $p(e_i)$. We also call such sampled outcomes *observations*. The law of large numbers state that if enough samples are made from a given sample space, the relative frequency of different observations converge toward their respective probabilities.

Prediction

The motivation for employing probability in the first place was to resolve the ambiguity of the sequence models. This is referred to as *prediction* and is the primary task of probabilistic inference. For the HMM for reading frames given in DNA from figure 4.4 (p. 45), a parse

$$\xi(\omega) = [(Begin, \pi_1), (\pi_1, \beta_1), (\pi_1, \pi_2) \dots (\pi_n, \beta_n), (\pi_n, End)]$$

completely explains how the DNA sequence ω is derived in that model.

When observing DNA, however, usually only the sequence of nucleotide symbols $\omega = \beta_1 \dots \beta_n$ are available, whereas the path $\Pi = \pi_1, \dots, \pi_n$ is unavailable. Examples like this are said to consist of both an *observed* part, here the symbol sequence ω, and a hidden part, here the path Π. Predictive inference refers to establish hidden data from observed data. In general, for a probabilistic sequence model M, like HMMs and PCFGs, with probabilistic parameter θ, the task of probabilistic predictive inference can be defined as finding a parse $\xi^*(\omega)$ with highest probability in that model, i.e.:

$$\xi^*(\omega) = \underset{\xi(\omega)}{\operatorname{argmax}}\, p(\xi(\omega)|\theta)$$

For a HMM M of m states and a symbol sequence ω of length n derivable in M, there are in the worst case m^n different parses $\xi(\omega)$. While a naïve comparison is clearly intractable, the well-known dynamic programming *viterbi-algorithm* [74] computes $\xi^*(\omega)$ in time $\mathcal{O}(m^2 n)$. The Viterbi algorithm can be adapted to efficiently apply to other classes of probabilistic models, including PCFGs.

Parameter estimation

Parameter estimation is basically a search task, i.e.,

Given: A probabilistic model and a set D of observed samples and their proper annotations, e.g., for sequence models training samples is of the form $(\omega, \xi(\omega))$.

Task: Find the parameter that maximizes the probability $p(D|\theta)$, called the *likelihood* of θ given the data D, which may be denoted $L(D|\theta)$

Note, that the two terms *probability* and *likelihood* refers to the same numeric value namely, $p(D|\theta)$. The term *probability* indicates that the value is to be regarded as a function of the data D, whereas *likelihood* indicates that it is to be regarded instead as a function of the parameter θ. The probability $p(D|\theta)$ is the combined probability of all the parses $\xi(\omega)$ in D and with regard to the probability of a path in a HMM (section 4.2), we get:

$$\begin{aligned} L(D|\theta) &= p(D|\theta) \\ &= \sum_{\xi(\omega)\in D} p(\xi(\omega)|\theta) \\ &= \sum_{\xi(\omega)\in D} \left(\prod_{(s,t)\in\xi(\omega)} p((s,t)|\theta) \prod_{(s,\beta)\in\xi(\omega)} p((s,\beta)|\theta) \right) \end{aligned} \quad (4.1)$$

Supervised learning - learning from fully observed data

Consider the HMM sketched in figure 4.4 on page 4.4, and assume that the probability distributions P^E and P^T are unknown. Assume fully observed parses, i.e., both the symbol sequence ω and path Π of each example in D are given. In this case, parameter estimation simply amounts to counting the frequencies of the various emissions and transitions from each state and set the probabilities accordingly.

Take as an example state, s_{20} of the HMM. For each of the parses in D we count the number of times and a, c, t or g was emitted from s_{20}. Let us name those frequency counts $f((s_{20}, a))$, $f((s_{20}, c))$, $f((s_{20}, t))$ and $f((s_{20}, g))$. Then set the emission probabilities, $p((s_{20}, \beta_i)) \in P^E$ as:

$$p((s_{20}, \beta_i)) = \frac{f((s_{20}, \beta_i))}{\sum_\beta f((s_{20}, \beta))}$$

Similarly for transition probabilities $p((s_{20}, s_i)) \in P^T$.

Unsupervised learning - learning from partially observed data

Consider now the case where only the symbol sequences ω_i are given and the paths $\Pi(\omega_i)$ are hidden. In this case, we seek to find the parameter with highest likelihood given the data D, i.e., $\theta^* = \text{argmax}_\theta L(D|\theta)$. We can construct a complete dataset D' containing all possible parses $\xi_i(\omega)$ for every observed ω in D and count the frequencies $f((s,t))$, $f((s,\beta))$, as before. This time we weigh each of them according to their probability given a candidate parameter θ. The maximum likelihood parameter can now be defined as:

$$\theta^* = \underset{\theta}{\text{argmax}} \sum_{\xi(\omega) \in D'} \left(\prod_{(s,t)\in\xi(\omega)} f((s,t)) \times p((s,t)|\theta) \prod_{(s,\beta)\in\xi(\omega)} f(s,\beta) \times p((s,\beta)|\theta) \right)$$

The well-known general *expectation maximization algorithm*, *EM*, (see for example [25]) is widely used for approximating the maximum likelihood parameter θ^*. It can be sketched roughly as follows for the HMM example:

1. Complete the dataset to get D'.
2. Assume some current parameter, θ^i.
3. Iterate the following two steps until the parameter and likelihood converge:

E-step: Count the frequencies $f((s,t))$ and $f((s,\beta))$ of the various transitions and emissions in the completed dataset D', exactly as for supervised learning; only this time, each frequency count is weighted by its probability given θ^i to form *expectation counts*, $ec((s,t))$ and $ec((s,\beta))$:

$$ec((s,t)|\theta^i) = f((s,t)) \times p((s,t)|\theta^i)$$

and

$$ec((s,\beta)|\theta^i) = f(s,\beta) \times p((s,\beta)|\theta^i)$$

M-step: Set new probabilities according to expected counts to form the new parameter θ^{i+1}:

$$p((s,t)|\theta^{i+1}) = \frac{ec((s,t)|\theta^i)}{\sum_\beta ec((s,t)|\theta^i)}$$

and similarly for the emission probabilities:

$$p((s,\beta)|\theta^{i+1}) = \frac{ec((s,\beta)|\theta^i)}{\sum_\beta ec((s,\beta)|\theta^i)}$$

The EM algorithm guarantees convergence on some local maximum likelihood parameter for every probabilistic model. While only a local maximum is guaranteed, there are several tricks to increase the chance of a global maximum likelihood result under various circumstances. Furthermore, for many families of probabilistic models there exist efficient specialised dynamic programming versions of the general EM-algorithm, like the *Baum-Welch algorithm* for HMMs. These will however not be described here.

Chapter 5

Probabilistic Logic Programming

Several general and powerful formalisms seeking to join logic and probability theory have been suggested the last 15 years. I exemplify here the PRISM system, [65, 66], that is a probabilistic extension to B-Prolog [79] – an distribution of the standard Prolog logic programming language. Being a declarative language, PRISM specifications of arbitrarily complex relationships are still comparatively clear and concise. PRISM inherits from Prolog a powerful mechanism for finding all solutions to a query, known as backtracking. Because of the potential very large solution space to be searched this way, Prolog and PRISM are in principle less efficient than more traditional programming languages. BProlog employs tabling to reduce the need for re-evaluation of already evaluated sub-goals and the dynamic programming Viterbi algorithm avoids searching the entire space. Other optimizations to the general performance have also been developed in the LoSt-project, [14]. Finally, the raw power and parallel capabilities of modern computers has rendered this drawback more or less negligible, except where speed is crucial to the application. For gene-finding and DNA-annotation, annotation-accuracy is arguably more interesting than execution time.

5.1 The PRISM programming language

PRISM is a programming language intended for probabilistic logic modelling. PRISM extends Prolog and essentially replaces the Prolog notion of truth with one of probability. To illustrate the semantics of PRISM, I first sketch the syntax and declarative semantics of Prolog based on proof trees (see for instance [72, 51]). Then I sketch the probabilistic extension.

Syntax and declarative semantics of Prolog

Prolog is a logic programming language defined in terms of Horn clauses which is a subset of first order predicate logic, and from here I adopt the associated notion of truth. Here I define truth and proof-trees directly for Prolog.

In the following I assume the following distinct sets:

- *Constants*; positive integers and strings starting with a lower-case letter, or quoted strings.
- *Variables*; strings starting with an upper case letter or "_"; the latter indicating distinct anonymous variables.
- *Functions*; f/n, where f is the function symbol and n is the arity.
- *Predicates*; p/n, where p is the predicate symbol and n is the arity.

Definition 25 (Terms) *A* term *in Prolog is defined inductively as follows:*

- *constants and variables are* terms.
- *If f/n is a function symbol of arity n and $\tau_1 \ldots \tau_n$ are* terms *then so is $f(\tau_1, \ldots, \tau_n)$.*
- *nothing else is a* term *in Prolog.*

Terms of the form $f(\tau_1, \ldots, \tau_n)$ are called compound terms.
Terms with no variables are called ground *terms.*

Definition 26 (Atoms) *If p/n is a predicate symbol of arity n and $\tau_1, \ldots, \tau_n$ are terms, then*

- *$p(\tau_1, \ldots, \tau_n)$ is an atom.*
- *nothing else is an* atom *in Prolog.*

Atoms are the building-blocks of Prolog *programs.*

Definition 27 (Clauses) *A Prolog* clause *is a structure of the form*

$$A \texttt{:-} \quad B_1, \ldots, B_n$$

where $n \geq 0$ and both A and all the $B's$ are atoms.
A is called the head *of the clause and $B_1, \ldots, B_n$ is called the* body.

- *A* fact*, written simply as* `A.`*, is a clause where $n = 0$,*
- *A* rule *is a clause where $n > 0$.*

Definition 28 (Prolog Programs) *A Prolog* program *P is a finite set of Prolog clauses.*

Definition 29 (Herbrand Base) *For a Prolog program P, the* Herbrand Base *of P, $\mathcal{B}(P)$, is the set of all ground atoms that can be constructed from the constants, function symbols and predicate symbols in P.*

Each atom $\alpha \in \mathcal{B}(P)$ has a truth value defined in terms of *proof trees*:

Definition 30 (Proof trees) *Given a Prolog program P the set of proof trees is defined inductively as follows:*

- *If α is a ground instance of a fact in P, then T_α is a proof tree rooted in α with no sub-trees.*
- *If α is an atom such that there is a ground instance, $\alpha :- \beta_1, \ldots, \beta_n$, of a rule in in P and $T_{\beta_1}, \ldots, T_{\beta_n}$ are proof trees rooted in $\beta_1, \ldots, \beta_n$, respectively, then T_α is a proof tree rooted in α having $T_{\beta_1}, \ldots, T_{\beta_n}$ as immediate sub-trees.*
- *Nothing else is a proof tree.*

Definition 31 (Truth) *An atom $\alpha \in \mathcal{B}(P)$ is* TRUE *iff a* proof tree *rooted in α can be constructed from the rules and facts in P and* FALSE *otherwise.*

Definition 32 (Declarative program meaning) *The* declarative meaning *P of a Prolog program P is defined as the subset of* TRUE *atoms in $\mathcal{B}(P)$*

Syntax and semantics of PRISM

PRISM extends Prolog with a notion of probabilistic variables called *Multi-valued switches* enabling the specification of probabilistic programs.

A Multi-valued switch S with outcome space $\{V_1 \ldots V_n\}$ is declared in PRISM by use of the `values/2` built-in predicate:

`values(`S`,[`$V_1, \ldots, V_n$`])`

The *switch identifier* S must conform to a Prolog term – including complex terms, so that for example `die`, `die(red,1)`, `die(red,2)` and `die(blue,1)` are all legal (distinct) switch identifiers. On this note, Prologs anonymous variable (initiated by "_"), works as usual so that for example `die(red,_)` will unify with any term that matches that particular pattern. A uniform probability distribution over the outcomes space is assumed per default, but a specific distribution may be set explicitly by use of the `set_sw/2` built-in predicate:

`set_sw(`S`,[`$p(V_1), \ldots, p(V_n)$`])`

The probability distributions for all Multi-valued switches in a PRISM-program corresponds to the parameter of a probabilistic model.

In PRISM, a Multi-valued switch defines a *family* of random variables with the same switch identifier and outcome-space. An independent *switch instance* of this family is generated and instantiated probabilistically to a value in the outcome-space by a call to the built-in predicate `msw/2`:

`msw(`S, V`)`

where $V \in \{V_1 \ldots V_n\}$.

Note that individual calls to `msw/2` are independent so that any call, `msw(`S, V`)`, will generate a new instance of S and instantiate it anew, i.e., independently from other such calls, to one of the possible outcomes of S.

PRISM atoms can be defined as for Prolog (definition 26, p. 52) with the addition that ground instances of Multi-valued switches, `msw(`S, V`)` also are *atoms*, for use exclusively in the body of clauses. In these terms, *probabilistic predicates* are predicates that depend on one or more probabilistic choices, i.e., calls to `msw/2`.

Clauses and programs in PRISM are defined as for Prolog (definitions 27 and 28, on page 52), with the constraint that for a program to be a legal PRISM program, all non-determinism must be resolved by calls to `msw/2`. Note, however, that a non-probabilistic interpretation of a PRISM program can be achieved simply by assuming that `msw/2`-calls always succeed or, alternatively, by redefining `msw/2` simply as the Prolog built-in `member/2`-predicate.

The *Herbrand Base* $\mathcal{B}(P)$ for a probabilistic program P is also defined as for Prolog programs (definition 29, p. 53), except that here each probabilistic atom $\alpha \in \mathcal{B}(P)$ is either FALSE or has a non-zero probability, defined in terms of *explanation tree* as follows.

Definition 33 (Explanation tree) *Given a PRISM program P the set of explanation trees is defined inductively as follows:*

- *If α is a ground instance of a fact in P, then E_α is an explanation tree rooted in α with no sub-trees and probability $p(E_\alpha) = 1$.*
- *If α is a ground instance, `msw(S,V)`, of a Multi-valued switch in P, then E_α is an explanation tree rooted in `msw(S,V)` with no sub-trees and probability $p(E_\alpha) = p(V)$.*

- *If α is an atom such that there is a ground instance, $\alpha :- \beta_1, \ldots, \beta_n$, of a rule in P and $E_{\beta_1} \ldots E_{\beta_n}$ are explanation trees rooted in $\beta_1 \ldots \beta_n$, with probabilities $p(E_{\beta_1}) \ldots p(E_{\beta_n})$, then E_α is an explanation tree rooted in α having $E_{\beta_1} \ldots E_{\beta_n}$ as immediate sub-trees and probability $p(E_\alpha) = \prod_i p(E_{\beta_i})$.*
- *Nothing else is an explanation tree.*

An explanation tree rooted in an atom α *is also called an* explanation *for* α.

Definition 34 (Probability of atoms) *Given a PRISM program P every (ground) atom in the Herbrand Base $\mathcal{B}(P)$ has a probability defined as follows:*

- *An atom $\alpha \in \mathcal{B}(P)$ has probability $p(\alpha) = \sum_i P(E^i_\alpha)$, where E^i_α are all* explanation trees *rooted in α and $p(E^i_\alpha)$ are their respective probabilities.*
- *Iff no* explanation tree *rooted in α can be constructed, then α has zero probability.*

Under the basic distribution semantics as defined by Taisuke Sato in [64], a PRISM program defines a probability distribution over the ground atoms in the associated Herbrand Base if the following conditions are met:

1. *Exclusiveness condition*, stating that disjunctive paths in a proof-tree must be *probabilistically exclusive*, basically meaning that all choice in the program must be governed by multi-valued switches.

2. *Uniqueness condition*, stating that all solutions to any goal must be probabilistically exclusive and their probabilities sum to 1[1].

In these terms we may define the declarative probabilistic meaning of PRISM programs as follows:

Definition 35 (Meaning of probabilistic programs) *The* probabilistic declarative meaning *of a PRISM program P under the distribution semantics is defined as the subset of ground atoms in $\mathcal{B}(P)$ with non-zero probabilities.*

Note, however, that any PRISM-program can be regarded as a Prolog-program by assuming a non-probabilistic functionality of `msw/2`, e.g., calls

[1]This ultimately implies the potentially very critical restriction that no goal may ever fail in a PRISM program. The newer versions of PRISM take such failures into account and thus relax the uniqueness condition, [67].

to `msw/2` always succeed or, alternatively, `msw/2` simply mimics the membership-predicate `member/2`. An annotation program defined this way constitutes the logic part of a *annotation model*, where the probabilistic parameter θ constitutes the probabilistic part.

5.2 Probabilistic sequence models in PRISM

The PRISM system is well suited for specifying probabilistic grammar formalisms and parsers, which I illustrate for HMM's and PCFG's in this section.

Hidden Markov Models in PRISM

Recall from definitions 12 (p. 31) and 23 (p. 23), that an HMM is defined as the octuple $\langle S, Begin, End, T, \Sigma, E, P^T, P^E \rangle$ where:

- S is the set of states,
- $Begin \in S$ is a distinguished *begin state*,
- $End \in S$ is a distinguished *end state*,
- $T \subseteq S \times S$ is a set of directed *state transitions*.
- Σ is the alphabet of *emittable symbols*,
- $E \subseteq S \times \Sigma$ is a set of possible *state emissions*.
- $P^T = \{p((s,t))|(s,t) \in T\}$ is a set of transition probabilities and
- $P^E = \{p((s,\beta))|(s,\beta) \in E\}$ is a set of emission probabilities.

In PRISM, HMM's can be defined concisely in terms of their states, S, the possible transitions and emissions from each state $s \in S$, $T(s)$ and $E(s)$, and their corresponding probability distributions. To model, for example, the DNA-HMM from example 7 (p. 44) following the diagram in figure 3.2 (p. 34) can be precisely specified in PRISM by declaring the components as follows:

```
  % (begin and end states)
initial(begin).
final(end).
  % (transitions in $T$)
values(trans(begin),[s(1), s(18)]).
values(trans(s(1)),[s(2)]).
  ...
```

```
values(trans(s(20)),[s(1),s(18),end]).
  %(emissions in $E$}
values(emit(s(1)),[a,g,t]).
  ...
values(emit(s(20)),[a,c,g,t]).
```

The respective probability distributions can be specified – using proper instances of `set_sw/2`, or the default uniform distributions may be accepted as is.

To generate a sample sequence from these specification we need a general parser for Hidden Markov Models, implemented in PRISM for example as follows:

```
hmm(Sequence,Path):-
  initial(BeginState)
  msw(trans(BeginState),State1),
  hmm(State1,Sequence,Path).

hmm(ThisState,[],[]):- final(ThisState).
hmm(ThisState,[ThisEmission | RestEmissions],[ThisState|RestPath]):-
  \+ final(ThisState),
  msw(emit(ThisState), ThisEmission),
  msw(trans(ThisState), NextState),
  hmm(NextState, RestEmissions, RestPath).
```

The program consists of a top-predicate `hmm/1` and a recursive predicate `hmm/2`. The top-predicate establishes the initial state by a call to the goal `msw(trans(BeginState),State1)` and initiates the recursion by a call to goal `hmm(State1,Seqeunce)`. The clauses of the recursive predicate establishes what symbols to emit and what state to transit to from each state, as governed by the two `msw/2`-calls. Recursion continues until the `end-state` is reached. Note, that the use of negation as failure (`\+`) may have undesirable effects in some programs due to backtracking, but is unproblematic here since in each case there is only one possible clause to choose.

Given an instance of sequence S that is explained by the probabilistic model, the built-in predicate `prob/1`:

```
:- prob(hmm(S))
```

implements an algorithm for computing

$$p(\texttt{hmm(S)}) = \sum_{E_{\texttt{hmm(S)}}} p(E_{\texttt{hmm(S)}})$$

i.e., the sum of the probabilities of all explanation tress rooted in `hmm(S)`, given the rules of the hmm-program and current probability distributions.

Probabilistic context-free grammars in PRISM

PCFG's are modelled in PRISM just as straightforwardly as were HMMs. Recall from definitions 14 (p. 33) and 24 (p. 45)that PCFG's are defined as the quintuple $G = \langle N, \Sigma, R, S, P^R \rangle$ of nonterminal symbols, N, terminal symbols, Σ, rewriting rules, R with corresponding probability distributions, P^R and a unique start symbol S, such that each nonterminal rule in the grammar is in essence a random variable ranging over its various rewriting alternatives.
Consider again the *HairpinLoop* grammar from example 6 on page 35,

$$G_3 = \langle \{HairpinLoop, Stem, Loop\}, \{a, t, c, g\}, R, HairpinLoop, P^R) \rangle$$

with grammar rules in R:

$$\begin{aligned} HairpinLoop \rightarrow &\quad a\ Stem\ t \mid t\ Stem\ a \mid c\ Stem\ g \mid g\ Stem\ c \\ Stem \rightarrow &\quad a\ Stem\ t \mid t\ Stem\ a \mid c\ Stem\ g \mid g\ Stem\ c \mid Loop \\ Loop \rightarrow &\quad a\ Loop \mid t\ Loop \mid c\ Loop \mid g\ Loop \mid \\ &\quad a \mid t \mid c \mid g \end{aligned}$$

This can be precisely specified in PRISM by declaring components, S, N, Σ, *and* R as follows:

```
  % (grammar symbols)
startsymbol(hairpinLoop).

nonterminal(hairpinLoop).
nonterminal(stem).
nonterminal(loop).

terminal(a).
terminal(t).
terminal(c).
```

```
terminal(g).

  % (rules in $R$)
values(hairpinLoop,[  [a,stem,t],
                      [t,stem,a],
                      [c,stem,g],
                      [g,stem,c]]).

values(stem,[         [a,stem,t],
                      [t,stem,a],
                      [c,stem,g],
                      [g,stem,c],
                      [loop]]).

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

note that each nonterminal is represented by a separate multi-valued switch an outcome space corresponding to its alternative rewriting rules in the grammar. As for HMMs, the probability distributions in P^R can be specified explicitly or we may (for now) accept the default uniform distributions.

Once again, we need a general generative parser for sampling execution. Using variable `RHS` for the alternative *right-hand sides* of rules, we can implement a parser for PCFGs as follows:

```
  % (top-predicate)
pcfg(Sequence):-
  startsymbol(Start),
  msw(Start,RHS),
  pcfg(RHS,Sequence).

  % (Cases of the recursive predicate)
  % (endcase)
pcfg([],Sequence):- Sequence = [].

  % (nontermainal case)
pcfg([First|Rest],Sequence):-
  nonterminal(First),
  msw(First,RHS),
```

```
  pcfg(RHS,Prefix),
  pcfg(Rest,Postfix),
  append(Prefix,Postfix,Sequence).

  % (terminal case)
pcfg([First|Rest],Sequence):-
  terminal(First),
  pcfg(Rest,Postfix),
  Sequence = [First|Postfix].
```

As for HMMs the parser consists of a top predicate and a recursive predicate. When the top predicate `pcfg/1` is called, it establishes the unique start-symbol and randomly chooses one of its rewriting-rules by calling the goal `msw(Start,RHS)`, with which it initiates the recursive predicate `pcfg/2`. Depending on the first symbol in the right-hand-side list of symbols, different clauses of the recursive predicate is matched:

- if the right-hand-side is empty, the empty list is returned.
- if the right-hand-side starts with a nonterminal, one of its rewriting rules is chosen and parsed recursively to the terminal sequence `Prefix`. Then the reaming right-hand-side is parsed recursively to the terminal sequence `Postfix`. Finally Postfix is appended to Prefix to form the final output sequence.
- if the right-hand-side start with a terminal symbol, it is simply copied to the output sequence of the remaining right-hand-side as established by the recursive call `pcfg(Rest,Postfix)`.

5.3 Probabilistic inference in PRISM

Apart from the built-in predicates `values/2`, `set_sw/2`, `msw/2` and `prob/1` already mentioned, PRISM includes a large collection of built-in tools in many variations for probabilistic analysis, including efficient implementations of well known algorithms for sampling, prediction and learning that, I will describe briefly.

Sampling in PRISM

Recall from section 4.4, that sampling refers to the statistical experiment of picking an atomic event e_i from a sample space with probability $p(e_i)$. When properly parametrised, a sequence model in PRISM represents a probability distribution over all possible sequence that can be derived from it. In the following I sketch the particular processes for HMMs and PCFGs in particular.

Sampling in PRISM A random sample sequence from the PRISM program, `hmm/1`, from section 5.2 is generated by the calling the built-in predicate `sample/1`:

```
?- sample(hmm(S)).
```

Similarly, a sample sequence given the probabilistic program `pcfg/1` is also generated as the result of a call of `sample/1`:

```
?- sample(pcfg(S)).
```

In which case the respective parsers are evaluated as described above to generate random sequences from the corresponding grammars.

Prediction in PRISM

Predictive inference in PRISM is catered for by a family of built-in predicates implementing different variations of a generalized Viterbi algorithm. The generalised Viterbi algorithm finds a most probable explanation, E_G^*, for a ground probabilistic atom G, given the PRISM rules and the current probability distributions. I.e., the call:

```
?- viterbig(G,P,E).
```

implements

$$E_G^* = \underset{E_G}{\operatorname{argmax}}\, p(E_G)$$

and instantiates `E` to E_G^* and `P` to $p(E_G^*)$.

As an important functionality of the viterbig-predicates in PRISM, all non-ground variables in the goal G becomes grounded to specific values that correspond to the maximal overall probability.

Example 8 *If `hmm/1` defines a parametrised HMM in PRISM as described in section 5.2, (parser repeated here)*

```
hmm(Sequence,Path):-
  initial(BeginState)
  msw(trans(BeginState),State1),
  hmm(State1,Sequence,Path).

hmm(ThisState,[],[]):- final(ThisState).
hmm(ThisState,[ThisEmission | RestEmissions],[ThisState|RestPath]):-
```

```
  \+ final(ThisState),
  msw(emit(ThisState), ThisEmission),
  msw(trans(ThisState), NextState),
  hmm(NextState, RestEmissions, RestPath).
```

and S is a ground sequence that can be derived from that model, then the query

`viterbig(hmm(S),P,E)`

will establish the most likely explanation E for S, and instantiate the argument P to the probability of that explanation.

If we extend the model to a version, `hmm/2`*, with an extra argument* `Path` *to record the states of paths as follows:*

```
hmm(Sequence,Path):-
  initial(BeginState)
  msw(trans(BeginState),State1),
  hmm(State1,Sequence,Path).

hmm(ThisState,[],[]):- final(ThisState).
hmm(ThisState,[ThisEmission | RestEmissions],[ThisState|RestPath]):-
  \+ final(ThisState),
  msw(emit(ThisState), ThisEmission),
  msw(trans(ThisState), NextState),
  hmm(NextState, RestEmissions, RestPath).
```

the query

`viterbig(hmm(S,A),P,E)`

will in addition instantiate `Path` *to a list of those states that corresponds to the most likely path deriving S, i.e., as an alternative to the full explanation tree E.*

Parameter estimation in PRISM

PRISM has a family of built-in predicates for parameter estimation, given a suitable set of training goals, i.e., a file of ground instances of the top-predicate. The simplest is `learn/1`, that implements the general EM-algorithm for approximating the maximum likelihood parameter, θ^*, given the training data, D, i.e.:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, L(D|\theta)$$

and sets the current parameter values accordingly.

Example 9 *Given the extended hmm* ,`hmm/2`*, from the previous example and a list of ground instances of goals,*

$$D = [\texttt{hmm}(S_1, A_1), \ldots, \texttt{hmm}(S_n, A_n)]$$

the query:

```
learn(D)
```

will invoke the built-in learning functionality of PRISM and set the individual switch-probabilities according to the recorded paths for the sequences in the training data D.

Summing up - Part I

In this part of the dissertation, I have defined the necessary biological, linguistic and probabilistic foundations for describing the problem context of applying probabilistic logic programming for DNA-analysis. In chapter 2, I explained the *central biological dogma*: transcription of DNA to mRNA and translation of mRNA to proteins. The important notions of *basepairing*, *codons*, *genetic code* and *reading frames* were introduced and explained. I also briefly introduced the concept of *DNA-annotation* and made the distinction between *extrinsic* and *intrinsic* approaches for *gene finding*. In chapter 3, I presented the formal definitions and background for formal language description, introducing formal grammars, automata and the notion of ambiguity. I indicated through examples how these formalisms easily lend themselves to the description of problems related to DNA-annotation, but also that the inherent ambiguity of the biological domain needs resolution. In chapter 4, I included the basic probabilistic foundations necessary for showing how probability theory offers a robust way of resolving ambiguity. Probabilistic grammars and automata were defined in chapter 4. In chapter 5, I briefly defined the syntax and semantics of the PRISM system and showed how probabilistic models can be formulated as PRISM-programs and how to perform the most important inferential tasks with those programs: *sampling*, *probability calculation*, *prediction* and *parameter estimation*.

Part II

Methodology and Examples

Chapter 6

Complex Sequence Analysis

In this chapter I define the proposed compositional approach, *Bayesian Annotation Networks* for integration of probabilistic models for sequence analysis. In chapter 7 and 8, I describe in detail two example applications to the domain of DNA-annotation and the experiments that were carried out with them. The approach evolved from experiments with preprocessing as a means of negotiating complexity of a complex annotation task

We attempted to first identify the different analytical tasks as being either simple or demanding in terms of computational complexity. We then decompose the overall model accordingly, i.e., into a specialised sub-model for each class of tasks. By means of yet another sub-model, called a *chopper*, for simple pre-analysis of the data-sequence, we attempt to distinguish corresponding types of sub-sequences: those, that require the complex analysis and those, that can make do with the less sophisticated analysis. Each sub-sequence is then submitted to the sophisticated annotation model of its type, and the individual sub-annotations combined to produce an annotation of the original sequence. That is, we follow an algorithm along these lines:

1. Apply the pre-processor to distinguish sub-sequences according to type.
2. Submit each sub-sequence to complex annotation-analysis according to type.
3. Append the annotations.

This work – published also in part in [20] (2008) and [42] (2008) – forms the substrate of chapter 8 of the dissertation.

At least one important insight in was gained from these early experiments. Namely, that when we, like this, distinguish first and perform complex analysis later, then the complex analysis obviously has no influence what so ever on the distinction. In many cases, where the pre-processor represents a very simple model without the sophistication of the specialised annotation-model, this corresponds to letting the blind lead the seeing. Therefore, if we want the sophisticated annotation programs to influence the distinction and classification of sub-sequences, we are required to annotate first and distinguish later. This let me to think of the overall model in terms of a tree-like structure (a directed acyclic graph, really) of sub-models, each with their special responsibility be it chopping, feature-analysis, or integrating, and each depending on the annotations of one or more of the others. It was observed that the idea resembled Bayesian Network only with annotation models rather than random variables in the nodes. From this evolved the general methodology of organizing specialized sub-models according to their interdependencies in a topology, that we now call a *Bayesian Annotation Network* (BAN).

6.1 Introduction

Having established the theoretical background and basic tools for biological sequence analysis and probabilistic logic programming, I can now present the research problem of my dissertation clearly – and in the proper context.

Recall from chapter 2.6, that by *genome-* and *DNA-annotation*, I refer to the identification and marking of features and properties in a sequence of DNA, that could be relevant for identifying genes in that sequence. The relationship between DNA and genes, is a complex one that involves a lot of biochemical mechanisms and aspects not all of which are entirely understood. While automatic genefinders do take a number of important aspects into regard, a small percentage of the genes that we know to exist in well studied genomes, for example that of Escerichia Coli, evade state of the art gene-finders like Glimmer [63, 28, 27], Genemark [7, 44, 6], and EasyGene [41, 50]. In fact, around 2% of the known genes in E.Coli go undetected by all the state of the art genefinders. The consistent failure to detect what we know to be there, only supports what is generally assumed, namely that there exist genes, even in well studied genomes, that have not been discovered yet, see for example [75] and elsewhere. Consequently, it is going to take quite a big effort, both biologically and computationally, to identify the remaining relevant features and increase the predictive power of automatic systems for gene-finding. To this end, a shift to a declarative

probabilistic-logic paradigm, as represented by the PRISM system, may contribute usefully by:

- increased expressive power - programs in these languages are generally shorter and more concise than implementations in procedural languages,
- executable problem specifications - once a problem has been properly specified, that specification functions as a working implementation for the solution of the problem,
- clear and consistent semantics based on classical 1^{st} order logic.

In part I, I demonstrated how comparatively easy it is to implement simple Hidden Markov models and stochastic context-free grammars in PRISM. The inherent modularity of predicate logic that is integral in PRISM, allows for flexible systems, where clear and well defined alterations and adaptations can be specified. That being said, PRISM is just as vulnerable to high complexity problems as any other programming language. The sheer size of data-instances involved in DNA-annotation, means that care has to be taken to keep computational complexity in check. Furthermore, DNA-annotation is extremely susceptible to the curse of dimensionality; i.e., as the number of parameters under consideration in a predictive system increases, the predictive power of the system tend to decrease.

As already mentioned in the introduction, this presents the non-trivial challenge, that is the motivating research question of the LoSt project as a whole:

How to implement clear, flexible and efficient systems using PRISM for accurate DNA-annotation?

As a partial problem of this overall goal the research question of the present dissertation can be formulated:

Can we establish a system of compositionality for probabilistic annotation programs in PRISM that retains the strengths of declarative programming but keeps computational complexity low enough for practical application?

I explore here an approach to the answer of that question, that involves two universal problem solving strategies: *problem decomposition* and *data partitioning*.

The compositional approach

The basic idea is to analyse a selection of individual features in isolation with separate *annotation models* and afterwards integrate the results in a compound analysis. To keep the size of data-instances manageable, suitable ways of partitioning data are also proposed. Together these strategies seek to rise to the general challenges of the LoSt-project by:

- reducing the number of required parameters in individual annotation models,
- allowing optimization of individual models with minimal regard to irrelevant factors,
- keeping the overall analysis flexible enough for easy comparison and experimentation with alternative combinations of feature signals.

This overall strategy raises several questions that will be addressed in this chapter: In section 6.2, I will discuss the different kinds of annotations that the methodology will involve and what I mean by *annotation models*. For any kind of DNA-annotation to be efficient, it is quite necessary to break up the extremely long genomic sequences into manageable chunks and in section 6.3, I discuss alternative strategies for doing so. I propose, in this work, a method for integrating information from separate annotations that is inspired from classical Bayesian Networks. It involves what we call *Bayesian Annotation Networks* or simply *BANs* which are defined and discussed in section 6.4. Different BAN's integrate individual annotations differently and section 6.6 discuss systematic comparison and evaluation of alternatives.

6.2 Annotations, programs and models

DNA-annotation in general, refers to any mark-up of the DNA-sequence representing some aspect of it. A *DNA-annotation program* then, is simply a computer program that analyses an input DNA-sequence with regard to some aspect and produces a suitable annotation representing the result of the analysis, i.e., in logic programming terms:

Definition 36 *An* annotation program*, or just* a program*, is a PRISM program prog, that defines a set of atoms, each of the form:*

$$prog(s, a, parents),$$

where

- *s is called* the sequence*, and represents the data sequence to be annotated by the program.*
- *a is called an* output annotation*, and*
- *parents represents zero or more* conditioning annotations.

The "*parents*" argument in this definition, anticipates the introduction of Bayesian Annotation Networks in section 6.4 below. Parent annotations represent annotations produced by other annotation-programs, serving as conditions for the analysis associated with *prog*.

Definition 37 *A* probabilistic annotation model

$$m = \langle prog, \theta \rangle$$

consists of an annotation program prog and a parameter θ*. The parameter is a set of specific probability distributions for the Multi-valued switches in the program, giving rise to a well-defined conditional probability distribution (def. 22, p. 41) for annotation programs,* $prog(s, a, parents)$*:*

$$P(a \mid s, parents, \theta)$$

The framework captures also analyses that are not necessarily written in a probabilistic-logic language.

Definition 38 *A* deterministic annotation model *is a program*

$$prog(s, a, parents)$$

where, for a specific sequence s^0 *and* $parents^0$*, there exists exactly one output annotation* a^0*, i.e.,*

$$P(a^0 \mid s^0, parents^0, \theta) = 1,$$

where θ*, in this case, refers to an empty parameter which is ignored.*

The empty parameter is included for uniformity of notation only. A deterministic annotation model with empty parents may for example represent an analysis provided by an external tool that, e.g., searches for similarities in a database of related sequence data or some other deterministic analysis.

Example 10 *Figure 6.1 illustrates an example of a simple probabilistic annotation model, that considers a specific nucleotide* preference *as an indication for coding potential in un-annotated DNA, S. This example is adapted from an example in [29] for illustrating hidden Markov models. Because*

there is a functional relationship between a protein and the nucleotides in the encoding gene, it may be assumed that the nucleotides in protein-coding genes are distributed differently than nucleotides in non-coding regions of DNA. Such a difference in bias of the two distributions (referred to as a preference*) may be sufficient to distinguish coding regions from non-coding regions. It is very straightforward to implement, for example, as a first order output HMM with two states, one for coding,* `cs`*, and one for non-coding,* `ns`*. Both states can be initial and final and both have transitions to themselves and each other. Each of the states* `cs` *and* `ns` *emits one of the four nucleotides a, c, t or g before making a transition to the next state, and the emitting state is recorded along the way as an annotation of that position in the sequence. The parameter, θ, of the model, consisting of the transition and emission probabilities, can be explicitly specified or estimated from existing training data.*

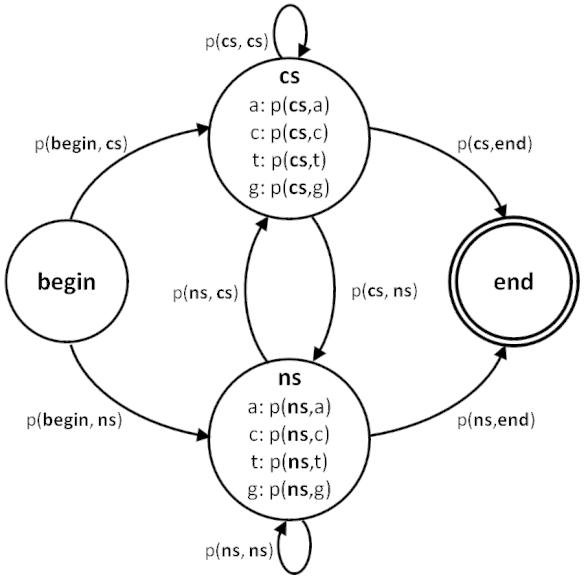
Following the general scheme for HMMs in PRISM, that was presented in section 5.2, it is equally straightforward to implement the model in PRISM (figure 6.1, Right).

Predictive analysis consists of finding a path through the HMM, that corresponds to which parts of the sequence represent coding regions, and which that do not. We can infer a most probable *path given the sequence and the model parameter, θ.*

The position specific annotation, A, produced this way forms a prediction of the location of the coding regions in the input DNA sequence, S, based on similarity between occurrence of nucleotides in the input sequence and the preference hypothesised by the model parameters (figure 6.1, Bottom).

Example 11 *As an example of a* deterministic *annotation model, consider again the FSA from figure 3.2, also shown in a simplified version in figure 6.2. It can be implemented as an HMM-based annotation program similarly to the one shown in figure 6.1 for annotating likely orf- and non orf-positions in an input sequence.. Recall that orf's are defined in terms of 3-nucleotide units called codons. They begin with a* start codon*, then follows a sequence of* coding codons *and terminates with the first* stop codon *in this sequence of codons.*

An annotation, a_{orf}, could be a codon-specific one, that assigns a discrete triplet-symbol, "...", "<<<", ">>>" or "---" to each non-coding codon, start codon, stop codon and orf codon, respectively (see figure 6.3). This annotation is necessarily specific to a specific reading frame and for each there is only one possible outcome of the corresponding annotation model, m_{orf}, that is therefore an instance of a deterministic *annotation model with an empty list on parents, by definition 38.*




```
1.  initial(begin).
2.  final(end).
3.  values(trans(begin),[cs,ns]).
4.  values(trans(cs),[cs,ns,end]).
5.  values(trans(ns),[cs,ns,end]).
6.  values(emit(cs),[a,c,t,g]).
7.  values(emit(ns),[a,c,t,g]).
8.  restore_sw("parameter_file").
9.  hmm(S,A):-
10.    initial(State0),
11.    msw(trans(State0),State1),
12.    hmm(State1,S,A).
13. hmm(State,[],[]):- final(State).
14. hmm(State,[Symbol|S],[Markl|A]):-
15.    \+ final(State),
16.    annotation_symbol(State,Mark),
17.    msw(emit(State),Symbol),
18.    msw(trans(State),Next),
19.    hmm(Next,S,A).
20. annotation_symbol(cs,c).
21. annotation_symbol(ns,n).
```

```
S: ataaactggcggaaggtatccgtaagtttgctattgaccaggaaaaactggaaaaaatga
A: nnnnnnnnnnnnnnnnnnnnnnnnnnncccccccccccccccccccccccccccccccc
```

Figure 6.1: *(Left)* A first order output HMM modelling a difference in nucleotide preferences between coding and non-coding states. Apart form the silent `begin` and `end` states, there are states `cs` for coding and `ns` for non-coding, that can each be responsible for any of the four nucleotide symbols but with assumed different probabilities. *(Right)* An implementation in PRISM that closely follows the general scheme for HMM's (presented in section 5.2) except for an extra argument, `A`, in the `hmm` predicates for collecting the sequence of emitting states in a path. The transition and emission probabilities are assumed to be contained in the file named "parameter_file". Together the PRISM program and its parameters constitute a *probabilistic annotation model*. *(Bottom)* For any complete set of transition and emission probabilities, the most probable state sequence, `A`, for a given sequence `S` of nucleotide symbols `a`, `c`, `t` or `g` can be identified as an example of a probabilistic annotation of `S`.

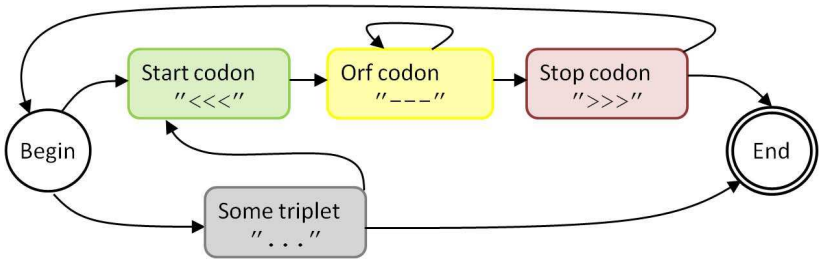


Figure 6.2: Simplified structure of the finite state automaton for DNA from figure 3.2. The annotations, "`...`", "`<<<`", "`---`" and "`>>>`", associated with the codon-classes in an ORF-annotation are also indicated.

```
seq: tttaactggcggaaggtatccgtaagtttgctattgaccaggaaaaactggaaaaaatga
orf: ........................<<<---<<<--------------------->>>
cns: 000000000000000000111111222222333333333333333333333222333
gen: 000000000000000000000000000000001111111111111111111111111
```

Figure 6.3: Three annotations, `orf`, `cns` and `gen` of a common DNA sequence `seq`.

Example 12 *Another example of a* deterministic *annotation model,* m_{cns}*, employs the analysis of an external tool that computes a* conservation *annotation.* Conservation *here describes a degree to which the codons of a DNA sequence are conserved across species. Mutation in the DNA, either due to environmental influences or to errors during DNA-replication from one generation of the organism to the next, deletes or inserts small nucleotide sequences or exchanges individual nucleotides. In general, codons of actual genes are better conserved (less mutated) between related organisms than codons of non-gene regions. A representation of conservation of the DNA can be established by aligning it against a selection of reference genomes using for example the* tBlastn program*, available from NCBI [49]. Informally, we can devise a codon-specific annotation,* a_{cns}*, that assigns a discrete triplet-symbol to each codon, for example "`000`", "`111`", "`222`" and "`333`" depending on the degree to which it is conserved across the reference genomes, i.e., conserved in 0-25%, 25-50%, 50-75% and 75-100% of the genomes in the reference set, figure 6.3.*

Example 13 *The two previous annotation models,* m_{orf} *and* m_{cns}*, annotated the input sequence* S *directly independently from any other annotations. Each of them can however be considered features that are possibly*

relevant for gene-finding and thus could themselves form parent *annotations to an annotation model, m_{gen}, producing an annotation, a_{gen}, that indicates possible genes in S, depending on annotations, a_{orf} and a_{cns}. A position-specific annotation could mark positions predicted as belonging to genes with* `1` *while other positions are marked* `0`*, shown in figure 6.3. Note how such a modularised scheme allows individual factors* orf-structure *and* conservation *to be modelled in separate specialised models, m_{orf} and m_{cns}, and leaves the overall reasoning with the corresponding signals to another specialised model, m_{gen}. A HMM-based PRISM-program implementing the logical part of m_{gen} is listed in figure 6.4.*

6.3 Data-partioning – *chunking*

Keeping individual annotation models as simple as possible clearly alleviates the complexity of the overall analysis. Some features may however still be too complex to annotate efficiently. With the vast size of data-instances in DNA-annotation problems, many analytical tasks are very vulnerable to possible combinatorial explosion of the solution space. For example, parsing of probabilistic context-free grammars is of polynomial complexity in terms of sequence length n, i.e., $O(n^3)$. With n in the magnitude of several millions, cubic analysis is impractical. The problem of course increases with analysis complexity, and several relevant analyses are significantly more complex than the relatively simple standard parsing-algorithm. However, the combinatorial explosion can in general be reduced to a series of smaller ones, by dividing long sequences into small enough partitions, that I refer to as *chunks*.

There is of course a myriad of ways to partition a sequence, not all of which are equally suitable. To be useful, a partition-strategy must fulfil the following three requirements:

- it must divide the input data into chunks that are small enough for efficient analysis,
- it must prevent that what we wish to analyse is destroyed or lost as a consequence of the partitioning
- the analysis needed for partitioning should not be more complex than the overall analysis itself.

What chunk-size is suitable for efficient analysis obviously depends on the computational complexity of the analysis, i.e., how fast time- and memory-consumptions grows in terms of data-size. As already, mentioned computational complexity of algorithms are usually distinguished as either

```
initial(begin).
final(end).

values(trans(begin),[gene_pos,nongene_pos]).
values(trans(gene_pos),[gene_pos,nongene_pos,end]).
values(trans(nongene_pos),[gene_pos,nongene_pos,end]).
values(emit(gene_pos),[(_O,_C,_S)]). *
values(emit(nongene_pos),[(_O,_C,_S)]). *

prog_gen(Seq,A_gen, A_orf, A_cns):-
 initial(State0),
 msw(trans(State0),State1),
 prog_gen(FirstState,A_gen, A_orf, A_cns).

prog_gen(ThisState,[],[],[],[]):- final(ThisState).
prog_gen(ThisState,[S|Seq],[G|A_gen],[O|A_orf],[C|A_cns]):-
 \+ final(ThisState),
 msw(emit(ThisState),(O,C,S)),
 annotation_symbol(ThisState,G),
 msw(trans(ThisState),NextState),
 prog_gen(NextState,Seq,A_gen,A_orf,A_cns).

annotation_symbol(gene_pos,1).
annotation_symbol(nongene_pos,0).
```

Figure 6.4: One implementation of the logical part of the annotation model m_{gen} in PRISM. Notice that the emission variables for both gene-positions and nongene-positions, (marked with *) range over all possible combinations, `(O,C,S)`, of position specific orf-, cns-, and sequence-symbols, each respectively ranging over {`.,<,-,>`}, {`0,1,2,3`}, and {`a,c,t,g`}. In a generative execution of this program, each iteration of the `prog_gen/5` predicate, emits one such combination from the current state, records the annotation symbol, `G`, associated with that state, `0` or `1`, in the `Ag`-list, decides on a transition state and calls itself recursively. Recursion stops once a final state, here `end`, has been reached. For explanation search, the most likely annotation, `Ag`, given specific `Seq`, `Ao` and `Ac` can be computed by (an adapted version of) the Viterbi utility of PRISM.

polynomial or exponential, where polynomial means that the time consumption as a function of data-size grows polynomially. Traditionally polynomial algorithms are considered good in contrast to exponential algorithms that should be avoided at all times. In domains like the present, with extreme data-sizes, only the simplest polynomial algorithms are really practical.

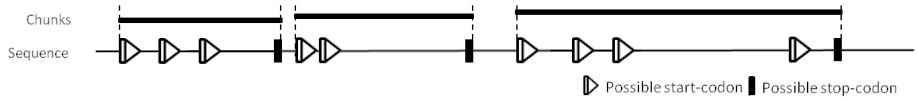
The object of the analysis of course constrains how to partition data in order not to destroy that which we wish to analyse. It is a general and challenging problem that is shared among others with natural language processing. Its solution requires sufficient knowledge of the domain of application in order to motivate the partitioning properly.

Of course, if the analysis necessary to perform the partitioning is too complex, i.e., as complex as the original analysis, the point of partitioning is lost al together.

Orf-based chunking for DNA

For gene-finding in DNA several strategies based on orf's (open reading frames) can be formulated, that keep genes intact after chunking. Recall that all genes occupies an orf consisting of a start-codon, a series of orf-codons and terminated by a stop-codon, where possible start-codons double as possible orf-codons but any stop-codon terminates the orf.

Only longest possible orf's: We may for example define chunks as longest possible orf's, i.e., starting with the earliest possible start-codon for each possible stop-codon:



The gene-finding task is then reduced to deciding whether each chunk contains a gene and, if so, which of the alternative start-codons is the actual one. While features that occur strictly within a given coding region may be analysed efficiently this way, it is clear that features concerning the transitions between non-coding and coding regions in the DNA cannot be analysed consistently due to lack of context, particularly so for the first start-codon in each chunk.

From stop to stop: We may instead consider chunks going from the stop-codon of one orf to the stop-codon of the next one:

This strategy has the advantage of covering the entire input sequence and thereby allow analysis of intergenic features in the same reading frame of the DNA. It also provides the "intergenic" pretext for each possible start-codon and thus allows for a more consistent analysis of the actual starts of potential genes. As a drawback of this strategy (as well as the previous one), genomic context around the stop-codons is consistently missing, preventing analysis of features concerning the transition from coding to non-coding regions in the DNA.

Orf's with context: The third strategy, shown below, defines chunks in terms of longest possible orf's framed by fixed amount of genomic context – both before and after:

This strategy allows analysis of transition between coding and non-coding regions but prevents intergenic DNA from consistent analysis. However, given that we are looking for genes that must involve an orf, it is perhaps reasonable to forego unnecessary analysis of intergenic DNA.

Other orf-based alternatives exist, that are not illustrated here, including of course analysing each orf individually – rather than *en bloc*, as indicated here with several orf's per chunk, which may be useful for some analyses.

Alternatives to orf-based chunking are of course also possible, and indeed necessary when analysis concerns for example cross-frame relationships between features in different reading frames. They could for example involve overlapping chunks of fixed length or be based on a larger unit of DNA like operons or even entire genomes for some purposes.

The choice of chunking strategy clearly places constraints on what features can be analysed consistently and different analyses may call for different chunking strategies. Different chunking strategies, however, are easily implemented as deterministic annotation models, marking the input sequence with appropriate chunk-starts and -stops, that subsequent analyses may depend on as parent annotations. As illustrated in chapter 8, it is possible to design a chunking mechanism that, in addition to partitioning input, also classifies individual partitions and thereby decides what subsequent analysis they are eventually submitted to. In general, chunking

models do not offer individual interpretation of the input, and can be seen simply as a pre-process for a more complex analysis.

6.4 Organising annotations in Bayesian networks

Given a collection of specialised annotation models that each regard factors that are possibly relevant to the overall analysis, the remaining challenge is to integrate the information represented by their respective annotations into a compound predictive analysis. The general idea pursued here, is to evaluate one model at a time and use their results as conditions for subsequent analyses. This is very similar to *forward* analysis in *Bayesian networks* of random variables, except that we are considering networks of entire probabilistic models rather than of single random variables.

Bayesian networks

Bayesian networks, also called *belief networks*, are general probabilistic models that represent the conditional dependencies (definition 22) among its random variables and also, very compactly, their joint probability distribution (definition 20). Bayesian networks can be defined as directed acyclic graphs, as follows:

Definition 39 *A* Bayesian network *(BN) is a directed acyclic graph* $g(V, E)$*, of nodes,* V*, and directed edges,* E*, where :*

- *nodes,* $v \in V$*, represent random variables,*
- *an edge,* $(s, t) \in E$*, indicates that a node,* t*, is* directly dependent *on a node* s*, and* s *is called a* parent *of* t*; the notation* $Parents(t)$ *refers to the set of parent nodes of* t*.*
- *Each node* s *has an associated* conditional probability distribution*,* CPD*,* $P(s|Parents(s))$*.*

The intuition is that the distribution represents a probabilistic quantification of the respective degrees of dependence on each of the nodes parents. We will write $Parents_i$ instead of $Parents(m_i)$. In these terms, a *partial ordering* of nodes, $\prec$, can be defined that satisfies the constraint that parents must *precede* their children, i.e.,

$$\forall x_i, x_j : x_i \in Parents_j \rightarrow x_i \prec x_j$$

The general independence assumption for Bayesian Networks is that given the values of its parents, any variable is independent from the rest of its predecessors. For any atomic event e of a node we may thus rewrite the corresponding entry of the full joint probability distribution in terms of conditional probabilities, i.e.:

$$p(e) = p(x_1, \ldots, x_n) = p(x_n|x_{n-1}, \ldots x_1).$$

By definition 4.1 – that states that if ϕ and ψ are independent events then $p(\phi|\psi) = p(\phi)$ – we may for any node x_i in a BN disregard those values that x_i does not directly depend upon, leaving the smaller set of conditionals $Parents_i$, i.e.:

$$p(x_i|x_1, \ldots, x_n) = p(x_i|Parents_i)$$

Example 14 *Figure 6.5 shows a classic example of a Bayesian network adapted here from [61]. All nodes in this network represent binary random variables, ranging over values true and false. The conditional probability tables are shown in a compact form containing only the probability of each being true, where for example* $p(A|B, E)$ *abreviates* $p(A = true|B = true, E = true)$*. In the example, a burglary alarm was installed, that quite accurately detects burglary but also sometimes goes off due to small earthquakes. A couple of neighbours, John and Mary, will call the owner of the house if they hear the alarm, and they do so with varying reliability. Both burglaries and earthquakes happen once in a while, but with slightly different frequencies, as represented by their marginal probabilities,* 0.001 *and* 0.002 *respectively. The probabilities that the alarm goes of and that John or Mary calls on different conditions are reflected in the respective conditional probability distributions.*

Being themselves probabilistic models, by definition 20, Bayesian networks can of course also be represented in PRISM. A PRISM program for the network from example 6.5 is listed in figure 6.6.

Bayesian annotation networks

In our application, the domains of the nodes in the network each range over the possible annotations from an annotation model. Usually, the CPDs associated with nodes in a BN are given in the form of tables, as also shown in figure 6.5, but since the random variables in our case range over huge sets of alternative annotations, that is quite impossible here and instead we exploit that a probabilistic annotation model $m = \langle prog(s, a, parents), \theta \rangle$ is a complete representation of the associated CPD, as by definition 37.

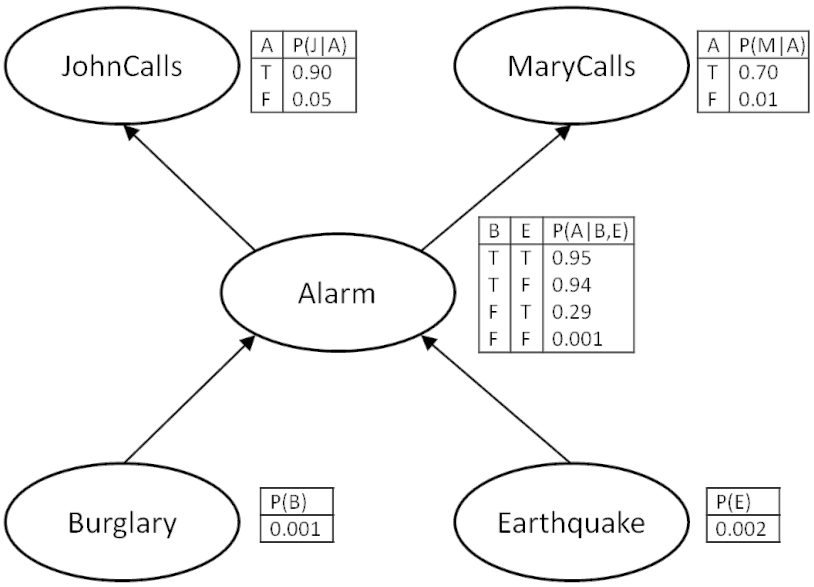


Figure 6.5: A classic example of a Bayesian network adapted from [61]. All nodes in this network represent boolean random variables, and the associated conditional probability distributions are shown in the tables next to each node.

```
values(burglary,[true,false]).
values(earthquake,[true,false]).
values(alarm(_Burglary,_Earthquake),[true,false]).
values(johnCalls(_Alarm),[true,false]).
values(maryCalls(_Alarm),[true,false]).

bn(B,E,A,J,M):-
 msw(burglary,B),
 msw(earthquake,E),
 msw(alarm(B,E),A),
 msw(johnCalls(A),J),
 msw(maryCalls(A),M).
```

Figure 6.6: A PRISM program that implements the classic Bayesian network from figure 6.5.

Definition 40 *A* Bayesian annotation network *(BAN) is a set of probabilistic annotation models,*

$$\{m_i \mid i = 1, \ldots, n\},$$

with

$$m_i = \langle prog_i(s, a_i, parents_i), \theta_i \rangle,$$

indexed according to the order, $\prec$, of precedence, i.e.,

$$parents_i \subseteq \{a_1, \ldots, a_{i-1}\}.$$

The model, m_n, is a designated top model*, and it is assumed that the parent relationship induces a path from any other m_i to m_n.*

As a shorthand, I will write Θ to mean the set of all parameters θ_i of all models m_i in a given BAN.

Given this definition it should be clear that a BAN in itself is not a BN, but also that it induces a BN in the following way (figure 6.7):

- There are nodes for the sequence, s, and every model, m_i, in the BAN, labelled a_i, $i = 1, \ldots, n$ and s.
- Whenever $a_i \in parents_j$ in the BAN, there is an edge from a_i to a_j, and there is an edge from s to any a_i.
- The CPD, $P(a_i \mid s, parents_i, \theta_i)$, associated with a_i is given by the model $m_i = prog_i(s, a_i, parents_i), \theta_i \rangle)$.

For ease of terminology, I refer to a suitable set of annotation programs as a BAN, even though they by the present definition do not constitute a BAN until their respective parameters, θ_i, are established and the programs become probabilistic models. Also, when doing predictive inference below, the sequence, s, is always fixed, so we can leave it out, assuming instead a particular BN for each individual s.

Example 15 *A Prism program that implements a BAN over the annotation-models m_{orf}, m_{cns} and m_{gen} from examples 11, 12 and 13 would simply follow the Bayesian structure imposed by their conditional dependencies but invoke annotation-programs rather than sample from random variable distributions:*

```
ban(Seq,Agen,Aorf,Acns):-
    prog_orf(Seq,Aorf),
    prog_cns(Seq,Acns),
    prog_gen(Seq,Aorf,Acns,Agen).
```

I will use this as a running example for the the remainder of this chapter, to illustrate the basic methodology of organising annotation-models in a BAN.

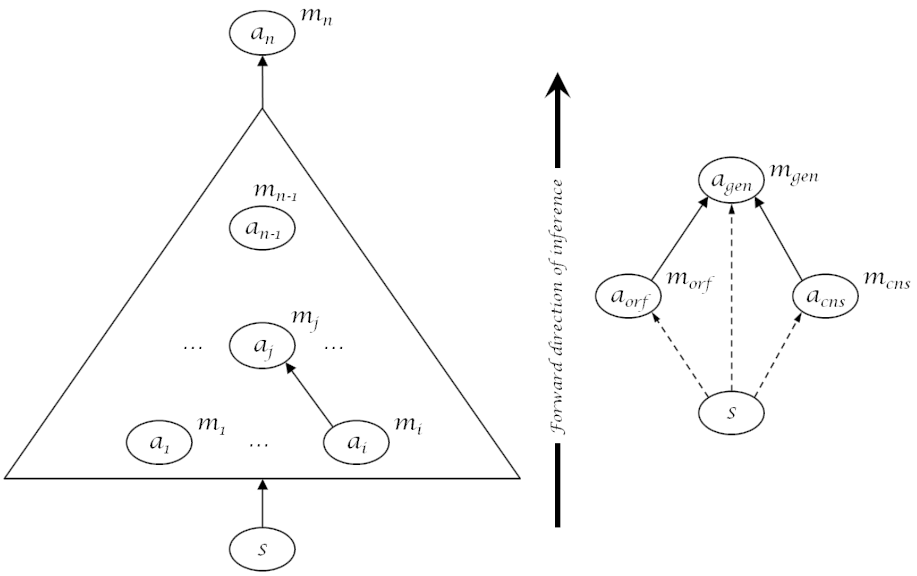


Figure 6.7: *(Left)* Abstract sketch of the Bayesian network induced by an arbitrary BAN for some sequence s. Each node, a_i, in the topology represents *the most likely* annotation of s, given the annotation model m_i and its parent-annotations. All local inference in our application goes in the *forward direction*, i.e., from parent-node(s) to their respective child-node(s). As an approximation of the updated probability distribution over all possible annotations from m_i, we take as the result of inference *a single most likely annotation*, a_i, depending on $Parents_i$. As the sketch indicates, the sequence, s, is in principle a parent to all nodes in the network, but because, in every case of application, we will be considering networks for one specific sequence, we may simply assume the relation to s as constant and implicit in any specific BAN. *(Right)* A concrete example BAN of the annotation models m_{orf}, m_{cns}, and m_{gen} from examples 11, 12, and 13, respectively. The dotted edges from s indicate that because all models in the network annotate the same s, we may ignore it as a separate node.

6.5 Inference

Three distinct inferential tasks are of interest to the present framework:

- *sampling*, i.e., constructing sample observations that fit with the distribution of a parametrised model.
- *predictive inference*, i.e., finding the most likely annotation from a model of a sequence, possibly depending on one or more parent annotations.
- *parameter estimation*, i.e., finding model parameter that maximises the likelihood of a set of suitable training data.

Sampling – inferring representative examples

A drawback of BAN's is that it is not possible in general to perform sampling using the builtin predicates, because here dependencies are represented via the arguments of the individual annotation models and not necessarily reflected in the Multi-valued switch declarations of the those models.

Recall, that sampling with a probabilistic model literally refers to picking an arbitrary atomic event e from the joint distribution of the model, with probability $p(e)$, section 4.4. Sampling atomic events from the distribution defined by a traditional Bayesian network requires that the dependencies between the random variables are respected. For Bayesian networks implemented in PRISM, this simply corresponds to the so-called *sampling execution* of the program, corresponding to the procedural meaning of the logic-programing. I.e., for the example BN-program in figure 6.6, calling the goal `bn(B,E,A,J,M)` corresponds to sampling an atomic event of the model in the following way:

- First values, `B` and `E`, for burglary and earthquake are sampled, by the two first calls to `msw/2`.
- Then a value, `A`, for alarm is sampled *conditioned on* `B` and `E`, by the call `msw(alarm(B,E),A)`.
- Finally values `J` and `M` are sampled for the events that John and Mary calls, *conditioned on* `A`.

Note that the dependencies of `A` on `B` and `E`, and of `johnCalls` and `maryCalls` on `A` are represented as distinguishing arguments to the respective switch-names `alarm/2`, `johnCalls/1`, `maryCalls/1)`, such that each distinct instance of these defines a distinct and independent switch, (section 5.1). This is the only way to directly express conditional dependencies in the PRISM system.

Following the same scheme with the example BAN, example 15, we

would execute the programs associated with the models in order of precedence in the network, i.e.:

`prog_orf(Seq,Aorf)`,
`prog_cns(Seq,Acns)`, and
`prog_gen(Seq,Agen,Aorf,Acns).`

and require that the instances of the respective variables unify between executions. Because of the inherent independence of random variables in PRISM, we would however have distinct instances of all the random variables, i.e., three distinct instances of `Seq`, and two of both `Aorf` and `Acns`, causing unification to fail in general.

Thus, when we require sample observations, for example for evaluation purposes, we are in general required to formulate a separate sampling-version of the network in question, that expresses conditional dependencies directly in the switch-declarations and manages the involved parameters accordingly. In chapter 8 we describe an experiment, where a BAN-approximation of a complex annotation model is evaluated by comparison to annotations sampled with the complex *canonical* model.

Predictive inference

Traditional probabilistic inference concerns establishing the *updated* or *posterior* probability distribution for a *query variable*, Q, given knowledge about the actual outcome of an *evidence variable*, E. In Bayesian networks, four categories of inference are typically distinguished, depending on the dependency relation between Q and E:

- *Forward inference* – follows the direction of the edges from *parent* to *child*, i.e., $E \in parents_Q$; Here for example observing, as evidence, a burglary and estimating the chance, that John will call. Forward inference is also called *predictive inference*.
- *Backward inference* – from *child* to *parent*, i.e., $Q \in parents_E$; here for example estimating the probability of a burglary given the evidence that John and/or Mary called, also called *diagnostic inference*.
- *Intercausal inference* – between conditions of a common dependent event; here for example estimating the risk that a burglary will occur during or after an earthquake.
- *Mixed inference*; any other kind of inference, perhaps combining two or more of the above.

In our case, we are interested only in the forward, predictive inference, that refers to the process of identifying a best proposal for an overall output annotation for a given input sequence, as indicated in figure 6.7. In

Bayesian networks, the usual result expected from a predictive inference, is the complete updated CPD for the query-variable given the observed evidence, i.e., by exhaustive application of Bayes conditioning (section 4.1):

$$\forall q_i \; p(q_i|E) = \frac{p(q_i, E)}{p(E)}$$

Translating to our setting, this ideally means iterating over all possible annotations from all annotations models in the network and selecting the most probable annotation, $ideal_n$, from the resulting distribution. As shown already, iteration over the outcome spaces of annotation models for DNA is not practical. We are accepting instead the *most likely* top annotation, $approx_n$, given the *most likely* parent-annotations, $approx_1 \ldots approx_{n-1}$, as an approximate predictive result.

Below, is first a precise, declarative characterization of the ideal but impractical top output annotation, $ideal_n$ and then the approximative calculation method which reduces computational complexity drastically.

Ideal but impractical method: We assume a BAN $\{m_i \,|\, i = 1, \ldots, n\}$ with $m_i = \langle prog_i(s, a_i, parents_i), \theta_i \rangle$ and a fixed sequence s^0 to be analysed. We use Θ to refer to the set of all parameters in the BAN, $\{\theta_1, \ldots, \theta_n\}$. Considering the BAN as one coherent model, we can describe the best solution as follows.

$$ideal_n(s^0, \Theta) =_{\text{def}} \operatorname*{argmax}_{a_n} P(a_n \,|\, s^0, \Theta)$$

where the term inside the argmax can be unfolded as follows.

$$\begin{aligned} P(a_n \,|\, s^0, \Theta) &= \sum_{\langle a_1, \ldots, a_{n-1} \rangle} P(a_1, \ldots, a_n \,|\, s^0, \Theta) \\ &= \sum_{\langle a_1, \ldots, a_{n-1} \rangle} \prod_{i=1}^{n} P(a_i \,|\, s^0, parents_i, \theta_i) \end{aligned} \tag{6.1}$$

It is practically impossible to iterate over all instances of any one annotation, a_i, let alone all instances of the tuple $\langle a_1, \ldots, a_{n-1} \rangle$, and we are not aware of any reasonable way to reduce this formula (although we do not have a formal proof that it does not exist).

Example 16 *For our example BAN (example 15) this corresponds to one overall call to PRISMs* `viterbig/1` *predicate, i.e.:*

```
ideal(Seq,Agen):-
   viterbig(ban(Seq,Agen,_Aorf,_Acns)).
```

where A_{gen} is calculated in one go from Seq. Note how we do not care about the specific values of the anonymous (beginning with the underscore-character) variables `_Aorf` *and* `_Acns`*. The Viterbi algorithm is thus allowed full unrestricted freedom to find the most likely of the vast number of possible combinations of annotations A_{gen}, A_{orf}, and A_{cns}.*

Approximative method: We propose instead an approximative algorithm that fixes one particular best annotation $a_i = approx_i(s_0, \Theta)$ for each sub-model and applies it subsequently in the prediction of those a_j with $a_i \in parents(a_j)$, i.e.:

$$approx_i(s^0, \Theta) = \underset{a_i}{\operatorname{argmax}}\, P(a_i \mid s^0, approx_{parents_i}(s^0, \Theta), \Theta),\ i = 1, \ldots, n \quad (6.2)$$

where $approx_{parents_i}(s, \Theta)$, for some sequence s, stands for the sequence of parent annotations $approx_j(s, \Theta)$ for all $a_j \in parents_i$.

Specifically, we take $approx_n(s^0, \Theta)$ as an approximated value for $ideal_n(s^0, \Theta)$. Notice, that because of the partial ordering of nodes, there is no circularity in this definition and $approx_n(\cdots)$ can be calculated in a single iteration calculating $approx_1(\cdots)$, $approx_2(\cdots)$, ... in that order. The "argmax" in (6.2) may be calculated proper application of the viterbi-predicates of PRISM. algorithms as we demonstrate below.

Example 17 *In terms of our example BAN (example 15) we first decide on most likely annotations a_{orf} and a_{cns}, and then use them as specific conditions for the top-annotation a_{gen} – getting in total three consecutive calls to* `viterbig/1`*:*

```
approx(Seq,Agen):-
   viterbig(orf_prog(Seq,Aorf)),
   viterbig(cns_prog(Seq,Acns)),
   viterbig(gen_prog(Seq,Agen,Aorf,Acns)).
```

Efficiency of approximate inference Measured in terms of sequence length, the complexity of approximate prediction with the entire BAN is constrained upwards by the complexity of its most complex sub-model. This

is a drastic reduction in complexity compared to the ideal algorithm, where the inferential complexity is constrained by the product of the complexities of all sub-models in the BAN. In practical applications of the methodology, we would expect the number of sub-models in a BAN to be a relatively small number (say, arbitrarily, < 10), but lengths of sequences and their annotations are expected to be huge.

Training – inferring parameters

In order to obtain the probabilistic parameters Θ for a BAN, we rely on existing training algorithms for supervised learning, for example the EM-algorithm as it is built into the PRISM system (explained in section 4.4 and 5.3). Such algorithms require a sufficiently large and representative collection of ground atoms for each sub-model, each representing a sequence with its correct annotation, which in our domain of application means annotations verified in the lab by the biologists.

To this end, we assume the availability of some state of the art training algorithm $T^{supervised}$, described as a function mapping a particular program together with its training data into a parameter.

For doing supervised training of any sub-model in a BAN, we need in principle ground data that exemplifies the relation between sequence, parent annotations, and output annotation. We define, thus, a *conditional training data set* for a model m_i as a set

$$CTD_i = \{prog_i(s_i^j, a_i^j, parents_i^j) \mid j = 1, \ldots\}.$$

It is called "conditional" since it includes parent annotations $parents_i^j$ for each output annotation a_i^j.

Iterative training of sub-models In practice, however, we cannot expect such conditional training sets to be available as this assumes that the signals represented by the different sub-models have been analysed consistently for the same set of sequences. In other words, we can only assume that the following sorts of training data are available in a more traditional format without explicit parent annotations.

$$TD_i = \{\langle s_i^j, a_i^j \rangle \mid j = 1, \ldots\}$$

However, if we train the different models one by one in the order m_1, m_2, ..., we can use the already trained models to supply parent annotations. We can thus specify an iterative BAN training algorithm as follows.

$$\theta_i = T^{supervised}(m_i, CTD_i)$$

where

$$CTD_i = \{prog(s_i^j, a_i^j, approx_{parents_i^j}(s_i^j, \{\theta_1, \ldots, \theta_{i-1}\})) \mid \langle s_i^j, a_i^j \rangle \in TD_i\}$$

There is no circularity in these equations which may be evaluated in one sweep θ_0, θ_1, ...

Example 18 *Consider again the example BAN (example 15) and assume that we have or can easily establish sufficient training data for the relationships between DNA and orf-annotation, DNA and conservation, and DNA and typical genes. I.e., we have available :*

$$TD_{orf} = \{\langle DNA_1, a_{orf} \rangle\}$$
$$TD_{cns} = \{\langle DNA_2, a_{cns} \rangle\}$$
$$TD_{gen} = \{\langle DNA_3, a_{gen} \rangle\}$$

where DNA_1, DNA_2, DNA_3 indicate sequences from possibly distinct sets of DNA-sequences.

To train models m_{orf} and m_{cns}, that have no parents in the network, we may simply construct conditional training data directly from TD_{orf} and TD_{cns} respectively, and perform supervised training.

To train m_{gen} we ideally require conditional training data

$$CTD_{gen} = \{prog_{gen}(s, a_{gen}, a_{orf}, a_{gen}\}$$

but we have only $TD_{gen} = \{\langle DNA_3, a_{gen} \rangle\}$. We may, however, apply the now properly parametrised models m_{orf} and m_{cns} to predict approximate annotations of sequences in DNA_3 to establish the necessary conditional training data for supervised training of m_{gen}, i.e.,

$$CTD_{gen} = \{prog_{gen}(DNA_3, a_{gen}, approx_{orf}, approx_{gen}\}.$$

This strategy can be adapted to handle cases where training data TD_i are unavailable for some non-top model m_i, i.e., $i < n$. Here we may use unsupervised training, or even set the parameters manually, and still hope for good results. It is not essential that model m_i is a faithful mirror of some physically measurable signal (call this m_i^{true}): the necessary property is whether a_i represents *some* annotation that can help the models m_j of which m_i is a parent to discriminate the details of the sequence under consideration.

6.6 Evaluating BAN-topologies

I have described the basic framework of how to construct annotation models and integrate them as constituent nodes in a BAN that reflects their interdependencies. An important question remains: how to evaluate an experimental BAN? There are two aspects of evaluation that both need addressing.

Firstly, from a methodological point of view we need to be able evaluate BANs as a framework for modular analysis. In particular we need methods for evaluation of the approximative analysis and also for evaluating the quality of the topology of a given BAN and the relative impact of constituent models on the overall analysis.

Secondly, from a more experimental point of view, we need to be able to evaluate the predictive power of the BAN. In the actual experimental setting where the BAN framework is designed to be used, it is of course also essential to be able to evaluate the quality of the predictions of a particular BAN. If sufficient authoritative test-data is available that can be used as golden standard, it is - also here - most convenient to apply accuracy measures that compare predictions to the that standard, e.g., specificity and sensitivity. In many cases, such data is not likely to be available, and suitable measures from the realm of statistical testing must be applied. There are many methods that can be applied for statistical evaluation of results with regard to format and domain of data they are applied to. To illustrate how the predictions of alternative topologies may be presented for comparison and inspected to motivate further experiments with novel or putative signals and models, I include in our example BAN (example 15) an evaluation of predictions based on sensitivity and specificity

In the present context, however, I am most concerned with the evaluation of the approximative BAN-analysis in general and the relative quality of alternative BAN-topologies and less so with the evaluation of the predictive power of specific BANs. Actual application to relevant biological problems and thorough statistical analysis of the predictive results remains future work in the LoSt project.

Evaluating the approximation

A BAN is an organisation of a set of possible interdependent annotation models as nodes in a network where the directed edges reflect node dependencies. Each constituent model produces an annotation of a common data-sequence depending in part on the annotations of other constituents. The method for integrating these annotations is similar to forward reasoning in

a Bayesian Network. In traditional Bayesian Networks, inference concerns, for each node, its entire CPD given the entire CPD's of all its parent-nodes. Because of the sheer size of data-instances in DNA-annotation exact inference is not practical and we employ an approximative method for inference that considers only the most likely outcome of any node given the most likely outcomes of its parents, rather than the entire distributions. As a scheme of approximation it seems reasonable enough to assume that a most likely overall result of a complex analysis involves only most likely constitutive analyses. In fact the much celebrated Viterbi-algorithm applies a similar scheme fro exact calculation of the most likely sequence of hidden states in a HMM, explaining a particular sequence of observable outcomes of that HMM. Realising that HMM's are indeed very simple BAN's, conditioning on most likely constitutive sub-analyses in the context of HMMs represents an optimization rather than an approximation. In general, however, we cannot make the same guarantee and the how evaluate the approximative BAN-inference in general represents an important research problem.

The standard way of evaluating an approximation is to compare to the exact analysis. This is however not possible in our domain of application, because exact inference is simply not practical in this domain. In general the choice of method must depend on the availability of authoritative test-data that can be used as a golden standard of analysis.

In the presence of authoritative test data

In those cases where sufficient amounts of authoritative annotated data are available for use as golden standard, it is natural to evaluate BAN's in terms of standard accuracy measures, sensitivity and specificity. Chapter 7 documents an experiment where evaluation is done this way.

Example 19 *In our example BAN (example 15) we would partition* TD_{orf}*,* TD_{cns}*, and* TD_{gen} *into training- and test-partition. After training on the training-partition alone we would then evaluate accuracy-measures against both training- and test-partitions. A common problem in machine-learning, known as* over-fitting*, concerns cases where a parametrised model predicts correctly those data that was used for establishing the model-parameters, but performs less good good on general data from the same domain. Dividing available authoritative annotated data in training- and test-data allows us to evaluate the performance on both sets, i.e., the one that it was used for training and one that it has not been exposed to before, to evaluate how well the model is adapted to data from the domain in general.*

In the absence of test data

In cases, where authoritative annotated data is lacking, it may be possible to compare probability distributions of annotations assigned by the respective models to a common set of data-sequences or even compare the annotations themselves directly. Because the exact inference is still not possible, we employ a scheme that we call *evaluation by sampling*. In this approach of evaluation, a large amount of observations along with all annotations, is first sampled with a properly parametrised *sampling model*. These samples are now used as test-data. In this case, where the golden standard consists entirely of artificial data, it becomes somewhat meaning-less to refer to the usual accuracy measures. In chapter 8 I explore, as a measure for the quality of the approximative analysis, the difference between the probabilities of canonically sampled annotations, A, and their approximated counterparts, A', relative to common data sequences S. I cross-validate by also comparing the assigned annotations position for position and compute what could be called their *Hamming-similarity*. This approach follows the same intuition of the more general *Kullbach-Leibler divergence* that quantifies the difference between an approximative and a canonic distribution:

Definition 41 (KL-divergence)

$$KL(P(A'|S), P(A|S)) = \sum_s P(a'|s) log \frac{P(a'|s)}{P(a|s)}$$

The KL-divergence is however not a true distance measure since it does not hold in general that $KL(P(A|E), P(B|E)) = KL(P(B|E), P(A|E))$. There are also other alternatives available for traditional Bayesian Networks, and interesting future work involves investigating which of these that may be adapted to apply to BAN's also.

The sampling model in this example is a separate PRISM-model where all dependencies are modelled in the arguments of the Multi-valued switches (rather than in the arguments of the model programs $prog_i$, and the approach, at least as implemented in the present work, seems rather clumsy and cumbersome for non-trivial applications.

Evaluating topology and relative impact of constituents

As another general approach, topologies may be evaluated in information theoretical terms [69]. The individual model parameters may be inspected to asses how they influence each other. In particular we may calculate the

entropy, $\mathcal{H}(Ban)$, of some BAN, and the *conditional entropy*, $\mathcal{H}(Ban|M_i)$, of that BAN given a constituent model M_i The difference between these two entropies is known as the mutual information $\mathcal{I}(Ban; M_i)$ and may be seen as an indication of the possible impact of M_i on Ban. Similarly, we may for any parametrised constituent model m_i calculate its actual entropy relative to the maximal entropy possible for that model, to calculate a ratio $1 - \frac{\mathcal{H}(m_i)}{\mathcal{H}^{max}(m_i)}$. This ratio that Shannon called the *redundancy* of m_i can be used to qualify the signal-value and thus the potential suitability of the model for inclusion in a BAN. This approach represent future work but is sketched in more detail in chapter 9 and appendix A.

Chapter 7

Example 1: Comparing Gene-finder Topologies

7.1 Introduction

In this chapter I include experiments published in [18] as an example of the proposed methodology for constructing and experimenting with complex sequence annotation models.

7.2 Annotation task

The experiments concern BAN's that represent genefinders for prokaryotic DNA sequences. DNA sequences are first divided into chunks according to the stop-to-stop partitioning strategy introduced in the previous chapter, i.e. the DNA is considered in six different reading frames each of which is partitioned after the stop-codon of each complete orf. Our annotation models are designed to annotate chunks, where the annotation task is to find if the chunk contains a gene and, if so, where that gene starts.

7.3 Constituent models

We design models for different signals, *codon preference*, *gene length*, and *conservation* that are all expected to have influence on whether an orf is treated as a gene or not. All our probabilistic models are output HMMs with a gene-state and a non-gene state, which can emit symbols of the annotations of the parent nodes. The transitions between the states reflect the described orf pattern. The resulting annotation from such models is a

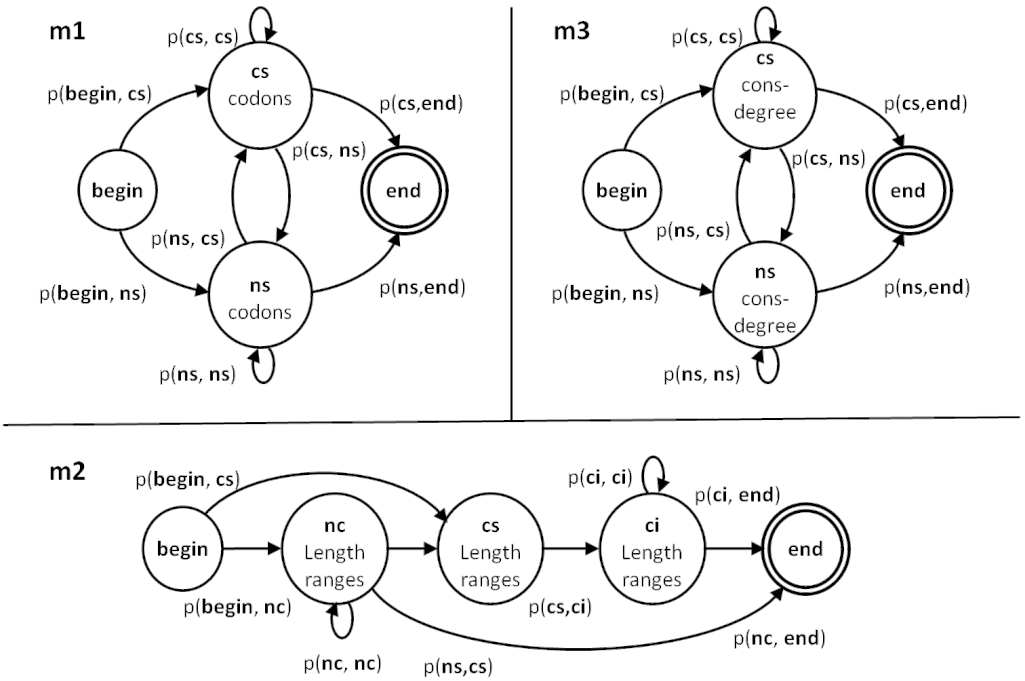


Figure 7.1: Graphical represetations of the three basic annotation models that we experimented with. **m1**, **m2**, and **m3**.

sequence that for each position in the original sequence contains a `1` if the position is predicted as part of a gene and a `0` if it is not. In the following I describe each of these in a little more detail, and in figure 7.1 I illustrate them graphically.

Codon preference, model m1

This annotation model, sketched in figure 7.1(*top left*), reflects preferential codon usage in the gene and non-gene state. Because of the way codons are transcribed and translated to proteins, the codons in actual protein-coding genes display a specific codon distribution that is different from the distribution in non-coding regions of the DNA. This can be modelled as a HMM, analogously to nucleotide-preferences in figure 6.1 on page 75, except here states each emit one of 64 possible codons, rather than one of four nucleotides. In the PRISM code shown below each codon, $\langle N1, N2, N3\rangle$, from the coding state is annotated with the triplet-symbol $\langle 1, 1, 1\rangle$ and codons from the non-coding state with $\langle 0, 0, 0\rangle$ (the angles are not part of the annotation, but included here for readability). Training- and test-data for the model, can be constructed as described in section 6.5 from existing

gene-annotations for example in the *RefSeq*-database, [59], i.e., individual training- and test-goals will look this: `codpref(Seq,RefSec)`, where `Seq` is a chunk and `RefSeq` is the gene-annotation of that chunk according to the RefSeq-database.

```
% Initialize
initial(begin).
final(end).

% Transitions from Hiddens states
values(trans(begin),[ns,cs]).
values(trans(cs),[cs,end]).
values(trans(ns),[cs,ns,end]).

% Emissions from hidden states
values(emit(State),Codons_List) :-
        member(State,[ns,cs]),
        get_list_codons(Codons_List).

get_list_codons(L) :-
        L = [ [a,a,a],[a,a,t],[a,a,c],[a,a,g],
              [a,t,a],[a,t,t],[a,t,c],[a,t,g],
              [a,c,a],[a,c,t],[a,c,c],[a,c,g],
              [a,g,a],[a,g,t],[a,g,c],[a,g,g],
              [t,a,a],[t,a,t],[t,a,c],[t,a,g],
              [t,t,a],[t,t,t],[t,t,c],[t,t,g],
              [t,c,a],[t,c,t],[t,c,c],[t,c,g],
              [t,g,a],[t,g,t],[t,g,c],[t,g,g],
              [c,a,a],[c,a,t],[c,a,c],[c,a,g],
              [c,t,a],[c,t,t],[c,t,c],[c,t,g],
              [c,c,a],[c,c,t],[c,c,c],[c,c,g],
              [c,g,a],[c,g,t],[c,g,c],[c,g,g],
              [g,a,a],[g,a,t],[g,a,c],[g,a,g],
              [g,t,a],[g,t,t],[g,t,c],[g,t,g],
              [g,c,a],[g,c,t],[g,c,c],[g,c,g],
              [g,g,a],[g,g,t],[g,g,c],[g,g,g]
            ].

% HMM-Parser (Classic implementation)
codpref(Seq,Annotation) :-
                                initial(State0),
                                msw(trans(State0),State1)
        msw(begin,State),
```

```
        codpref_rec(State1,Seq,Annotation).

  % end State
codpref_rec(State,[],[]):- final(State).

  % coding state
codpref_rec(State,[N1,N2,N3|Rest_Nuc],
                                    [1,1,1|Rest_Annot]):-
        State == c,
        !,
        msw(emit(State),[N1,N2,N3]),
        msw(trans(State),New_State),
        codpref_rec(New_State,Rest_Nuc,Rest_Annot).

  % non coding state
codpref_rec(State,[N1,N2,N3|Rest_Nuc],
                                    [0,0,0|Rest_Annot]):-
        State == n,
        !,
        msw(emit(State),[N1,N2,N3]),
        msw(trans(State),New_State),
        codpref_rec(New_State,Rest_Nuc,Rest_Annot).
```

Gene length, model m2

This annotation model, figure 7.1(*bottom*), considers the length of candidate orf's in a chunk as an indication of their coding potential. The rationale is that actual protein-coding genes in DNA require a certain minimum of codons to translate into a non-trivial protein, and also that the vast majority of actual genes are not longer than a some soft maximum length. Each chunk is first reduced to the list of positions of potential start-codons. Each of these potential start-codons can be classified as belonging to one of three classes, corresponding to three states in the HMM::

- the `nc`-state, corresponding to codons prior to the actual start-codon of the gene,
- the `cs`-state, corresponding to the actual start-codon in a protein-coding gene
- the `ci`-state, corresponding to internal orf-codons (other than potential start-codons).

Each of these states emits a symbol corresponding to the distance to the upstream stop-codon, as returned by the `get_range/2`-predicate. Rather than considering all possible lengths in isolation, the model regards a set

of pre-specified ranges of lengths, as returned by the `length_ranges/1`-predicate. The `cs`-state follow one emission-distribution as reflected in the `gene_length_range`-switch, whereas the other states, `nc` and `ci`, share the emission distribution represented by the `nongene_length_range`-switch. Also here, training- and test-data can be constructed from existing databases of curated genome annotations. Below is the core HMM-structure for the length model, the logic for predicates `get_range/2` and `length_ranges/1` have been left out for brevity.

```
% Initialize
initial(begin).
final(end).

% Transitions from hidden states
values(trans(begin),[nc]).
values(trans(nc),[nc,cs,end]).
values(trans(cs),[ci]).
values(trans(ci),[ci,end]).

% Emissions from hidden states
values(gene_length_range, Ranges)    :- length_ranges(Ranges).
values(nongene_length_range, Ranges) :- length_ranges(Ranges).

% HMM structure
length_model(Lengths,CodingAnnot) :-
        length_model_rec(nc,Lengths,CodingAnnot).

% end state
length_model_rec(State,[],[]):- final(State).

% other states
length_model_rec(State,[L|Ls],[NextState|Cs]) :-
        msw(trans(State),NextState),
        % Emission:
        get_range(L,Range),
        ((NextState==cs) ->
                msw(gene_length_range,Range)
                ;
                msw(nongene_length_range,Range)),
        % Recursion:
        length_model_rec(NextState,Ls,Cs).
```

***Conservation*, model m3**

This annotation model, figure 7.1 (*top right*), considers codon-conservation across species as a signal to gene-finding. The rationale is that if actual protein coding genes of an organism are subjected to destructive mutation, the organism is likely to suffer from it and likely perish. Thus the protein-coding genes of healthy organisms are typically less mutated than non-coding portions of their DNA. To detect conservation, each chunk is matched to a database of genome sequences of distantly related organisms[1] using the tblastn tool, which produce a gapped alignment of the matches. Only statistically significant matches (*E-value* $< 10^{-34}$)[2] and only one match per organism is reported. The states in the conservation model **m3** each emit the number of identity positions of reported matches to individual chunk positions, as sketched below. Training- and test-data can be constructed as described for the other models.

```
% Initialize
initial(begin).
final(end).

% Transitions from hidden states
values(begin,[c,n]).
values(trans(c),[c,end]).
values(trans(n),[c,n,end]).

% Emissions from hidden states
values(emit(_),[0,1,2,3,4,5,6,7,8]).

% HMM structure
cons_model(C,A):-
        initial(State0),
        msw(State0,State1),
        cons_model_recursive(State1,C,A).

% end state
cons_model_recursive(State,[],[]):- final(State)

% coding state
```

[1]The sequences: NC 004547, NC 008800, NC 009436, NC 009792, NC 010067, NC 010694 and NC 011283. They were chosen by hand from the RefsSeq database according their perceived evolutionary distance from each other and from E.Coli (NC 000913)

[2]The *E-value* describes the number of matches expected by chance. The lower the *E-value* , the more significant the match.

```
cons_model_recursive(c,[C|C2],[1|A2]):-
        !,
        msw(emit(c),C),
        msw(trans(c),Next),
        consorf_recursive(Next, C2, A2).

% noncoding state
cons_model_recursive(n,[C|C2],[0|A2]):-
        !,
        msw(emit(n),C),
        msw(trans(n),Next),
        cons_model_recursive(Next, C2, A2).
```

7.4 Experimental BAN-topologies

In the following we discuss and assess five BAN topologies for gene-finding constructed with these three signal-models as basic building blocks. The considered models are

- **m1**; a gene-finder based on the codon-preference signal alone.
- **m3**; a gene-finder based on the conservation-signal alone.
- **m1(m2)**; a gene-finder based on codon-preference, **m1**, conditioned on the length-signal, **m2**.
- **m1(m3)**; a gene-finder based on codon-preference, **m1**, conditioned on the conservation-signal, **m3**.
- **m1(m2,m3)**; a gene-finder based on codon-preference, **m1**, conditioned on both length- and conservation-signals, **m2** and **m3**.

also illustrated in figure 7.2 below.

All the conditional models are two-state HMM's, each with a coding and a noncoding state, just as **m1** and **m3**. We integrated the various signals in the emissions of the respective models, i.e.

- states in **m1(m2)** emit tuples $\langle C, L\rangle$, where C is one of the 64 codons, and L is a length-range.
- states in **m1(m3)** emit tuples $\langle C, P\rangle$, where C is one of the 64 codons, and P is a conservation degree.
- states in **m1(m2,m3)** emit triples $\langle C, P, L\rangle$, where C is one of the 64 codons, P is a conservation degree and L is length-range.

This results in a modest number of switches, each with a large number of possible outcomes. Another alternative is to include the number of states to two (one coding and one non-coding) per combination of signals. This,

in contrast, results in a vast number of switches in each model and we preferred the former alternative.

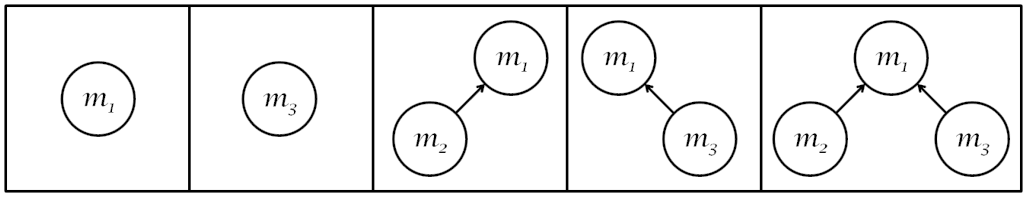


Figure 7.2: Graphical representations of the five experimental topologies

Training

We used the well-annotated *Escherichia Coli* genome and its curated gene annotations from the RefSeq database, (NC 000913), for training and testing. We randomly divided the ORFs of the genome into a training and a test set. Supervised training is done using only the former and the method for supervised training algorithm described in section 6.5. Specifically, for the three basic models **m1**, **m2** and **m3** we first applied the PRISM built-in predicates for learning with the constructed training-set. For each of the conditional models we first used the now fully parametrised sub-models to annotate all chunks of the E.Coli-genome and subsequently produced the necessary conditional training goals (similarly for the chunks in the test-set).

Accuracy

We report prediction accuracy results for both sets. Accuracy is measured as $\mathit{Sensitivity}(SN) = \frac{TP}{TP+FN}$ and $\mathit{Specificity}(SP) = \frac{TP}{TP+FP}$, with respect to annotation of start and stop codons. The results are summarized in table 7.1.

7.5 Results

The results in table 7.1 provides a convenient basis for assessing the pro's and con's of the alternative topologies. It can be observed that all our models have good generalization capabilities, since the performance is very similar on both the training and test set.

The best model seems to be **m1(m2)**, (i.e., codon preference conditioned on length) which achieves a significant increase in specificity with

	Training set (114429 ORFs, 2075 genes)			
BAN	SN_{start}	SP_{start}	SN_{stop}	SP_{stop}
m1	0.7701	0.2935	0.9711	0.3701
m3	0.0636	0.0322	0.8255	0.4183
m1(m2)	0.6723	0.5011	0.9345	0.6965
m1(m3)	0.4405	0.2243	0.8255	0.4204
m1(m2,m3)	0.4361	0.2228	0.8255	0.4217
	Test set (114404 ORFs, 2065 genes)			
BAN	SN_{start}	SP_{start}	SN_{stop}	SP_{stop}
m1	0.7564	0.2920	0.9719	0.3751
m3	0.0140	0.0072	0.8412	0.4298
m1(m2)	0.6489	0.4896	0.9433	0.7117
m1(m3)	0.4315	0.2216	0.8416	0.4323
m1(m2,m3)	0.4174	0.2149	0.8416	0.4333

Table 7.1: Accuracy of predictions using different BAN topologies.

only slightly degraded sensitivity, e.g. it predicts fewer genes but those that are predicted are more reliable.

By themselves, both **m1** (codon preference) and **m3** (conservation) have reasonable stop specificity, but **m3** displays a consistent tendency to predict too long genes, leading to severely decreased start specificity.

Interestingly, and contrasting to our expectations, conditioning codon preference on the conservation additional signal, i.e., in **m1(m3)**, does not improve prediction accuracy much. It does lead to slightly better stop specificity but it tends to degrade the start specificity.

Additionally, conditioning on the length signal as done in **m1(m2,m3)** does not seem to help, even though the impact observed in **m1(m2)** was quite significant.

It would seem that the **m3** signal dominates decisions about which orf's should be predicted as coding, and this gives a direction for further research.

7.6 Remarks

These experiments shows how separate annotation models can be combined in alternative BAN-topologies, and how the relative quality of those topologies may be evaluated and discussed. This way, a BAN analysis constitutes

a useful approach for exploring new signals of new combinations of signals in a systematic way.

Chapter 8

Example 2: Approximation by Decomposition

8.1 Introduction

The experiments in this chapter concerns a somewhat different approach to problem-decomposition published [21], and also in part in [20, 42].

They explore two distinct aspects of the basic BAN-methodology.

1. Given a complex model, that is canonical in the sense that it is correctly models a number of DNA features, but also too complex for predictive inference, it may be possible to decompose it into manageable sub-models to establish an approximated analysis, and

2. while we cannot establish the canonical annotations because of the complexity of that model, we may be able to evaluate the approximative annotations by compare to *sampled* canonical annotations.

The rationale for these experiments, stem from the observation that for a given computationally complex annotation task, not all subtasks need be equally demanding. It is usually possible, for example, to identify portions of a context free grammar, that are in fact regular, in that the parts of the language covered by these portions of the context-free grammar could equally well be defined by a regular one. Since parsing with a regular grammar is linear in the length of the sequences to be parsed as opposed to the cubic complexity of context-free parsing, there is a potentially very big efficiency gain to be had, if those less complex parts of the grammar can be distinguished from the computationally more demanding parts.
Suppose for example that we have a language of two classes of sentences. One class A is context-free and requires a PCFG for parsing, the other class

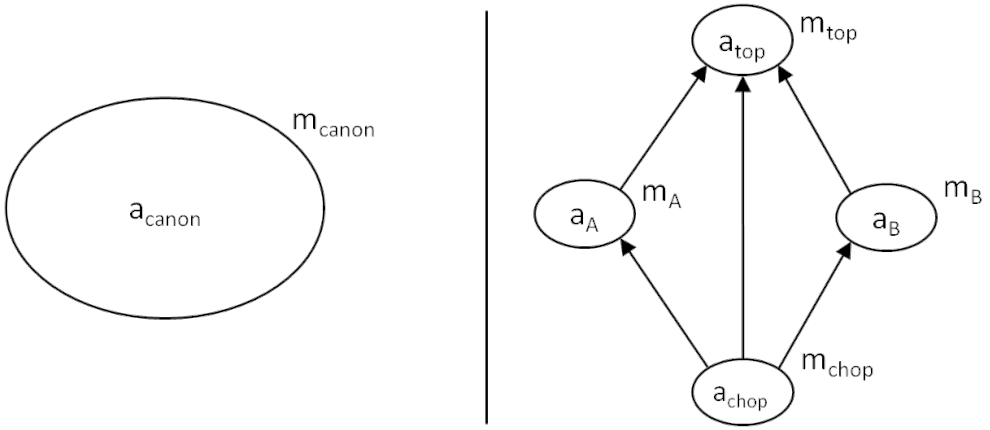


Figure 8.1: Sketch of the complex canonical topology and the decomposed approximative topology.

however, is merely regular. Suppose furthermore that we can represent a complex grammar m for the language as a PCFG of the following form (probabilities omitted):

$S \rightarrow A\ S \mid B\ S \mid \epsilon$
$A \rightarrow$ rules for A-class sequences
...
$B \rightarrow$ rules for B-class sequences
...

such that only the A-rules involves features requiring context-free analysis, while the rules for B-class sequences in principle could be stated as a HMM instead. Then m can simply be reformulated as the interaction of four interacting sub-models (also sketched in figure 8.1):

- a *chopper-model*, i.e, a HMM for S of states A and B each initiating the corresponding sub-model,
- an *A-model*, i.e, PCFG for A-type periods, and
- a *B-model*, i.e., a HMM for B-type periods.
- a *top-model*, i.e., some logic to combine the results

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

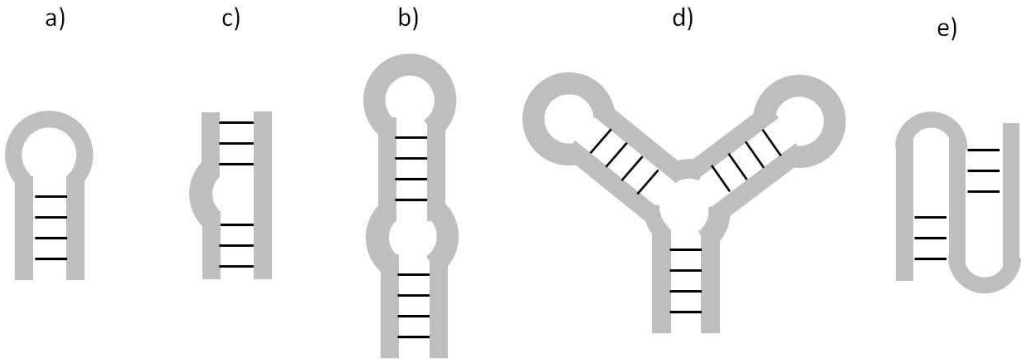


Figure 8.2: *Hairpins* or *hairpin loops* consists of a *stem* of mutually abstracted letters and a *loop* of unpaired letters

8.2 Annotation tasks

We show here an example of a pair of canonical and approximative model. The canonical model m^c is a simplified PCFG that resembles the sort of models that one will expect for genomic sequences for prokaryotic organisms. It distinguishes between sub-sequences 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 8.2 a), 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 sub-models of m^c for the each of the two kinds of sub-sequences.

As discussed, m^c requires cubic time in terms of sequence length 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.

8.3 Constituent models

Both the canonical model m^c and the approximative one m^a can be described in terms of the following components:

- m_{chop}, a two-state HMM that in each state emits an entire subsequence DNA - invoking a coding or non-coding sub-model depending on the state.
- $m_{non-gene}$, a HMM for non-coding regions.
- $m_{gene-pcfg}$, an assumed canonical PCFG for coding regions that takes both orf-structure and possible hairpins in to regard.
- $m_{gene-hmm}$, an approximating HMM for coding regions that only regards codon-structure.
- m_{approx}, a top-model for combining sub-annotations.

Identifying subsequences, m_{chop}

This kind of model follows the exact same structure as all the previous two-state HMMs that I have shown so far, except that instead of determining the emissions from each state by a `msw/2`-call, emissions are instead determined by invoking an entire sub-model of the type corresponding to the state. There are in fact two instances of this, namely one for the canonical analysis and one for the approximative. They differ only in their respective gene-models i.e., the HMM-routine for canonical version looks like this:

```
% Transitions from hidden states
values(dna,[gene, nongene]).
values(trans(gene),[gene,nongene, end]).
values(trans(nongene),[gene,end]).

% Canonical chopper
dnaCanon(S,L,A):-
  msw(dna,State1),
  dnaHMM1(State1,S-[],0,L,A-[]).

% end state
dnaHMM1(end,S-S,P,P,A-A).

% Other states
dnaHMM1(State,S1-S3,P1,P3,A1-A3):-
  % emission
  (
  State = gene,
      genePCFG([gene],S1-S2,P1,P2),
```

```
      A1=[(gene,P1,P2)|A2]
  ;
  State = nongene,
      nongeneHMM(ns,S1-S2,P1,P2),
      A1=[(nongene,P1,P2)|A2]
  ),
  % transition and recursion
  msw(trans(State),Next),
  dnaHMM1(Next,S2-S3,P2,P3,A2-A3).
```

HMM-based submodels, $m_{non-gene}$ and $m_{gene-hmm}$

These are straightforward Markov chains for emitting non-coding and coding sequencing respectively. I Include their states and Markov processes below:

```
% Gene HMM
%=========
% Transitions
values(trans(gene(start)),     [gene((codon,1))]).
values(trans(gene((codon,1))),[gene((codon,2))]).
values(trans(gene((codon,2))),[gene((codon,3))]).
values(trans(gene((codon,3))),[gene((codon,1)),gene(stop)]).
values(trans(gene(stop)),[end]).

% Emissions
values(emit(gene(start)),      [[a,t,g],[g,t,g],[t,t,g]]).
values(emit(gene((codon,_))), [a,c,t,g]).
values(emit(gene(stop)),       [[t,a,a],[t,g,a],[t,a,g]]).


% NonGene HMM
%============
% Transitions
values(trans(ns),[ns,end]).

% Emissions
values(emit(ns),[a,c,t,g]).
```

Canonical gene-grammar, $m_{gene-pcfg}$

The canonical gene grammar is a PCFG as defined and exemplified in example 6 on page 35. Its implementation has the exact same structure as

the one sketched in section 5.2. Below I include the grammar rules in a shortened form and a generic parser. In the actual implementation, there are three versions of each of the nonterminals `codons`, `hairpin`, and `loops`, to account for the codon structure of genes.

```
% Grammar d) Gene PCFG
values(gene,     [[start, codons, stop]]).
values(start,    [[a,t,g],[g,t,g],[t,t,g]]).
values(stop,     [[t,a,a],[t,g,a],[t,a,g]]).
values(codons,   [[n, codons],
                  [a, hairpin, t,],
                  [t, hairpin, a,],
                  [c, hairpin, g,],
                  [g, hairpin, c,]
                  ]).
values(hairpin,  [[a, hairpin, t,],
                  [t, hairpin, a,],
                  [c, hairpin, g,],
                  [g, hairpin, c,],
                  [loop]
                  ]).
values(loop,     [[l, loop],
                  []
                  ]).
values(l,        [[a],[c],[t],[g]]).
values(n,        [[a],[c],[t],[g]]).

% grammar symbol declarations
startsymbol(gene).

nonterminal(gene).
nonterminal(start).
nonterminal(codons).
nonterminal(hairpin).
nonterminal(loop).
nonterminal(stop).
nonterminal(l).
nonterminal(n).

terminal(a).
terminal(c).
terminal(t).
terminal(g.
```

```
% parser
pcfg(Sequucen):-
          startsymbol(Start),
          msw(Start, RHS),
          pcfg(RHS,Sequence).

% No more symbols to rewrite
pcfg([],Sequence):-
          Sequence = [].

% Head is a noterminal, rewrite it, and continue
pcfg([First|Rest], Seqeunce):-
          nonterminal(First),
          msw(First,RHS),
          pcfg(RHS,Prefix),
          pcfg(Rest,Postfix),
          append(Prefix, Postfix, Seqence).

% Head is a terminal copy it and continue.
pcfg([First|Rest], Seqeunce):-
          terminal(First),
          pcfg(Rest, Postfix),
          append(First, Postfix, Seqence).
```

Coordinating topmodel m_{approx}

This is not really a model, but rather a predicate that calls the respective sub-models in order and computes the combined probabilities form the probabilities of sub-models.

8.4 Experimental topologies

In BAN terminology we are considering two different topologies. also illustrated in figure 8.3:

- the canonical m^c, where the chopper model controls the non-coding HMM and the gene-PCFG in one complex annotation-model (figure 8.3, *Left*).
- the approximative m^a, where chopper controls the non-coding HMM and the approximative gene-HMM to identify putative sub-sequences of the two types. In this model sub-sequences of gene-type are then

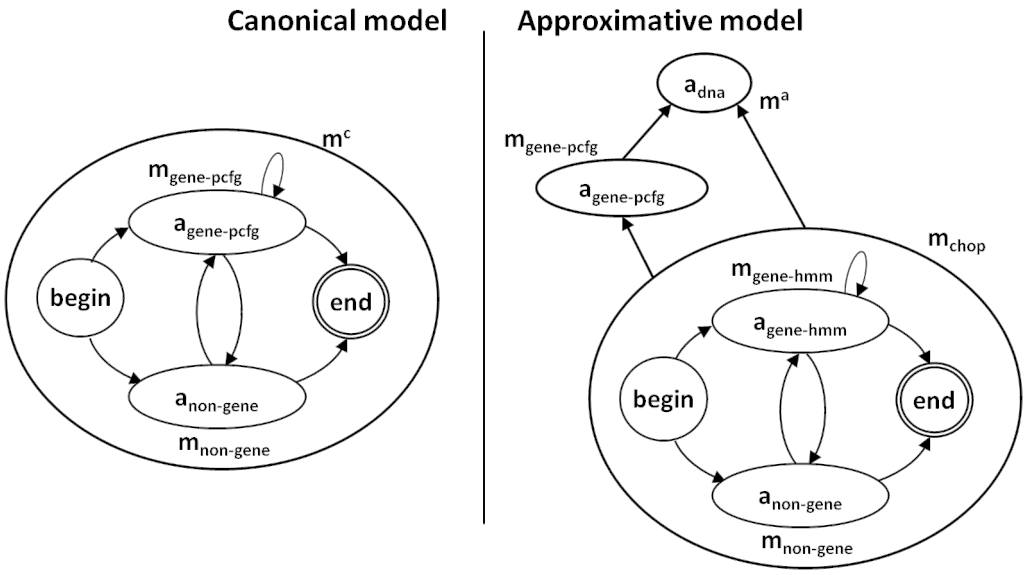


Figure 8.3: Sketch of the experimental toplogies: one canonical and one approximative.

passed on to complex gene-PCFG for detailed annotation.The resulting annotations are then combined in the top-model (figure 8.3, *Right*).

The approximative approach exemplified here, relies on the assumption that the simple HMM-based chopper model, when all submodels are porperly trained, get the partitioning into sub-sequences reasonably right, even without the detailed analysis from the gene-PCFG model. This is of course not always the case, and the experiments seek to clarify how good the approximation is.

8.5 Evaluation by sampling

As I have discussed earlier, we may in absence of authoritative test data for evaluation, produce such data instead with an authoritative sampling-model. In the present example, the canonical model takes the place of sampling model to produce test-data, with which to evaluate the approximative model.

Let us consider a single sample $\langle s, a^{samp} \rangle$ generated from the canonical model m^c. We can run s through the approximative model in question, m^a,

and obtain its best annotation a^{approx} as described above.

To compare the two, we can measure their probabilities $p^{samp} = p^c(a^{samp}, s)$ and $p^{approx} = p^c(a^{approx}, s)$; notice that we measure both probabilities 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 experiments that follows, we apply a simple principle for measuring similarity that may apply independently of actual sort of annotations in question. 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 of 12 character, i.e., a similarity measure of 0.667. When only the classification of particular 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 sub-sequences. So the similarity between

`nnnnnnngggggggggggnn`

and

`nnnnnnnggggggggggggn`

is $\frac{19}{20} = 0.95$.

8.6 Experiments

We conducted four experiments[1], whose results are summarized in Fig. 8.4 and 8.5. Everywhere we measured the probabilities in log space, so the

[1]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 sub-model and put this best sub-parse into the annotation instead of the sampled one. Be aware that this is different from the approximating model, 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 the reported tests may be a bit more accurate in sorting out bad annotations produced by m^c.

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 scatter plots in Fig. 8.4 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 b) training the models using the 100 shortest annotated genes from E. Coli K12, whose lengths are between 50 and 178 letters; c) we constrained generated samples to those satisfying the inherent length constraint implied by the training data; and d) we applied both of the changes b) and c).

8.7 Results

From Fig. 8.4 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. 8.4 and 8.5 indicate here that most approximated and sampled annotations correlates closely. Even with

these deviations, the two measures correlate perfectly in the vast majority of cases, coinciding in (0, 100%) coordinates, as shown in Fig 8.5. All in all this indicates that especially with the precautions taken in experiment d), we may trust the approximating model to produce reasonably reliable results.

8.8 Remarks

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 analysed, 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 or sufficiently annotated data, we doubt that such quantitative measures can be devised. Also here, it seems however, that information theoretical measures like conditional entropy, mutual information and redundancy might provide parts of the answer to this issue.

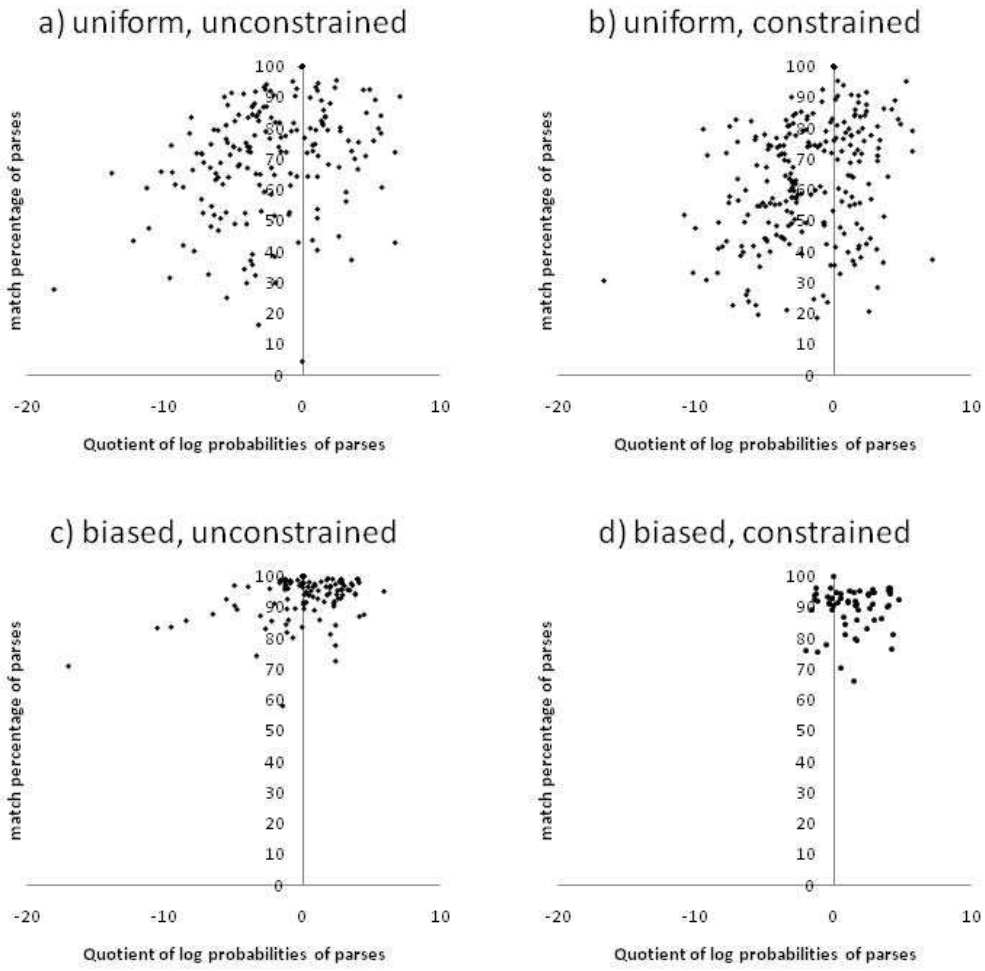


Figure 8.4: Representation of the sampling distributions of the four conducted experiment. Along the X-axis are logarithmic probability quotients of the canonical vs. approximative analyses. A quotient of 0 represents identity. Along the Y-axis are the corresponding percentages of Hamming-like similarity 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	% quotient = 0	% match = 100	avg. coordinate
a)	82.3%	82.2%	(-0.42, 94.48)
b)	83.9%	83.7%	(-0.38, 91.84)
c)	86.7%	86.5%	(0.09, 99.38)
d)	94.7%	94.6%	(0.01, 99.17)

Figure 8.5: 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 8.4. 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

.

Summing up

In this part of the dissertation, I have presented a general methodology called Bayesian Annotation Networks for defining probabilistic annotations models and experimental combinations of them, reflecting possible inter-dependencies. The methodology was exemplified for models implemented in PRISM, but the formal definition does not rely on any programming language in particular and the methodology is expected to be language independent. I demonstrated the methodology with two non-trivial examples from the domain of DNA annotation.

In the first I demonstrated how separately defined annotation models may take the role of constituent models in a BAN and how they can be trained, either by using existing training data or constructing training from properly parametrised parents in the network. I showed how inference takes place iteratively to properly establish the conditions for dependent models, one constituent at a time, parent nodes before their children. I also demonstrated how traditional accuracy measures can be established systematically for alternative topologies to form a sound metric of comparison.

In the second example application, I took a slightly different approach, where an existing complex model was decomposed into an approximative BAN. The input was first partitioned into sub-sequences of perceived different types, in what we could call a *biased chunking scheme*. Depending on their respective types, sub-sequences were then passed on to corresponding detailed analyses and the individual annotations subsequently combined. In this example, I also experimented with a sampling based method for evaluation, where the complex model was used for sampling a large number of data, along with annotations, to function as evaluation data for the approximative analysis. I also demonstrated the impact of trained and constrained sampling.

Part III

Conclusions and Achievements

Chapter 9

Future Work

9.1 Introduction

In chapter 6 I introduced the basic framework of how to construct annotation models and integrate them as constituent nodes in a BAN, and in chapters 7 and 8, I described applications to non-trivial problems relating to gene-finding. I exemplified showed how the predictions of a BAN may be evaluated by way of sensitivity and specificity if sufficient authoritatively annotated data is available for the purpose. I also exemplified how the approximative analyses could be evaluated by analysing the difference between approximative and canonically sampled annotations of common data-sequences, either directly or by way of their respective probabilities.

Future work includes researching suitable annotation tasks that benefits from the extra expressive power. Current plans the in LoSt-project concerns an examination of what we call *gene-finder evasive genes* in E.Coli. We have implemented annotation models for establishing a set of gene-finder evasive genes, i.e., trusted existing genes that none of the state of the art-genefinders detects. We plan to analyse this list of evasive genes along various measures to establish what distinguishes them from the rest of the known genes. The reason for evasiveness likely includes both relational and statistical aspects of recognised DNA-features and likely also some features that have yet to be discovered, making it a reasonable problem for BAN-analysis. We hope to identify one or more specific factors that may be incorporated in future genefinders to increase accuracy.

A more thorough treatment of BAN-evaluation is however also necessary for comparison to existing systems and for general application in the domain.

9.2 BAN-evaluation

As mentioned already there are several important aspects to evaluation of BAN's.

First of all, for the general methodology to be useful for analysis purposes it is necessary to develop robust methods for evaluation of the approximative method of inference that we employ.

Secondly, it is of course important to be able to evaluate the alternative BAN-topologies in a robust manner, including at least the evaluation of

- the predictive power of both the BAN as a whole and of the constituent models
- the relative relevance of a constituent model to a given topology
- how well constituents are integrated in a given BAN

9.3 Adapting methods for traditional Bayesian Networks

In the domain of probabilistic models as a whole, there is a long tradition for optimizing and approximating inference with Bayesian network, see [24] for an extensive exposition. It is likely that a number of these methods may be adapted to apply to BAN's as well.

9.4 Statistical testing for evaluation of predictive power

The predictive power of an annotation model is hard to evaluate when there is no golden standard to which to compare predictions. In the domain of statistical testing there is however a large set of methods for measuring the significance of of results in the absence of traditional accuracy measures. These methods, demands the formulation of hypotheses that can be tested by the distribution of a sufficiently large set of sampled results. In the case of for example the experiments in chapter 8, where we compute the match between the canonically sampled annotation of some sequence and the approximated annotation of that sequence, a so-called *t-test* could be applied to verify or reject the hypothesis that the proposed method produces good approximations, i.e., above some level of confidence, of the canonically sampled annotations. The t-test is a very standard statistical test that

computes the t-value in terms of the average divergence of a set of values, here the match-percentages, from some carefully chosen mean-value. The t-value is then compared to a table of confidence scores to obtain the level of confidence, to which the hypothesis hold. There are several related tests for measuring significance and other evaluation-measures, and which to choose depends on what to measure, the domain of the results, whether they are continuous or discrete, etc. And important study includes researching and experimenting with statistical tests for predictive power of BAN's, either in general or w.r.t. different applicable domains of analysis. Statistical tests may also apply, that measure the quality of the approximative analysis.

9.5 Information theory for BAN-evaluation

Recall from chapter 6 that the predictive analysis in a given BAN must be considered an approximation of an ideal analysis, albeit that the ideal analysis is practically impossible for non-trivial applications. This hinges on the fact that in each step of the inference we accept one most likely annotation $approx_i$ from a model M_i as a representative of the entire distribution of possible a_i annotations. We would like to be able to analyse the quality of this and I wrote that given the assumed complexity of the ideal analysis we cannot measure the divergence between the approximate and ideal analysis.

A related matter concerns the ability to evaluate the actual benefit from including a candidate constituent model in an experimental BAN. For both these evaluation tasks I suggested different alternatives:

- Standard accuracy measures of sensitivity and specificity may be used in the presence of sufficient authoritative test data, which we did in the experiments of chapter 7.
- evaluation relative to sampled ideal annotations can be used if sampling version of the complex model can be made available, as also explored in the experiments of chapter 8.

I also indicated an evaluation might be possible that involves information theoretical analysis of the probabilistic parameters of the involved annotation models alone, and thus precludes the need for actual data instances other than for training. An interesting prospect is to apply measures from Information Theory [69]. Information Theory is defined for random variables and in appendix A, I include the theoretical background from Information Theory and a detailed sketch of how basic theory may adapted to apply to annotation models and BAN's. A formal proof of the claims and supporting experiments is however still ongoing work.

9.6 The LoSt-framework

Finally, the general methodology defined in this work forms a flexible, and powerful framework for conducting sophisticated experiments in the important academic field of DNA-annotation. In order to make the methodology accessible also to non-experts in computer science, there is an ongoing effort in the LoSt-project towards a unified modular framework for managing constituents and topologies, called the LoSt-framework. In this framework, each constituent is implemented separately along with an interface specifying how to interact with the model. Each candidate topology is then implemented so as to specify the order of application of constituent model as well as what inferential tasks to perform, i.e., sampling, training or prediction An detailed file-system for keeping track of intermediate annotation and parameter-files is also part of the framework, that allows reuse of exisiting annotations in subsequent analyses. Further development of this interface is ongoing.

Chapter 10

Related Work

10.1 Introduction

The work described in this dissertation is inherently interdisciplinary and relates to varying degrees to at least the following fields of scientific research: *bioinformatics*, *formal language theory*, *probability theory*, *artificial intelligence*, *machine learning*, *logic programming*, and *information theory*. The general optimizing strategies of problem decomposition and data partitioning apply to all these fields and it is quite impossible to present a thorough survey of related work in these fields at this point. I will however attempt to place the present work in relation to the most important trends of the above areas.

10.2 Formal language theory

HMM's, and SCFG's are traditional methods for sequence analysis that can be seen as instances of probabilistic-logic models and there exists a plethora of efficient algorithms and implemented systems, (see [29] for an extensive background and overview), including *Factorial Hidden Markov models*, *FHMM's* [31] and *Hierarchical Hidden Markov Models*, *HHMM's* [47].

Factorial HMM's

The rationale for FHMM's is that a given signal may stem from several different sources. The situation is often described as that of a dinner party, where many simultaneous conversations constitute the total signal and the focus of individual participants determines what message to listen to and what irrelevant, in this respect, parts of the total signal to factor out. As

such there are clearly many applications in the domain of DNA-annotation and FHMM's candidate for potentially very useful constituents in a BAN as defined here.

Hierarchical HMM's

HHMM's describe a class of HMM's, where the so-called *internal* hidden states are themselves HMM's. These states do not emit single symbols but rather entire sequences of symbols by invoking their respective sub-models. Thus the chopper models m_{chop} of chapter 8 can be seen as instances of HHMM's.

These provide a catalogue of possible sub-models to be used within our BAN methodology.

10.3 Probabilistic logic programming

Being an integral part of the LoSt, project there are of course important relations to *Probabilistic logic programming*. Several general and powerful formalisms have been suggested as probabilistic extensions to logic programming languages within the last 15 years, we may mention PHA and ICL [55, 56], PRISM [65, 66], that we have exemplified, Stochastic Logic Programs [45, 46], Relational Bayesian Networks [35], CLP(BN) [23], and ProbLog [39, 37, 38] See also [26] for an extended survey.

There is a growing interest in such models for bio-informatical applications. In particular [11, 4] discuss applications to Systems Biology, but also various kinds of sequence analysis have been studied.

An early inspiration to the preprocessing approach of chapter 8 is [13], where similar ideas were applied for a comparative test of three different gene-finder programs [9, 40, 44]. 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 sampling artificial genomic sequences. These sampled 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 chapter 8 was used to produce auxiliary annotations for speeding up training.

10.4 Compositional probabilistic models

The general BAN-methodology is closely related to *Dynamic Bayesian Networks* (DBNs) of [48]. Similar to BANs, DBN's also exploit the attractive

properties of traditional Bayesian networks to formulate highly versatile networks of separate probabilistic models for sequential analysis. Inference in DBN's takes place in so-called time-slices in each of which all constituents are applied to the position or portion of the sequence corresponding to that time-slice. By our definition of a BAN, the detailed dependencies between individual models in the network are left abstract, but a concrete instantiation of a BAN may indeed be a DBN. However, as the nodes in a BAN may be arbitrary probabilistic models, for instance context-free grammars, not all BAN instantiations can be represented as DBNs. Oppositely, we only define BANs for discrete models but DBNs may include continuous-valued nodes.

10.5 Classification

In the sub-field of machine-learning concerned with automatic classification techniques, it is common to combine the results of different classifiers of the same phenomena in such ways that the combined classifications outperform the individual constituent classifiers. Such methods are generally known by the name *ensemble methods*, which covers a wide range of different ways to the combine classifiers [60]. Our method is related but quite different; this is not just because we consider sequence annotation rather than classification, but also because constituent models of a BAN may model very different phenomena.

In biological sequence analysis, the most successful genome annotation programs are *combiners* [32]; programs which combine different sources of annotation evidence using some sort of weighting scheme. Evidence may come in diverse forms, including comparative analysis sources [57], but are typically predictions (e.g. annotations) from other annotation programs (e.g. gene finders). Brent [8] makes a distinction between combiners and joint models, where joint models are described as models which consider the full joint probability distributions evidence and combiners as probabilistic models of the relative accuracy of evidence sources they are combining. Using our approximate inference algorithm we have a situation similar to combiners in that predictions of parents are combined by child nodes.

While many combiners use non-probabilistic combination methods, several are explicitly based on principles of (dynamic) Bayesian networks [53, 43]. A main difference is that our framework allows multi-layered and branching topologies where the combiners are usually just single layered probabilistic models.

Our approach also has analogies to *annotation pipelines* [58, 10] where a complex sequence of analysis steps are performed in a possibly branching topology and perhaps synthesized (e.g. by a combiner) in a final annotation as the last step. Opposed to combiners, pipelines usually allow complex topologies like our framework. However, such pipelines are usually just practical and pragmatic ways of combining existing tools and incorporate probabilistic modeling only to a very limited degree.

There are other declarative approaches to combining evidence in biological sequence analysis. In GAZE [34], a configurable XML-based specification describes a particular composition of evidence sources. However, GAZE integrates existing tools, where our PRISM based approach allows for much more modelling flexibility and has a clear and well-defined semantics.

Chapter 11

Achievements

11.1 Introduction

In this final chapter of my dissertation, I present identified achievements made and conclusions reached from the research described in chapters 6, 7, and 8 and repeat some of the directions for future work described in chapter 9. While to some extent falling beside the main focus of the dissertation, I will also make note of published work to which I contributed as a member of the highly cross-fertilizing LoSt-project community at Roskilde University.

11.2 Bayesian Annotation networks

We have proposed a Bayesian framework, *Bayesian Annotation Networks*, which allows the representation and composition of models for complex sequence analysis. In a modular way, it supports experimentation with and evaluation of models and signals and it is a practically useful tool for modelling and analysing sequences. I have motivated its application to the domain of biological sequence analysis and DNA-annotation in particular. I have shown how practically applicable analysis can be achieved by the use of tractable, incremental algorithms for inference and training, which can be implemented by successive calls to PRISM, and shown that these algorithms may produce useful annotations.

Especially for exploring novel or putative relationships in a complex system of influences, the BAN-approach offers a systematic tool for early, easy to understand and compare, full-scale experiments with alternative hypothesis. Such models are efficient enough to serve as independent standalone systems but will clearly benefit tremendously from even more effi-

cient tabling and modules support. BAN topologies are perhaps even more useful as prototypes for specialised implementations. Thanks to the clear semantics of Prolog and PRISM the BAN will serve a precise specifications of the different analytical task of the complex system.

BAN evaluation

As mentioned before, a BAN-analysis is, In general, an approximation of an ideal but impractical – often impossible – analysis. Since, by assumption, the ideal analysis is too complex for predictive inference, I have not presented any precise quantitative measures of the quality of the approximated annotations compared with the ideal ones though. This is also the case when evaluating the respective impacts of individual constituents of a BAN.

In the trivial case, where all freedom of choice is implemented in the top node of the Bayesian Network, the approximate algorithm coincides to the ideal.

Beyond the trivial case, however, it is difficult (impossible in general) to give sufficient conditions for which the approximate inference method will yield good results.

Traditional measures of accuracy

In the experiments of chapter 7, we settled with traditional measures of quality (e.g. sensitivity/specificity) for evaluating annotations and topologies and we applied cross-validation to build confidence about generality. Obviously, this may require a considerable amount of, possibly unavailable, labelled training data. A second consequence, also observed in chapter 7, is that the measure optimized by the training algorithm does not necessarily coincide with the external measure of quality. Model constraints and independence assumptions play a key role affecting the correlation between these measures.

Evaluation by sampling

In experiments of chapter 8 we intended to get the best of both worlds, flexibility and sophistication of the probabilistic-logic models combined with feasible execution times, by using preprocessors, e.g., based on existing and efficiently implemented technologies, as a way to reach realistic execution times.

In the absence of authoritative test-data, we explored a possible but somewhat complicated approach to evaluation based on using the complex impractical model for sampling annotated test data, and then compare with the annotations produced by the implemented approximative BAN. This approach requires an implementation of the complex model as a single PRISM model, where the conditional dependencies of the respective features can be represented directly in the switch-declarations. To get an indication of whether comparing probabilities for the two annotations is informative w.r.t. the quality of the approximative analysis, we applied also a measure of the similarity between the annotation sequences themselves in a straightforward, syntactic fashion. We noticed that the sampling method provides the best indications when the model has biased probabilities, as it is easier to distinguish between good and bad annotations and increase the probability of samples that resemble the training data.. 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).

Mutual information and conditional entropy

Finally, I have proposed how mutual information and conditional entropy may be calculated for probabilistic annotation models, grounding the above observations in classical informational theoretical terms. I discussed how, in these terms, it may be analysed in a systematic way how well suited a given annotation model is for the approximative setting of a BAN. I also discussed how the impact of including some constituent model in a given BAN might be represented in terms of mutual information relative in to the maximal entropy of the BAN. However, these efforts are still ongoing.

11.3 Other achievements

The ongoing work in the LoSt-community at Roskilde University has furthermore sprouted several initiatives and explorative forays into the intersecting areas of machine learning, logic programming, probability theory, graphical models for sequence analysis and constraint logic programming

Program transformation

Early experiments with PRISM was hampered by general and persistent difficulties in computing Viterbi in PRISM for models with "extra arguments", e.g., for measuring sequence length or collecting the Viterbi path of a model explicitly in the arguments of the recursive goal. The problem was identified as stemming from the how PRISM implements tabling for probabilistic goals. This inspired a general method [14], for automatically recognising and slicing offending arguments from the PRISM program to produce a version that caters for efficient Viterbi-computation. The arguments are then spliced back in and instantiated from analysis of the internal representation of the Viterbi-path. The method is under consideration for incorporation in subsequent versions of PRISM.

Constraints in probabilistic logic programming.

Based on common interest in the group and the realization of the potential benefit to probabilistic logic programming in general, quite a lot of work was published concerning constraint based methods and probabilistic reasoning.

In [15, 54] we introduced HMMs with constraints and showed how the familiar Viterbi algorithm can be generalized, based on constraint solving methods. HMMs with constraints have advantages over traditional ones in terms of more compact expressions as well as opportunities for pruning during Viterbi computations. We exemplified this by an enhancement of a simple prokaryote gene finder given by an HMM.

We elaborated on this work in [16], and showed how defining HMMs with side-constraints in Constraint Logic Programming have advantages in terms of more compact expression and pruning opportunities during inference. We presented a PRISM-based framework for extending HMMs with side-constraints and showed how well-known constraints such as *cardinality* and *all different* are integrated. We validated our approach experimentally on the biologically motivated problem of global pairwise alignment.

In [17] We furthermore showed how the Viterbi algorithm can be implemented in the declarative CHR programming language (Constraint Handling Rules, [30]).

Probabilistic graphical models for sequence analysis

An inherent characteristic of all subspecies of Hidden Markov models is their control by some sort of probabilistic, finite state machine, but which may differ in the detailed structure and specific kinds of conditional probabilities. In the literature, however, the different HMM subspecies tend to be

described as different beasts to some degree with modelling and inference methods defined from scratch in each particular case. In [19] we suggested a unified characterization using a generic, probabilistic-logic framework and generic inference methods, which also promote experiments with new hybrids and mutations. We indicated how this might even involve context dependencies that traditionally are considered beyond reach of HMMs.

In [33], an *Extended regular grammars* is presented for modelling repeating sections in DNA. Extended regular expressions are inherently nondeterministic and require procedural control such as backtracking. A probabilistic version of extended regular expressions is proposed, where the affinity for strings and matches can be learned from examples.

11.4 Conclusion

In this dissertation, I have researched an answer to the question: *Can we establish a system of compositionality for probabilistic annotation programs in PRISM that retains the strengths of declarative programming but keeps computational complexity low enough for practical application?*

I have researched how to implement clear, flexible and efficient systems using PRISM for accurate DNA-annotation and I have motivated the need for new flexible and expressive formalisms for biological sequence analysis and demonstrated the potential benefit from formulating such formalisms in probabilistic logic programming languages.

The developed framework, Bayesian Annotation Networks, represents a simple, flexible and general compositional approach to answer the overall question. The approach furthermore has the added benefit of allowing local structure to be exploited both in learning and prediction. The framework was implemented and exemplified in PRISM but is in essence language independent.

I have experimented with several applications of the modular methodology and demonstrated the framework as a general tool for expressing stand-alone systems that are efficient enough for application to non-trivial problems from the general domain of DNA-annotation. It is clear, that the efficiency of the framework depends on the efficiency of integrated systems, including PRISM and BProlog, and that it would benefit greatly from support of modules and even more efficient methods for tabling with lists.

Inference in the framework is however approximative and a remaining challenge concerns establishing general quantitative measures of the quality

of that approximation and how to evaluate the suitability of candidate components in a BAN.

The answer to the posed question, must then be that such system can likely be developed, for example along the lines of Bayesian Annotation Networks presented here. The developed methodology is however approximative and satisfying methods for evaluation of the approximation is still future work. Several experiments documented here does however indicate a high degree of correlation between the approximative and the exact analysis at least for some domains. The proposed methodology seems applicable to many important problems in bio-informatics and likely extends to other domains as well.

Appendix A

Information Theory for BAN's

A.1 Basic definitions from Information Theory

In Information theory, the *information content*, $\mathcal{I}(x)$ of the observation of a specific outcome x of a random variable X is inversely proportional to its probability $p(x)$. Information is traditionally measured in log-space to gain additivity and the base of the used logarithm gives the unit of measure. We use base-2 and get the bit as unit of measure, i.e.:

Definition 42 (Information Content) *The* information content, $\mathcal{I}(x)$, *of an outcome x of a random variable X, is given as :*

$$\begin{aligned} \mathcal{I}(x) &= \log_2 \frac{1}{p(x)} \\ &= -\log_2 p(x) \end{aligned} \tag{A.1}$$

Definition 43 (Entropy) *The* entropy, $\mathcal{H}(X)$, *of an random variable, X, is given as the weighted average of information content of all outcomes of X:*

$$\begin{aligned} \mathcal{H}(X) &= \sum_x p(x)\mathcal{I}(x) \\ &= -\sum_x p(x)\log_2 p(x) \end{aligned} \tag{A.2}$$

Definition 44 (Conditional entropy) *The* conditional entropy, $\mathcal{H}(X|Y)$, *represents the average uncertainty in the outcome of X after observing Y:*

$$\mathcal{H}(X|Y) = \sum_y p(y)\mathcal{H}(X|y) \tag{A.3}$$

where

$$\mathcal{H}(X|y) = \sum_x p(x|y) \log_2 p(x|y) \tag{A.4}$$

Definition 45 (Mutual information) *The mutual information $\mathcal{I}(X;Y)$ between two random variables X and Y can be calculated in terms of joint and marginal probabilities:*

$$\mathcal{I}(X;Y) = \sum_{x,y} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)} \tag{A.5}$$

or in terms of relative entropies, i.e., the difference between Entropy and conditional entropy :

$$\mathcal{I}(X;Y) = \mathcal{H}(X) - \mathcal{H}(X|Y) \tag{A.6}$$

A.2 Entropy and Conditional Entropy for annotation models

For a given sequence s, the entropy of an annotation model $m = \langle prog_m(s, a, parents), \theta_m \rangle$ must quantify the uncertainty in prediction the outcome annotation A, given $parents$, i.e., as per the definition of conditional entropy above:

$$\mathcal{H}(A|s, parents) = \sum_a p(a|s, parents)\mathcal{H}(a|s, parents)$$

and

$$\mathcal{H}(a|s, parents) = \sum_{\langle parents \rangle} p(a|s, parents) \log_2 p(a|s, parents)$$

Noting that, for an annotation model that can annotate each position in m different ways we have m^n different annotations of a sequence of length n, this is not practical, even for a decomposed model.

A possible alternative is to consider the entropy of an annotation model as the average uncertainty in the annotation of each position in the sequence. This approach exploit the structure of the annotation model itself. For example for a simple annotation-model based on a first-order output, HMM, M, with transitions, T, and emissions ,E:

$$\mathcal{H}(M) = -\sum_{\langle s,t,b \rangle} p((s,t),(t,b)) \log_2 p((s,t),(t,b))$$

where, $(s,t) \in T$ and $(t,b) \in E$.

Similarly the conditional entropy $\mathcal{H}(M|A)$ of a HMM M given the outcome of a parent-model A can be equally straightforwardly expressed if we keep to the position specific version, i.e.:

$$\mathcal{H}(M|A) = \sum_{a} p(a)\mathcal{H}(M|a) \tag{A.7}$$

where

$$\mathcal{H}(M|a) = -\sum_{\langle s,t,b,a \rangle} p((s,t),(t,b,a)) \log_2 p((s,t),(t,b,a)) \tag{A.8}$$

where, $(s,t) \in T$ and $(t,b) \in E$.

A.3 Mutual information of annotation models

Given that we can calculate the entropy and conditional entropy for annotation models we can also calculate the mutual information, $\mathcal{I}(M;A)$, that quantifies the amount of information shared by two models M and A, assuming a BAN consisting of a model M and a parent model A. To see how we may use mutual information to evaluate the impact of a parent A on M, consider these three cases:

$\mathcal{I}(M;A) = 0$: this indicates A contributes none of the information covered by M and thus might as well be left out of the topology, see figure A.1, 1).

$0 < \mathcal{I}(M;A) < \mathcal{I}(A)$: this indicates a non-empty intersection of information covered by both models, and the larger this intersection – the more relevant the information in A, figure A.1, 2).

$\mathcal{I}(M;A) = \mathcal{I}(A)$: here all of the information offered by A is relevant to M and meaning that unless that information is also offered by other sources A should be kept in the topology, figure A.1, 3).

In general we can define a measure of benefit of an annotation model A in a BAN M as follows:

$$\frac{\mathcal{I}(M;A)}{\mathcal{I}(M)}$$

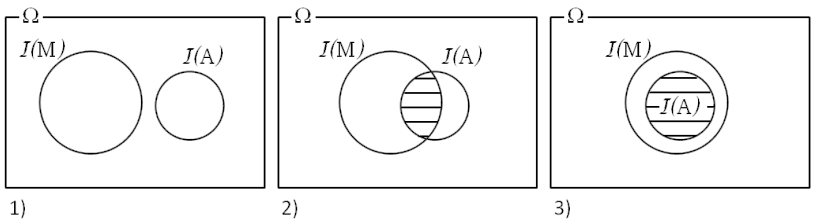


Figure A.1: Mutual information between probabilistic a probabilistic model M and an assumed possible parent model A, schematically illustrated as the intersection of distributions of atomic events represented by models M and A. In **1)** the intersection of information is empty, meaning that if the model A is a constituent in M it might as well be removed. In **2)** the benefit of keeping A as a constituent in M is quantified by the mutual information proportional to information of M. In **3)** all information in A is relevant for M

.

A.4 Relative Entropy

The sense in accepting a most likely event from a distribution of event as an approximation of the entire distribution of course depends on the distribution. To see how the entropy measure may be used to indicate whether a constituent model M is approximated well in a BAN consider first the trivial case where one annotation in the distribution of M has probability 1 and all others 0, figure A.2 1). Here the most likely annotation is clearly a good choice as a representative of the entire distribution. Note that in this case, $\mathcal{H}(M) = 0$.

In the opposite case all annotations have the same probability, e.g., $\frac{1}{n}$, figure A.2 2). In this case the most likely annotation represents the entire distribution poorly, and we note that we have maximal entropy for the given model, i.e., $\mathcal{H}(M) = \log_2 n$.

If we denote the maximal entropy for a given model M as $\mathcal{H}^{max}(M)$ we can indicate the chance that a given model is approximated well in a BAN by the following ratio, that Shannon calls the *redundancy* of the model:

$$1 - \frac{\mathcal{H}(M)}{\mathcal{H}^{max}(M)}$$

.

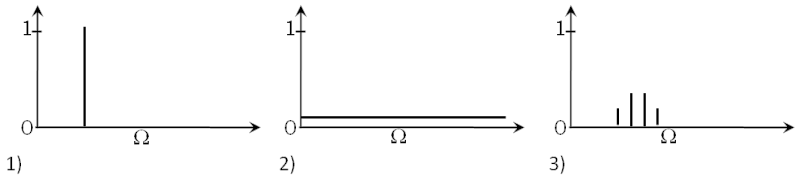


Figure A.2: Schematic sketch of three different distributions illustrating how the entropy of a model can be used as a measure of the distinguishing power of a model. the lower the entropy the better the distinguishing power of the model and the better the chance of a good approximation in our framework. In **1)**, one event dominates the distribution we have minimal entropy. In this case, the most likely event represents the distribution perfectly. In **2)** all events are equally likely an thus any one of would be an equally bad approximation of the distribution. In **3)** is the case where several event are almost equally likely for there is at the same time some difference between high and low probability events. In general the greater the difference between event probabilities the better the most likely event will represent the distribution.

Bibliography

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Theory of Parsing, Translation and Compiling*. Prentice Hall, 1972.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison Wesley, Reading, MA., 1986.

[3] Bruce Alberts, Dennis Bray, Julian Lewis, Martin Raff, Keith Roberts, and James D. Watson. *Molecular Biology of the Cell*. Garland Science, New York, 3 edition, 1994.

[4] Bruno Apolloni, Robert J. Howlett, and Lakhmi C. Jain, editors. *Knowledge-Based Intelligent Information and Engineering Systems, 11th International Conference, KES 2007, XVII Italian Workshop on Neural Networks, Vietri sul Mare, Italy, September 12-14, 2007. Proceedings, Part I*, volume 4692 of *Lecture Notes in Computer Science*. Springer, 2007.

[5] Robert B. Ash. *Basic probability theory*. Wiley, 1970.

[6] John Besemer and Mark Borodovsky. GeneMark: web software for gene finding in prokaryotes, eukaryotes and viruses. *Nucleic acids research*, 33(Web Server issue):W451–4, July 2005.

[7] Mark Borodovsky and James D. McIninch. Genmark: Parallel gene recognition for both dna strands. *Computers & Chemistry*, 17(2):123–133, 1993.

[8] Michael R. Brent. Steady progress and recent breakthroughs in the accuracy of automated genome annotation. *Nature reviews. Genetics*, 9(1):62–73, January 2008.

[9] Chris Burge and Samuel Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997.

[10] Brandi L. Cantarel, Ian Korf, Sofia M. C. Robb, Genis Parra, Eric Ross, Barry Moore, Carson Holt, Alejandro Sánchez Alvarado, and Mark Yandell. MAKER: an easy-to-use annotation pipeline designed for emerging model organism genomes. *Genome Research*, 18(1):188–196, 2008.

[11] Jianzhong Chen, Stephen Muggleton, and Jose Santos. Learning probabilistic logic models from probabilistic examples (extended abstract). In Hendrik Blockeel, Jan Ramon, Jude W. Shavlik, and Prasad Tadepalli, editors, *ILP*, volume 4894 of *Lecture Notes in Computer Science*, pages 22–23. Springer, 2007.

[12] N Chomsky. Three models for the description of language. *Information Theory IRE Transactions on*, 2(3):113–124, 1956.

[13] 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.

[14] Henning Christiansen and J. Gallagher. Non-discriminating arguments and their uses. *Lecture Notes in Computer Science*, 5649:55–69, 2009.

[15] Henning Christiansen, Christian Theil Have, Ole Torp Lassen, and Matthieu Petit. A Constraint Model for Constrained Hidden Markov Models. In *The proceedings of the 2009 Workshop on Constraintbased Methods in Bioinformatics*, pages 19—-26, 2009.

[16] Henning Christiansen, Christian Theil Have, Ole Torp Lassen, and Matthieu Petit. Inference with constrained hidden Markov models in PRISM. *Theory and Practice of Logic Programming*, 10(4-6):449–464, July 2010.

[17] Henning Christiansen, Christian Theil Have, Ole Torp Lassen, and Matthieu Petit. The Viterbi Algorithm expressed in Constraint Handling Rules. In *7th International Workshop on Constraint Handling Rules*, page 17, 2010.

[18] Henning Christiansen, Christian Theil Have, Ole Torp Lassen, and Matthieu Petit. Bayesian Annotation Networks for Complex Sequence Analysis. In John Gallagher and Michael Gelfond, editors, *Technical Communications of the 27th International Conference on Logic Programming (ICLP'11)*, pages 220–230, Lexington, KY, USA, 2011. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[19] Henning Christiansen, Christian Theil Have, Ole Torp Lassen, and Matthieu Petit. Taming the Zoo of Discrete HMM Subspecies & Some of their Relatives. In Gemma Bel-Enquix, Veronica Dahl, and M. Dolores Jiménez López, editors, *Biology, Computation and Linguistics, New Interdisciplinary Paradigms*, volume 228, pages 28–42, Tarragona, Spain, 2011. IOS Press.

[20] Henning Christiansen and Ole Torp Lassen. Optimization and Evaluation of Probabilistic-Logic Sequence Models. In *Workshop on statistical and Relational Learning in Bioinformatics*.

[21] Henning Christiansen and Ole Torp Lassen. Preprocessing for optimization of probabilistic-logic models for sequence analysis. In *Lecture Notes in Computer Science*, volume 5649, pages 70–83. Springer, 2009.

[22] William W. Cohen. *A computer scientist's guide to cell biology*. Springer, 2007.

[23] Vítor Santos Costa, David Page, Maleeha Qazi, and James Cussens. Clp(bn): Constraint logic programming for probabilistic knowledge. In *UAI*, pages 517–524, 2003.

[24] Adnan Darwiche. Modeling and Reasoning with Bayesian Networks. *Book*, page 562, 2009.

[25] Luc De Raedt. *Logical and Relational Learning*. Springer-Verlag, 2008.

[26] Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors. *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*. Springer, 2008.

[27] Arthur L. Delcher, K. A. Bratke, E. C. Powers, and S. L. Salzberg. Identifying bacterial genes and endosymbiont DNA with Glimmer. *Bioinformatics*, 23(6):673–679, 2007.

[28] Arthur L. Delcher, Douglas Harmon, Simon Kasif, Owen White, and Steven L. Salzberg. Improved microbial gene identification with GLIMMER. *Nucleic Acids Research*, 27(23):4636–4641, 1999.

[29] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.

[30] Thom Fruhwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.

[31] Zoubin Ghahramani and Michael I. Jordan. Factorial Hidden Markov Models. *Machine Learning*, 29(10):245–273, 1997.

[32] Roderic Guigó, Paul Flicek, Josep F. Abril, Alexandre Reymond, Julien Lagarde, France Denoeud, Stylianos Antonarakis, Michael Ashburner, Vladimir B. Bajic, Ewan Birney, Robert Castelo, Eduardo Eyras, Catherine Ucla, Thomas R. Gingeras, Jennifer Harrow, Tim Hubbard, Suzanna E. Lewis, and Martin G. Reese. EGASP: the human ENCODE Genome Annotation Assessment Project. *Genome Biology*, 7(Suppl 1):S2, 2006.

[33] Christian Theil Have and Henning Christiansen. Modeling Repeats in DNA Using Probabilistic Extended Regular Expressions. In Gemma Bel-Enquix, Veronica Dahl, and M. Dolores Jiménez López, editors, *Biology, Computation and Linguistics, New Interdisciplinary Paradigms*, volume 228, pages 55–70, Taragona, Spain, 2011. IOS Press.

[34] Kevin L. Howe, Tom Chothia, and Richard Durbin. GAZE: A Generic Framework for the Integration of Gene-Prediction Data by Dynamic Programming. *Genome Research*, 12(9):1418–1427, 2002.

[35] Manfred Jaeger. Relational Bayesian Networks. In Dan Geiger and Prakash Pundalik Shenoy, editors, *Association for Uncertainty in Artificial Intelligence*, pages 266–273. Morgan Kaufmann, 1997.

[36] Laura Kallmeyer. *Parsing Beyond Context-Free Grammars*. Springer, 2010.

[37] Angelika Kimmig. *A Probabilistic Prolog and its Applications*. PhD thesis, Katholieke Universiteit Leuven, 2010.

[38] Angelika Kimmig, Bart Demoen, Luc De Raedt, V.S. Costa, and R. Rocha. On the Implementation of the Probabilistic Logic Programming Language ProbLog. *Arxiv preprint arXiv:1006.4442*, 2010.

[39] Angelika Kimmig, Luc De Raedt, and Hannu Toivonen. Probabilistic explanation based learning. In *ECML*, pages 176–187, 2007.

[40] Anders Krogh. Using database matches with for HMMGene for automated gene detection in Drosophila. *Genome research*, 10(4):523–8, May 2000.

[41] Thomas Schou Larsen and Anders Krogh. EasyGene a prokaryotic gene finder that ranks ORFs by statistical significance. *BMC bioinformatics*, 4(1):21, 2003.

[42] Ole Torp Lassen. Biosequence Analysis in PRISM. In *Lecture Notes in Computer Science*, volume 5366, pages 809–810. Springer, 2008.

[43] Qian Liu, Aaron J. Mackey, David S Roos, and Fernando C N Pereira. Evigan: a hidden variable model for integrating gene evidence for eukaryotic gene prediction. *Bioinformatics*, 24(5):597–605, 2008.

[44] Alexander V. Lukashin and Mark Borodovsky. GeneMark.hmm: new solutions for gene finding. *Nucleic Acids Research*, 26(4):1107–1115, 1998.

[45] Stephen Muggleton. Stochastic Logic Programs. In L De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, volume 32, pages 254–264. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[46] Stephen Muggleton. Learning from Positive Data. In *Inductive Logic Programming Workshop*, pages 358–376, 1996.

[47] K. P. Murphy. *Dynamic Bayesian Networks : Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, 2002.

[48] K. P. Murphy. Dynamic bayesian networks. *Probabilistic graphical models*, 41(November):515–29, 2003.

[49] NCBI. National. Center for Biotechnology Information. http://www.ncbi.nlm.nih.gov/.

[50] Pernille Nielsen and Anders Krogh. Large-scale prokaryotic gene prediction and comparison to genome annotation. *Bioinformatics (Oxford, England)*, 21(24):4322–9, December 2005.

[51] Ulf Nilsson and Jan Maluszynski. *Logic, programming and prolog*. JOHN WILEY & SONS LTD, 2 edition, 1995.

[52] LoSt on the web. The lost project. http://lost.ruc.dk.

[53] Vladimir Pavlović, Ashutosh Garg, and Simon Kasif. A Bayesian framework for combining gene predictions. *Bioinformatics*, 18(1):19–27, 2002.

[54] Matthieu Petit and Henning Christiansen. In *5ème Journées Francophone de Programmation par Contraintes, JFPC*, pages 285–294, Orléans, France. http://www.univ-orleans.fr/evenements/jfpc/jfpc_2009_proceedings.pdf.

[55] David Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[56] David Poole. Abducing through negation as failure: Stable models within the independent choice logic. *The Journal of Logic Programming*, 44(1-3):5–35, 2000.

[57] Maria S. Poptsova and J. Peter Gogarten. Using comparative genome analysis to identify problems in annotated microbial genomes. *Microbiology*, 156(7):1909–1917, 2010.

[58] Simon C Potter, Laura Clarke, Val Curwen, Stephen Keenan, Emmanuel Mongin, Stephen M J Searle, Arne Stabenau, Roy Storey, and Michele Clamp. The Ensembl Analysis Pipeline. *Genome Research*, 14(5):934–941, 2004.

[59] Kim D. Pruitt, Tatiana Tatusova, William Klimke, and Donna R. Maglott. NCBI Reference Sequences: current status, policy and new initiatives. *Nucleic acids research*, 37(Database issue):D32–6, January 2009.

[60] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2009.

[61] Stuart Russell and Peter Norvig. *Artificial Intelligence - a modern approach*. Prentice-Hall, Inc., 1 edition, 1995.

[62] Stuart Russell and Peter Norvig. *Artificial Intelligence - a modern approach*. Prentice-Hall, Inc., 2 edition, 2003.

[63] Steven L. Salzberg, Arthur L. Delcher, Simon Kasif, and O White. Microbial gene identification using interpolated Markov models. *Nucleic acids research*, 26(2):544–8, January 1998.

[64] Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In Leon Sterling, editor, *ICLP*, pages 715–729. Springer, 1995.

[65] Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *J. Artif. Intell. Res. (JAIR)*, 15:391–454, 2001.

[66] 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.

[67] Taisuke Sato, Yoshitaka Kameya, and Kenichi Kurihara. *PRISM User's Manual*. 2010.

[68] D.B. Searls. *Artificial intelligence and molecular biology*, chapter The computational linguistics of biological sequences, pages 47–120. American Association for Artificial Intelligence, 1993.

[69] Claude E Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1998.

[70] Jay Shendure and Hanlee Ji. Next-generation DNA sequencing. *Nature biotechnology*, 26(10):1135–45, October 2008.

[71] Roger Staden. A computer program to search for tRNA. *Nucleic Acids Research*, 8(4):817–825, 1980.

[72] Leon Sterling and Ehud Y. Shapiro. *The Art of Prolog*. MIT Press, 1986.

[73] Leslie G. Valiant. General Context-Free Recognition in Less than Cubic Time. *Journal of Computer and System Sciences*, 10:308–315, 1975.

[74] Andrew Viterbi. Viterbi algorithm. *Scholarpedia*, 4(1):6246, 2009.

[75] Andrew S. Warren, Jeremy Archuleta, Wu-Chun Feng, and João Carlos Setubal. Missing genes in the annotation of prokaryotic genomes. *BMC bioinformatics*, 11:131, January 2010.

[76] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids. *Nature*, 171(4356):737–738, 1953.

[77] J. Xiong. *Essential Bioinformatics*. Cambridge, 2006.

[78] Lin Xu, Hong Chen, Xiaohua Hu, Rongmei Zhang, Ze Zhang, and Z W Luo. Average gene length is highly conserved in prokaryotes and eukaryotes and diverges only between the two kingdoms. *Molecular biology and evolution*, 23(6):1107–8, June 2006.

[79] Neng-Fa Zhou. B-Prolog web site, 1994–2011. http://www.probp.com/.

RECENT RESEARCH REPORTS