Chapter 1

Introduction

1.1 Motivation

All the inherited information that determines the physiology of an organism is encoded in a genome which is present in each cell of the organism. The genome of an organism typically consists of one or more molecules, each consisting of a linear succession of four **DNA** (deoxyribonucleic acid) bases symbolised by the four letters A, C, G and T. The complexity of an entire organism is thus encoded in a few long molecules of apparently striking simplicity.
Recent systematic genome sequencing efforts have determined the complete sequence of A, C, G and T letters for a number of organisms. Knowledge of the complete **DNA** sequence gives **us** the opportunity to study the genetics of organisms both at a fundamental molecular level and on a global scale. Since the first genome of a multi-cellular organism, the nematode ***Caenorhabditis elegans,*** was sequenced in 1998 [eSC98] comprising about 100 Mb (million bases), the genome sequence of the fruit fly ***Drosophila melanogaster*** (120 Mb) [ea00] and of the plant ***Arabidopsis thaliana*** (125 Mb) [Ini00] have been determined. We have good quality draft sequences of the human ***Homo sapiens*** [Con01] and the mouse ***Mus musculus*** genomes [Con02] (both around **3000** Mb) which will be completed in the near future. We are thus for the first time in the possession of the blueprints of several organisms, but without knowing how to understand the **DNA** text's contents.
The life of an organism depends on a variety of molecules that carry out specific tasks, one of the major groups being proteins. Each protein consists of a linear sequence of amino-acids which is encoded in a subsequence of the genome, called a gene. One of the most important challenges is to find the sections of the **DNA** which encode proteins, i.e. to find protein coding

genes, and to determine the amino-acid sequence of the encoded protein.
With the sequencing of the mouse genome soon to be finished, we *can* compare the human DNA sequence to that of the evolutionarily related mouse genome. By comparing the DNA sequences of two related organisms, we can not only study the large scale organisation of their genomes, but can also try to use pairs of related subsequences to predict genes. This is a promising approach because related organisms have similar proteins which are encoded by conserved genes in the DNA of the genomes. By finding and comparing subsequences which are conserved between two genomes we can try to predict protein coding genes.
The main goal of my work presented here is to develop a method for the comparative prediction of protein coding genes in pairs of related genomes. This task can be compared to the invention of a method for the automatic deciphering of the Rosetta stone. This stone contains one text in three different languages (Egyptian hieroglyphics, Demotic and Greek), see Figure **1.1,** and was carved in **196** BC in Egypt. When it was found in **1799,** only one of the three languages (Greek) was known. Manual comparative analysis of the three texts was completed in **1822** and led to the first understanding of both Egyptian hieroglyphics and Demotic. The task of comparatively deciphering the DNA texts of several related organisms is similar to the Rosetta stone deciphering in that we know that subsections of the texts are in close relation to ***each*** other, but it is complicated by the following:

- We do not have a DNA text which we completely understand, i.e. a text corresponding to the Greek text of the Rosetta stone does not exist in the DNA deciphering problem.
- The different sections of related DNA texts are not necessarily collinear or have a one-to-one correspondence. Figure **1.2** shows an example. There are **22** chromosomes plus the X and Y chromosomes in the human genome and **19** chromosomes plus the X and Y chromosomes in the mouse genome and the current estimate is that about **99 %** of the mouse genes have a corresponding human gene. We thus have to deal with rearrangements within the related DNA texts and cannot expect the contents of the texts (e.g. genes) to have a one-to-one correspondence.
- The DNA texts can be very long (around 3000 million A, C, G, T letters each for both the mouse and the human genome) and cannot be manually compared on a global scale and in a reproducible and efficient way.
- We do not know all the functional entities within DNA texts (like chapters, paragraphs,

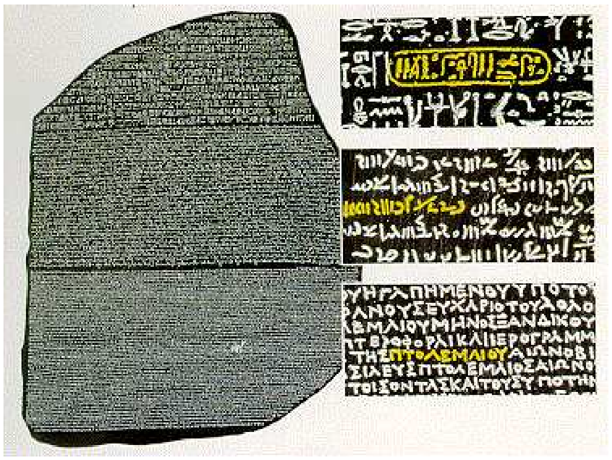

Figure 1.1: Photograph of the Rosetta stone. The upper text is written in Egyptian hieroglyphics, the text in the middle in Demotic and the lower text in Greek. The enlarged sections of the text show corresponding parts of the three texts. The highlighted word reads ‘Ptolemy’.

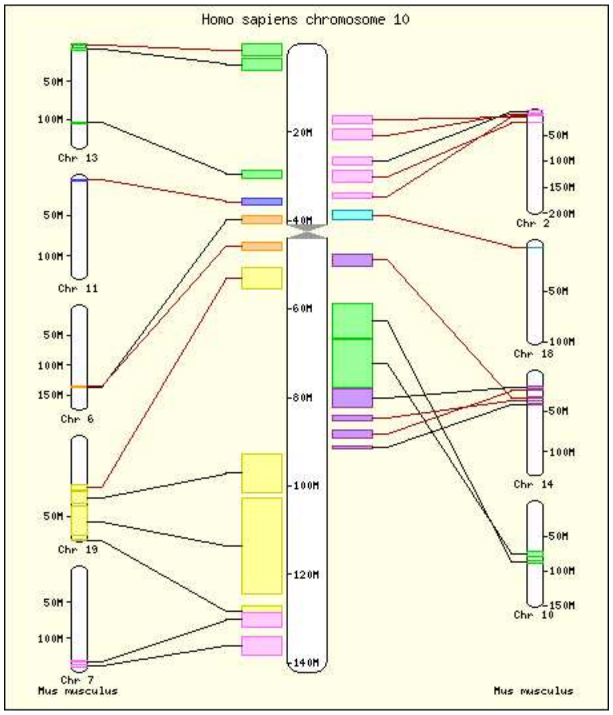

Figure 1.2: Large scale correspondence between human chromosome 10 (in the middle) and several mouse chromosomes (on the left and right) [Ens]. Note that most, but not all, of the human chromosome has a corresponding related region in the mouse genome. Also note that corresponding regions may appear in a rearranged region in the other organism (see human chromosome 10 and mouse chromosome 2) and that there is no collinearity between the human and mouse chromosomes at the large scale (see human chromosome 10 and mouse chromosomes 10 and 14).

```
...ctcagccttgtgtgagttgaggggaggtgtcacatccagctggagtcctttctaagcagc
   cacagcctgatcctcccacttcctcccccaagaaaacattgt~tgatggccataccc
   tgaggttctggtccaaatcggactttctatgaccttctgggtctctagtgaaaactaaag
   actcctctccagaaaaaaacatttggtttctaatgaggcctggaatcttattcttgacct
   ggggagcggaatccctttttgcagtactcccgggccctctgttggggcctccccttcctc
   tccagggtggagtcgaggaggcggggctgcgggcctccttatctctagagccggccctgg
   ctctctggcgcggggccccttagtccgggctttttgccATGGGGTCTCTGTTCCCTCTGT
   CGCTGCTG'ITRTMTGGCGGCCGCCTACCCGGGAGTTGGGAGCGCGCTGGGACGCCGGA
   CTAAGCGGGCGCAAAGCCCCAAGGGTAGCCCTCTCGCGCCCTCCGGGACCTCAGTGCCCT
   TCTGGGTGCGCATGAGCCCGGAGTTCGTGGCTGTGCAGCCGGGGAAGTCAGTGCAGCTCA
   ATTGCAGCAACAGCTGTCCCCAGCCGCAGAATTCCAGCCTCCGCACCCCGCTGCGGCAAG
   GCAAGACGCTCAGAGGGCCGGGTTGGGTGTCTTACCAGCTGCTCGACGTGAGGGCCTGGA
   GCTCCCTCGCGCACTGCCTCGTGACCTGCGCA~AAAACACGCTGGGCCAC~CCAGGA
   TCACCGCCTACAgtgagggacaggggctcggtcccggctggggtgaggggagggggctgg
   aagaggtgggggaagggtagttgacagtcgctctatagggagcgcccgcggacctcactc
   agaggctccccccttgccttagAACCGCCCCACAGCGTGATTTTGGAGCCTCCGGTCTTAA
   AGGGCAGGAAATACACTTTGCGCTGCCACGTGACGCAGGTGTTCCCGGTGGGCTACTTGG
   TGGTGACCCTGAGGCATGGAGCCGGGTCATCTATTCCGAAAGCCT~AGCGCTTCACCG
   GCCTGGATCTGGCCAACGTGACCTTGACCTACGAG~G~~TGGACCCCGCGACTTCT
   GGCAGCCCGTGATCTGCCACGCGCGCCTCAATCTCGACGGC~GGTGGTCCGCAACAGCT
   CGGCACCCATTACACTGATGCTCGgtgaggcacccctgtaaccctggggactaggaggaa
   gggggcagagagagttatgaccccgagagggcgcacagaccaagcgtgagctccacgcgg
   gtcgacagacctccctgtgttccgttcctaattctcgccttctgctcccagCTTGGAGCC
   CCGCGCCCACAGCTTTGGCCTCCGGTTCCATCGCTGCCCTTGTAGGGATCCTCCTCACTG
   TGGGCGCTGCGTACCTATGCAAGTGCCTAGCTATGAAGTCCCAGGCGtaaagggggatgt
   tctatgccggctgagcgagaaaaagaggaatatgaaacaatctggggaaatggccataca
   tggtggctgacgcctgtaatcccagcactttgggaggccgaggcaggagaatcgcttgag
   cccaggagttcgagaccagcctggacaacaacatagtgagaccccgtctatgcaaaaataca
   caaattagcctggtgtggtggcccgcacctgtggtcccagctacccgggaggctgagttg
   ggaggatcctttgagccctgaaagtcgaggttgcagtgagccttgatcgtgccactgcac
   tccagcctgggggacagagcacgaccctgtctccaaaaataaaataaaaataaaaataaa
   tattggcgggggaaccctctggaatcaataaaggcttccttaaccagcctctgtcctgtg
   acctaagggtccgcattactgcccttcttcggaggaactggtttgtttttgttgttgttg
   ttgtttttgcgatcactttctccaagttccttgtctccctgagggcacctg~ttcct
   cactcagggcccacctggggtcccgaagccccagactctgtgtatccccagcgggtgtca
   cagaaacctctccttctgctggccttatcgagtgggatcagcgcgggccggggagagcca
   cgggcaggggcggggtggggttcatggtatggctttcctgattggcgccgccgccaccac
   gcggcagctctgattggatgttaagtttcctatcccagccccaccttc~accctgtgct
   ttcctggaggccaaacaactgtggagcgagaactcatctccacgctg
   gagtgagaccacgaatggtggggaggggagggtcccacggacatattgagggacgtggat
   acgcagaagaggtatccatgtggtggcagccgggaaggggtgatcagatggtccacaggg
   aatatcacaaactcgaattctgacgatgttctggtagtcacccagccagatgagcgcatg
   gagttgggggtggggggtgtcaaagcttggggcccggaagcggagtcaaaagcatcaccc
   tcggtcccttgttctcgcgtggatgtcagggcccccacccaccgagcagaaggcggactc
   aggggcgctccagggtggctcgagctcacacacgctgagtagacacgtgcccgctgcacc
   ctgggtaaatacagacccggagccgagcggattctaatttagacgcccgcgaacgctgcg
   cgcacgcacacgtgtcctcggctcgctggcactttcgtcccgccccctccgtcgcgtgcg
   ggagctgacccggaggggtgcttagaggtatggctccgcggggtcaaaaggagaaggatc
   agtgagagaggcatccccacaccctccc...
```

Figure 1.3: A short subsequence of DNA from the human ***Homo*** sapiens genome comprising one protein coding gene which is evolutionarily related to the gene contained in the mouse Mus ***musculus*** sequence shown in Figure 1.4. The protein coding parts of the gene are highlighted by capitalisation. They correspond to the protein coding exons which are shown **as** hashed boxed in Figure **1.5.** The DNA sequence of the whole ***Homo*** sapiens genome would correspond to about one million pages.

```
...gagtgtcttgtgagtttgtgtacagtcatcacatcagttaggcaaagccctaaggactgc
   cgactcccataatgcctcagggttgtctggtaacctaaccctaactctgagtctgtggat
   caggttggtccccacccccacccccctttctttttttgagacaggttctctttgtggccatg
   gatgtcctgaaatctgctatgtggaatgggctggccttgacttcacaaagat~ccaac
   ctgtctcctgaatgctaggactaaatgacaaagccactgccatgtct~aaaatctacg
   ttagatagacagggtttcccagtgtagatcaggatggccttgaacttacagagatctgcc
   tccctgggagtgctgggatcaaaggcatgtgccatcaccaagcgttattttatttttttaa
   tttttaaagacttcttggggcttacgtaaaaactaaagagcaggtccagaactgtgcaat
   ggcttttggttgattgtagggtctgatgggaggggaggcaggtatcttcatcagggccgg
   ccgaggcccattctggggcgtggccagggtgccttcttatctcctgcggccagcctaaac
   tccctggcgttccgcccgcacttcagcgcgggctttgtgccATGGAGTCTGCCCTTCTGC
   TCCCGTCGCTTTTGCTGGTGGCTGCCTATCCGAGGGGTGGGAGCCCCCAGCAAGAGTGGA
   TGCAAAGTCCTCCCGCGCCTTCCGTGACCTCAGCACCI'TTCTGGGTGCGTCTTAATCCAG
   AGCTAGAGGCCGTGCCTCCCGGG~TCAGCGTG~AACTGCAGCCACAACTGCCCCC
   TGCCGGTGCATTCCAGCCTCGCACCCAACTGCGGCAGGGAAAGATAG~AATGGATCCG
   GCTGGGTATCTTACCAGCTACTGGATGTGAGGGCCTGGAATTCCAAGGTGCGCTGCGTCG
   TCACTTGCGCAGGAGAAACCCGAGAGGCCACCGCCAGGATCACTGCTTACAgtgagggag
   accggggctcaggccgggctggggtgaggggagaggggtggaggaagcggatagatggta
   attgctttaaggggtgcctgtgggccttatctctcttgccttagAACGGCCCAGAAGCGT
   GATCTTGGAGCCTCCGGTCCAGTGGGCCACAAGTACACTCTGCGAT~ATGTGACACA
   CGTGTTCCCAGTGGGATGTGGTGAGCCTGAGAAGAGGT~CGAGTGATTTATCA
   TGAAAGCCTGGAGCGCTTCACCGGTTCAGATTTGGCTAATGTCACI'TTGACCTACGTGAT
   GCGGGCCGGACTCAACGACC~~AGCCACTCACCTGCCATGCGCGCCTCAATCTCGA
   CGGGCTAGTGGTGCGCAGCAGCTCGGCACCTGTTATGTTGACAGTCCTCGgtgaggcatc
   ctgtaatcccagggaatgggtgcgggagaggggatgttgccactccaaggggggcctgcag
   aacaggcgtgggctccacgcttggcggtaacctcctcagacctcctagttcctgattttcactcc
   tgcccacagCTTTAAGCCCAGCCTCTATAGCCTTGGCCTCTACCTCCATCGCAACCCTGG
   TGGGGATCCTCCTGGCTGTGGGGGCTGTCTACGTGCGCAAGTACCTGGCTGTGCAGACTt
   agttatagatctgttttcgatgcctgacaagagggaggggaaaagaacttcagagtaatt
   aattcagagactcttattgaaacaataaagtcttcctcctcagtctctgccttacggttc
   ttggagaaagtggtttctttttttaaggtaccttacttttttccaaattccttacgtagggg
   ctgaagattagtagattagaggtagtactggaggaaacaacaccttgaaatttctccttc
   aaggccagcatggggtcctagaacccgagttcctctgcgtagagttttgttagctttatt
   tgtgcggggcagaaagactaaactgacctcccctccagggctgactcttggtatggcttt
   ttctgattggctccgctgatacaggccggagctctgattggaggctaagtttcccttctc
   ctccctccttttccactacggagcctgtgcgttactagagaaggccagcgggtggagcta
   gacctgattccccaaggttatcattaattgggggggggggggggaggtagaaacactcgag
   taggcggggccttcttcaagtagtagaggaagcggctaactagataggaaatctagcata
   gcaacaagttaagagatgattgttcaggccacgtgagctgtcacagacttgcttcctggc
   gttgtgcttgttgtctccgagtctggtatgtatgtagagagggatgtcaaagctggggtc
   aaagtgtccccagttgatcttttggtccagcgtgaattgcagaatctcgcactagttacc
   cagtagaggcggccacactcctggcgaggagggcgcagaagctctgctgagagactagac
   acacaacagcgttgtagacacattcccgctgcactctgggtaaataaagatcgggtgccg
   gagtcgactctaatttagaagcctgcgaacgctgcgcacacgcacacgtgtccgagtctt
   gctggcacttgatccccctcttccttcgccgcgtgcgcggag...
```

Figure 1.4: A short subsequence of DNA from the mouse *Mus musculus* genome comprising one protein coding gene. The protein coding parts of the gene are highlighted by capitalisation. They correspond to the protein coding exons which are shown as hashed boxed in Figure 1.5. The DNA sequence of the whole *Mus musculus* genome would correspond to about one million pages.

sentences and words) and how they structure the text hierarchically. We know for example of protein coding genes and promoters and other small entities, but do not know very much about how they are grouped into larger functional entities.

As is apparent from viewing only small pieces of DNA data such as those shown in Figure 1.3 and Figure 1.4, computational methods which can be applied to large amounts of data in a reproducible way have much potential for helping to unravel the text of genomes by proposing answers to biologically interesting questions which can be experimentally verified.
This introductory chapter provides the biological background, an overview of already existing methods for gene prediction and the theoretical background on which my work is built. Chapter 2 presents the pair HMM underlying DOUBLESCAN and PROJECTOR, two new methods which can be used for the comparative prediction of genes, as well as a new algorithm, called the Stepping Stone algorithm, by which genes can be predicted with essentially linear time and memory requirements, thus enabling large scale analyses. Chapter 3 demonstrates that DOUBLESCAN can be used to predict genes in mouse and human DNA. Chapter 4 presents a variant of DOUBLESCAN, called PROJECTOR, by which genes which are known in one organism can be used to find related genes in another related organism as exemplified on a set of mouse and human DNA sequences. Chapter 5 demonstrates that DOUBLESCAN and PROJECTOR can be easily adapted to analyse other pairs of related genomes by showing their performance for predicting genes in C. *elegans* and C. *briggsae* DNA sequences. Chapter 6 introduces a library of C++ classes by which large and complex projects such as DOUBLESCAN and PROJECTOR can be implemented in a short time.

1.2 Biological background

In eukaryotes, a subsequence of the genomic DNA is linked to its functional expression as a protein by a series of steps which can be roughly grouped into [HRS+87]:

- transcription of a DNA subsequence into an RNA (ribonucleic acid) sequence
- modification of the RNA sequence to produce a mature messenger RNA
- translation of the messenger RNA sequence into a protein sequence
- modification of the protein sequence

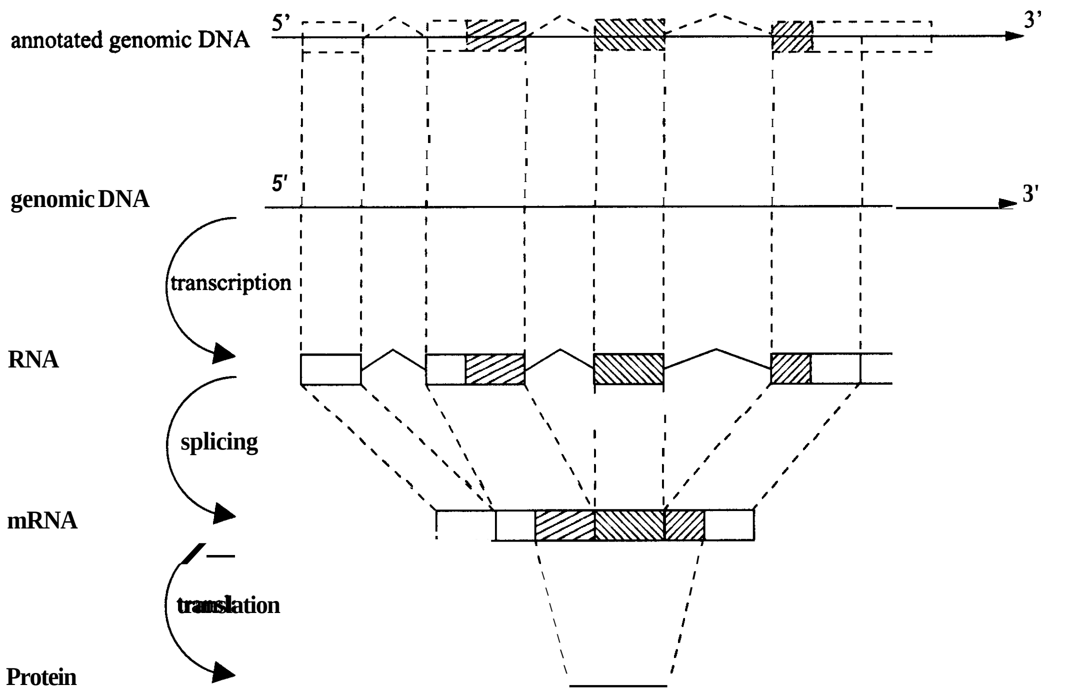

Figure 1.5: Schematic view of the steps by which a subsequence of the genomic DNA is linked to the amino-acid sequence of the protein it encodes. Each box (see boxes at RNA level) represents an exon and each kinked line an intron. The protein coding parts of each exon are hashed. See the text for a description of the processes.

See Figure 1.5 for a schematic view of the events in which the sequences involved are simplistically represented as linear molecules. A subsequence of the genomic DNA is transcribed into an RNA molecule by RNA polymerase after the genomic DNA has been prepared for transcription. The RNA molecule then undergoes a variety of modifications such as the modifications of its ends (as shown in Figure 1.5 for the 3' end) and splicing. During splicing small nuclear ribonucleoproteins (snRNPs) excise intronic sequences and join the exons into a shorter messenger RNA molecule (mRNA). This mRNA molecule is then transported from the nucleus to the cytoplasm where the continuous segment of exons is translated into the corresponding sequence of amino-acids by the ribosome and a variety of tRNA molecules. Depending on the final location of the protein, the amino-acid sequence may undergo modifications such as the cleavage of signal sequences. The protein coding part of a gene starts at the 5' end with a start codon and finishes on the 3' end before a stop codon. Splice sites are the 5' and 3' ends of introns.

The aim in *ab* initio gene prediction is to find genes and to infer their gene structures from a given DNA sequence of A, C, G and T letters only. The assignment of functional information to the DNA sequence is called annotation. By knowing the annotation of the DNA sequence i.e. the exon-intron structure of its genes as shown in Figure 1.5, we can directly infer the amino-acid sequence of the corresponding protein. In principle, it suffices to known the protein coding parts of the exons of a gene to derive the amino-acid sequence of the encoded protein. For the rest of this dissertation, the term gene refers to protein coding genes and the term exon to the protein coding part of an exon unless stated otherwise.

Traditionally, *ab* initio gene prediction methods for eukaryotes deal with one DNA sequence at a time. Methods for comparative gene *ab* initio gene prediction exploit the fact that related proteins have similar amino-acid sequences which are encoded in genes of similar exon-intron structure and that the exons of related genes are typically much more conserved than the introns which do not encode protein information.

As the method presented in this dissertation annotates two DNA sequences simultaneously, gene prediction methods which deal with only one sequence are only briefly reviewed and only those that are of relevance to this work are presented.

1.3 Existing non-comparative methods for *ab initio* gene prediction

There have been numerous studies with the aim to predict the intron and exon structure of eukaryotic genes given the DNA sequence as the only input information. Each of them typically consists of one or more programs which employ one or more methods to finally arrive at a prediction. The discussion is grouped by the methods employed rather than by the different ways in which they are combined into one program to emphasise the different underlying concepts.

1.3.1 Types of evidence

When trying to annotate a sequence of DNA we can make use of a variety of sequence signals which indicate the presence of functional elements or which mark a boundary between them. The principal measures used are:

Coding measures Exons and introns exhibit a different usage of nucleotide patterns. One statistically significant measure of difference found [CB86, JMCB90, FLS92, FT92] was that of relative frequencies of six nucleotide words, so-called hexamers.

Sequence signals Besides a compositional bias between exons and introns, their boundaries can be detected by certain sequence signals, as for example the acceptor and donor splice sites at the 5' and 3' sides of introns. Other signals include the translation initiation signal around the start codon (Kozak consensus [Koz81]), the translation terminal signal around the stop codon, the poly-adenylation signal and promoter sequences.
The **individual** statistical significance of any of these measures is not sufficient to reliably predict the exon and intron structure of a given DNA sequence [BEK91, CA96]. Only by combining several signals into a valid gene structure can we attempt to successfully predict genes.

1.3.2 Methods

This section concentrates on the description of those methods for integrating sequence signals which have turned out to be the most successful in ***ab* initio** gene prediction, namely neural networks, discriminants and hidden Markov models.

Besides these methods, other methods, such as rule-based methods (as used in the program GENEID [GKDS92]), linguistic methods (see for example the program GENLANG [DS94]) and decision trees (employed by the program MORGAN [SDFH97]) have been proposed to predict exons or genes.

Neural networks

Neural networks in gene prediction are typically used for combining signals from numerous sources, as for example from sequence motifs and nucleotide frequencies, into one score.
A neural network consists of an input layer of so-called neurons which accept the input values, i.e. the scores. The input signals propagate from the neurons of the input layer to the neurons of one or more layers of hidden neurons until the propagated signal finally reaches the output neuron. The final result depends on the architecture of the neural network as well as on the function with which each neuron merges several incoming scores into one outgoing score. These functions typically depend on multiple parameters which are given some initial values without knowing their optimal values. These parameters and even the architecture of the neural network can be adjusted by training it with a representative data set for which the correct outcome is known. The trained neural network can then be used on unknown data sets. Signal propagation in the neural network is unidirectional though the training of the parameters need not be. A general overview on neural networks can be found in [Bis95].
Neural networks are used in the GRAIL program [UM91] to identify exons which are later assembled into genes. The program GENEPARSER [SS93] exemplifies how neural networks can be used in conjunction with other methods to predict the intron and exon structure of a DNA sequence. The neural network is used to combine the scores of different sources of information such as codon usage, compositional complexity, length distributions, k-tuple frequencies and splice site signals into one score under the hypothesis that an exon or intron is found at a certain position in the DNA sequence. Dynamic programming is then used to assemble these potential exons and introns at different positions along the DNA sequence into a gene structure with a valid splicing pattern which maximises the overall score.

Discriminant methods

Linear discriminants are another approach to the classification of signal sequences that was used in the programs HEXON and FEX [SSL96]. Here an optimal separating plane is obtained between the true and false examples viewed as points in a multi-dimensional space, under the assumption that the true and false distributions are both gaussian with the same covariance matrix, but different means. The program MZEF [Zha97] employs quadratic discriminants [McL92] to predict independent internal coding exons in genomic DNA sequences. The characteristic features (e.g. splice site scores) of true and false internal exons are assumed to be described by two multinomial distributions in a multi-dimensional space which may have different means and different covariances. Quadratic discriminants can model the boundary between these two distributions and thus distinguish between true and false exons more effectively than linear discriminants as they are not limited to separating the two distributions by hyper-planes.

Hidden Markov models

Hidden Markov models (HMMs) are a mathematical method to linearly label a sequence with labels from a finite set of states. The states of the finite set can be defined to reflect our knowledge of the biological problem and classify the letters of the input sequence into mutually exclusive classes, as for example 'intron' and 'exon' and other labels used for an annotation.

A hidden Markov model can be imagined as a finite set of states which are connected by directional transitions. Each transition connects two states and has a transition probability associated with it. Each state has a predefined action, for example it reads one letter from the input sequence and thereby assigns the state's label to it. From that state one can pass to one of the states to which it is connected. By thus walking along a state path in the Markov model, the letters of the input sequence are successively labelled with state labels.

The following paragraphs give some definitions and explain Markov models by giving a simple example.

Definitions A ***Markov*** model or ***Markov*** chain associates every random variable of a discrete time stochastic process $x_1, x_2, \ldots$ with a state from a finite set of states. A Markov chain is said to be of order n if the probability of a transition from state s_i at time i to state s_{i+1} at time $i + 1$ depends only on the n previous states $(s_{i-n}, \ldots s_i)$. If the transition probabilities are independent of time, the Markov chain is said to be homogeneous, and inhomogeneous otherwise.

The notion of Markov chains can be extended by associating a probability distribution τ_s with every state s in order to model the time the Markov chain spends in this state. This type of Markov chain is called a semi Markov model [How71].

Another way to extend the notion of Markov chains is to define a so-called hidden Markov model *(HMM)* for which every step x_i of the Markov process consists itself **of** a stochastic process that generates the observed values y_i. Also this inner stochastic process can take values from another finite state space [BP66, Rab89]. If the underlying Markov process is a semi Markov Model, the model is called an explicit state duration hidden Markov model, generalised hidden Markov model or hidden semi Markov model (HSMM) [Rab89, KHRE96]. A text book on the application of Markov models in the context of biology is [DEKM98].

Example **of** a simple **Markov** model The above definitions can be illustrated by the example of a very simple Markov model which is entirely trivial, but which helps to make the distinction between a Markov model and a hidden Markov model clear.

As already mentioned, a Markov model can be imagined **as** a finite set of states which are connected by transitions. Each state corresponds to one of the four observable bases of the DNA alphabet (i.e. A, C, G and T). The states are connected by directional transitions which each have a transition probability associated with them. $t_x(y)$ denotes the transition probability for going from state x to state y. By reading the letters of an input DNA sequence $X = (x_1, x_2, \ldots, x_Z)$ of length Z, $x_j \in \{A, C, G, T\}$, the Markov model assigns the following probability to the sequence:

$$P(X) = \prod_{i=1}^{Z-1} t_{x_i}(x_{i+1})$$

As each state of this Markov model corresponds to one of the four possible observables A, C, G and T, the state path in this Markov model simply corresponds to the sequence **of** letters in the input DNA sequence.

Turning the **Markov** model into **a** hidden **Markov** model The above Markov model can be turned into a hidden Markov model by separating the states from the observables and by introducing emission probabilities e, for each state s. The emission probability of state s for reading letter x_j at position j in the sequence is denoted $e_s(x_j)$.

For a hidden Markov model, the above formula for the probability which the hidden Markov

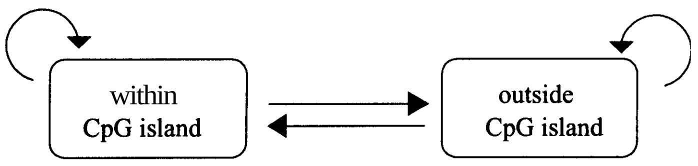

Figure **1.6:** Example of a simple HMM which classifies the letters of an input **DNA** sequence into those within and those outside CpG islands.

model assigns to an input **DNA** sequence $X = (x_1, 22, \ldots, x_Z)$ and a chosen state path $S = (s_1, s_2, \ldots, s_Z)$ of length Z then reads:

$$P(X,S) = e_{s_1}(x_1) \cdot \prod_{i=1}^{Z-1} t_{s_i}(s_{i+1}) \cdot e_{s_{i+1}}(x_{i+1})$$

The use of emission probabilities in hidden Markov models facilitates the definition of states which closely represent biologically motivated classes. To give an example, see Figure **1.6:** when searching a **DNA** sequence for CpG islands [Bir87], we can encounter situations where a C in the **DNA** sequence can either be a frequently occurring C within a CpG island (read by the state within *CpG* island and thereby labelled **C** within *CpG* island) or a C outside a CpG island (read by the state outside *CpG* island and labelled **C** outside *CpG* island) which we would expect to encounter only rarely. The emission probabilities of these two states can be defined to distinguish between these two classes, e.g. the state within *CpG* island might have a high emission probability for reading a C and the state outside *CpG* island have a lower emission probability for reading a C. CpG islands are typically several thousand bases long and are better modelled using first order emission probabilities $e_{\text{state}}(x_i|x_{i-1})$ so that the probability of reading letter x_i at position i in the sequence depends on the letter x_{i-1} at the previous position $i - 1$.

The action of states in HMMs can be extended to read zero, one or more letters from the input sequence.

Using an HMM to predict an annotation Once the states and transitions of a hidden Markov model have been defined to capture the features of the biological system which one wishes to describe and its emission and transition probabilities have been set, it can be used to assign a probability to a given input sequence and a chosen state path. In general, there exist a multitude of possible state paths and it is not clear a priori which state path to choose.

As the state path can be translated into an annotation of the input sequence, the aim is to select the state path that corresponds to the correct annotation of the sequence. The task is therefore to find a method which retrieves the desired state path for a given input sequence and a given HMM.

The above formula for $P(X, S)$ expresses the probability which the hidden Markov model assigns to a given input sequence X and a chosen state path S as function of the transition and emission probabilities encountered on the state path. If the emission and transition probabilities of the HMM have been chosen appropriately, we assume that the state path with the highest probability $P(X, S)$, denoted S_{opt}, corresponds to the correct annotation. The task is then to find this *optimal state path*, S_{opt}, which maximises $P(X, S)$. This optimal state path can be retrieved using the *Viterbi algorithm* [Vit67] and is therefore also called the *Viterbi path*. Once the optimal state path has been determined, its sequence of states can be translated into an annotation of the input sequence.

HMM based gene prediction programs The program GENSCAN [BK97] employs an explicit state duration HMM which models the length distribution of exons. It is capable of predicting complete, partial and multiple genes and simultaneously predicts genes on the forward and the reverse complemented strand of the input DNA sequence. GENSCAN's HMM has separate states for the exon of single exon genes and for initial, intermediate and terminal exons, as well as for a promoter, the 5' untranslated region, the 3' untranslated region and the poly-A signal. The HMM integrates information about several sequence signals such as splice sites, promoters, poly-A signals and start codons. GENSCAN'S parameters are chosen according to one of four GC contents intervals [DMG95, ConOl] in which the GC contents of the input sequence falls. Initially, GENSCAN was trained to predict human genes, but its performance at nucleotide or exon level on genes of rodent (mostly mouse and rat DNA sequences) and non-mammalian vertebrates (fish, amphibian, reptilian and avian DNA sequences) is not much lower than that for primate genes, see [Bur97, pp. 106–1071. GENSCAN is one of the reference programs for the *ab initio* prediction of human genes.

A program which combines an HMM with neural networks is GENIE [KHRE96]. The program HMMGENE [Kro97] is also based on an HMM, but uses series of identical states to model length distributions for exons whereas DOUBLESCAN employs an explicit state duration HMM. For a given input sequence, HMMGENE reports the best labeled state path using a heuristic method (N-best method) rather than the most likely state path using the exact Viterbi algorithm as is done by GENSCAN.

1.3.3 Summary

The methods presented above can be grouped in two main groups: methods which can have a probabilistic interpretation and those which cannot. Those based on Markov models are the most amenable to a probabilistic interpretation. With some effort, rule-based methods, linguistic methods, decision trees and discriminants can also be provided with a probabilistic framework, see [SH94]. Neural networks lack this feature as probability tags attached to the input cannot be propagated to the output. This lack of statistical accessibility does not mean that they have an inferior performance with respect to other methods, but it limits the amount we can learn about how they produce results, how they can be trained and why they may fail to perform well.

Some features of the above methods, for example rule-based methods or decision trees, can be captured by hidden Markov models. The advantage of the latter method is that it can simultaneously work on splice sites in a way a decision tree method might do, and at the same time keep track of more global features such as the exon phase. If a hidden Markov model is set up correctly, it will by *definition* retrieve a valid state path. There is, for example, no need to go through a set of single exons and to decide how to combine them into one gene, a partial gene, several genes or maybe even no gene at all.

These are the main reasons why we chose to work with Markov models. Comparisons of the performance of different gene predicion program on a variety of data sets can be found in [BG96, Cla97, RHH$^+$00].

1.4 Existing comparative methods

Research in the area of ***ab* initio** prediction of genes has so far focused on methods that take one DNA sequence and predict its gene structure, e.g. [BK97], and comparative gene prediction methods have only recently started to emerge. They use the same types of evidence as non-comparative methods, see Section **1.3.1,** together with similarity information from evolutionarily conserved subsequences and gene structures. The following paragraphs present the different methods which are used to combine these types of evidence into a prediction.

1.4.1 Conservation detection methods

Many of the earlier comparative methods do not aim to identify functional elements such as exons or entire gene structures, but only report subsequences which are conserved between two input DNA sequences without explicitly assigning a functional annotation to them.

Dot plots This method is based on the simple idea that two sequences can be compared by drawing one sequence along one axis and the other sequence along the other axis of a two-dimensional matrix and by assigning a 1 to a matrix element if the two corresponding letters of the sequences match, and a 0 where they do not match. This gives rise to a two dimensional matrix with 1s and 0s. Two identical subsequences give rise to a diagonal of 1s, whereas nonmatching subsequences correspond to areas with randomly distributed 1s. This is the basic principle upon which dot-plots are based. These plots can be refined by averaging over a selected diagonal and by applying some threshold value as done in the DOTTER program [SD96].
Dot plots do not predict a functional annotation as the underlying method does not know about exons, introns and valid gene structures.

Percent identity plots In this method, a gapped alignment is made between two sequences, say A and B. The percent identity plot is made by showing one of the two sequences, say sequence A, along the horizontal axis with the vertical axis showing how similar this part of sequence A is to the section of B which this is aligned to. Conserved regions show a high value of percent identity, non-conserved regions a low value.
A program called PIP was used in [OMM+97] to gain a first overview of the level of similarity between two DNA sequences. The authors refined their analysis by searching for gapped alignments using the SIM program [HHM90] in which the user can specify the penalty for a non-match and the two parameters for affine gap-penalties. These alignments were then transformed into precent identity plots relative to the positions in one of the two sequences. Similarly to dot plots, also percent identity plots do not give a functional annotation and do not predict genes.

Block aligner A pair hidden Markov model called DBA ('DNA block aligner') was introduced in [JBD99] to divide **DNA** subsequences into segments of different levels **of** percent identity. However, its states do not capture the different types of conservation between protein coding and non coding subsequences. DBA thus does not try to identify exons, introns or gene structures and was used in [JBD99] to study non protein coding **DNA** sequences in orthologous mouse and human gene pairs.

1.4.2 Methods for comparative functional prediction

The following methods have all emerged in the last few years and aim to make use of comparative information in two **DNA** sequences to predict functional elements such as exons or entire gene structures (refer to Figure **1.7** for an overview).

Prediction of protein coding subsequences The first attempt towards a comparative prediction of pairs of exons in two evolutionarily related **DNA** sequences was made in [KZOO] by introducing the program **WABA** ('wobbleaware bulk aligner'). The underlying pair hidden Markov model (pair HMM, see Section **1.5.1** for an introduction) can distinguish between the different types of conservation between conserved protein coding and non coding **DNA** subsequences. It identifies and aligns subsequences which may be protein coding. However, as the pair HMM neither includes special states for splice sites nor uses scores from a splice site prediction program, the identification of the exact exon boundaries is not attempted.

Incorporating similarity information into non-comparative hidden Markov models Cross species similary can be incorporated into non-comparative methods such as hidden Markov models which operate on one **DNA** sequence only. [KFDB01] proposed an extension **of** the **GENSCAN** program [BK97], called **TWINSCAN,** which integrates cross-species similarity at **DNA** level into the probabilities of a non-comparative model. In the first step, a local alignment is generated between the target sequence (which is the **DNA** sequence to be annotated) and the informant sequence (which is a **DNA** sequence which is similar to the target sequence). This local alignment is then converted into a conservation sequence which indicates for every nucleotide in the target sequence one of three possible levels of conservation. Using the target sequence of **DNA** letters and the conservation sequence, the state path which maximises the joint probability of observing both the nucleotide and the conservation sequence is derived using the same optimisation algorithm as in **GENSCAN.** This joint probability is the product of the **DNA** sequence's probability and the conservation sequence's probability. The latter is calculated according to a conservation model which is defined for every state in the HMM of

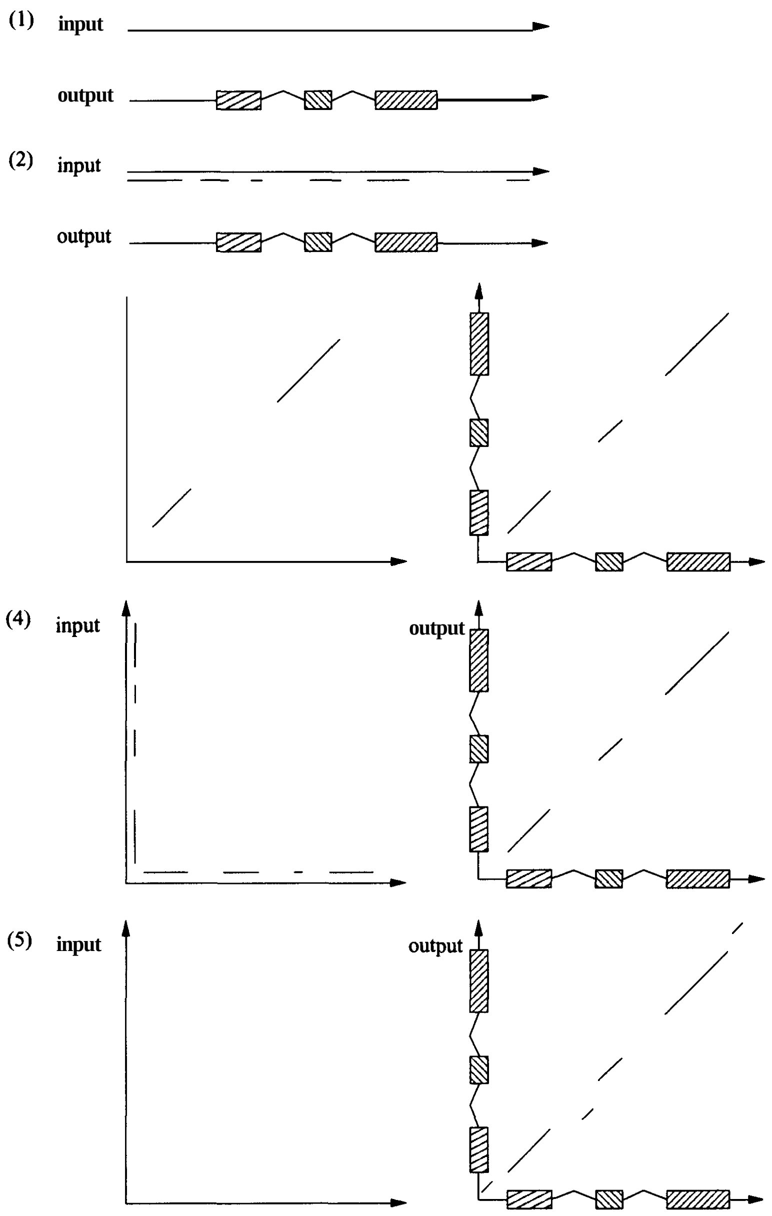

Figure 1.7: Overview over different types of ab initio gene prediction methods: **(1)** non-comparative gene prediction (e.g. **Genscan**), (2) non-comparative gene prediction which integrates homology information from a local alignment (e.g. **Twinscan**), (3) comparative gene prediction which is based on a global alignment (e.g. **Glass** and **Rosetta**), **(4)** comparative gene prediction which is based on a local alignment (**e.g. CEM** and **SGP-1**) and **(5)** comparative gene prediction where both alignment and genes are simultaneously predicted (**Doublescan** used with the Hirschberg algorithm). Refer to the text for a detailed description of the methods.

GENSCAN and which is based on fifth order Markov chains. TWINSCAN is not symmetric in the two input sequences and typically uses repeat-masked input sequences.

Comparative gene prediction in both sequences using a multi-step approach Any method for comparative ***ab* initio** gene prediction has to solve two problems: that of aligning the two input sequences and that of predicting gene structures for each of the two input sequences. One can try to solve these two problemes simultaneously (as is attempted in this dissertation, *see* Chapter **2,** Chapter **3** and Chapter ***5)***, but one can also try to solve them with some level of independence.
The latter approach was taken in the following studies:
[BPM+00] predicts gene structures in a two step approach by first globally aligning the two input DNA sequences using the program GLASS and by then identifying coding exons in both sequences and by merging them into identical gene structures using the program ROSETTA. The gene structures in the two sequences are assumed to have the same number of exons. The program does not deal with the two strands of each sequence simultaneously, but generates two independent gene predictions, one for each strand. The program works throughout with repeat-masked sequences which are used both for generating the global alignment and also the final gene structures.
Another multi-step program, called CEM ('conserved exon method'), is presented in [BH00]. In the first step, a local alignment of the two repeat-masked input sequences is generated using one of the existing programs. The next steps are executed for every match of the local alignment separately: a set of putative conserved exons is identified for every sequence separately. Next, only those pairs of putative conserved exons are retained that contain the match. The optimal alignment between the start point of each exon pair and the midpoint of the match, as well as that between the midpoint of the match and the end point of each exon pair is calculated using full dynamic programming. These alignments are converted into a set of alignments between every start and every end point which each have the score associated to it that was calculated in the dynamic programming. For every match of the initially generated local alignment, we then have a set of n-tuples each consisting of a start and end point, an alignment between them and a score. Complete gene structures are then built from this set of all n-tuples using dynamic programming with the assumption that the correct orthologous gene structures have the highest overall alignment score. CEM is capable

of predicting partial, complete and multiple genes. It predicts genes on both strands by running once on the forward and once on the reverse complemented strand and then merging the results into one set of genes which can lie on both strands. The prediction steps in CEM rely very much on the matches returned by the local alignment. If an exon pair is not hit by a match in the local alignment, it will be missing in the predicted genes. Similar genes are assumed to have the same number of exons.

Another example for the multi-step approach to comparative gene prediction is introduced in a program called SGP-1 ('syntenic gene prediction') [WGJMOG01]. In the first step, a local pairwise alignment is computed with one of the available programs. The matches of the local alignment may then be post-processed to reduce noice, if desired. In the second step (which is completely independent of the first step), a list of potential exons is generated for each of the two sequences separately. In the third step, the results of the first two steps are merged by retaining only those exons that are compatible with the alignment. This generates pairs of potential exons. In the fourth step, each exon pair is a assigned a score which is the sum of a similarity score and a sequence signal score. Finally, the list of exons is assembled into gene structures for each sequence independently. SGP-1 can deal with genes on both strands as well as with the partial and multiple genes. Further, as it is based on a local alignment of the two sequences, the genes do not have to appear collinearly within the two sequences. As the gene structures within each sequence are assembled independently of the other sequence, a one-to-one relationship between the genes in the two sequences or a one-to-one correspondence between the exons of two related gene structures are not automatically guaranteed. SGP-1 relies on the initial local alignments only for the definition of potential exons, but does not use similarity information to predict similar gene structures in the two sequences simultaneously (though this is what is likely to happen effectively if the sequences are well conserved and the local alignments coincide with the global alignment). There are thus very few steps involved in SGP-1 which depend quadratically on the length of the input sequence and which are thus time and memory consuming (a problem which GLASS and ROSETTA face to some extent and which limits the use of the fully pair HMM based programs DBA and WABA to rather short sequences). On the other hand, the similarity information is there and it should be possible to make good use of it in the simultaneous prediction of gene structures in the two sequences.

***Ab* initio comparative gene prediction using pair HMMs** Pair hidden Markov models (pair HMMs) are the natural generalisation of hidden Markov models (see Section 1.3.2) to two input sequences. They provide a fully probabilistic framework and the pair HMM's states, transitions and parameters have an intuitive interpretation. Pair HMMs are introduced in Section 1.5.1. As the mathematical concept of Markov models was very successfully applied to non-comparative ab initio gene prediction as shown by the program GENSCAN, see Section 1.3.2, it is tempting to try to use pair HMMs for the comparative *ab* initio prediction of genes.

[NGM01] present a method called PRO-GEN which they evaluate on a set of human-mouse, human-Xenopus and human-Drosophila *melanogaster* gene pairs. The underlying pair HMM can deal also with pairs of genes that are related by events of exon-fusion or exon-splitting, but it assumes each of the two input DNA sequences to contain exactly one complete gene. It predicts genes only in the input sequences, but not simultaneously in their corresponding reverse complemented strands. In a first step, potential splice sites and translation start and end sites are predicted for each of the two input DNA sequences separately. The Viterbi matrix is then calculated taking into into account the constraints imposed by the potential splice sites and translation start and end sites. The Viterbi algorithm is then used to derive the optimally scoring state path through the pair HMM with memory and time requirements which depend quadratically on the length of the input sequence. Pairs of codons within exons are scored using scores from the PAM120 matrix, whereas introns are not scored on a nucleotide by nucleotide basis, but rather by a fixed constant which is independent of the intron's length. A prediction generated by PRO-GEN consists of a complete gene in each DNA sequence as well as an alignment between corresponding exons. Note that the program does not predict conserved subsequences within introns or intergenic regions.

The pair HMM presented in this dissertation has been accepted for publication [MD02]. Recently, similar strategies were proposed by other authors [PAC01], but no implementation and evaluation has yet been published.

1.4.3 Summary

The comparative methods above can be subdivided into those that try to predict functional elements and those that do not. The latter provide some sort of alignment between two input DNA sequences which subdivides every sequence into subsequences of different levels of conservation without explicitly giving them a functional prediction. The findings in [JBD99] suggest that the length and percent identity of these conserved subsequences are generally not sufficient to allow their reliable functional annotation.

In order to attempt the comparative prediction of genes, sequence signals such as splice sites and start codons have to be integrated into the prediction process and single functional elements such as potential exons have to be grouped into valid gene structures. As mentioned earlier, any method for comparative *ab initio* gene prediction has to solve both an alignment and a gene prediction problem. These two problems can be solved simultaneously or sequentially, with some independence (see Figure 1.7).

Except for the method presented in [NGM01] (which assumes the presence of one complete gene in each input DNA sequence), the method proposed in [PAC01] and the method presented in this dissertation, the methods developed so far align and predict gene structures sequentially. This approach has the advantage that the time and memory requirements of most steps in the prediction process scale linearly with the length of the input DNA sequence as they are applied to each sequence independently. The multi-step methods for comparative *ab initio* gene prediction are therefore naturally well suited for applications on large DNA sequences. However, as the final gene prediction steps rely on the initial local or global alignment between the two input sequences, errors in the initial alignment may propagate to the gene prediction step which then has difficulties correcting for them. These methods thus assume that a fairly accurate local or global alignment can be made between the two input sequences. Furthermore, the prediction of the final gene structures in the two sequences is either done independently and thus does not make maximal use of the similarity information (as e.g. in the program SGP-1) or is done in very close dependence (as e.g. in the program CEM) which is probably best suited for pairs of closely related genes.

As opposed to the multi-step methods which try to solve the alignment and gene prediction sequentially, pair HMM based methods are suited to solve both in one step. As is shown in this work (see Chapter 2), the states and transitions of a pair HMM can be set up to simultaneously align the two input DNA sequences and to predict gene structures in both of the two sequences. The aim is to thereby obtain both, improved gene predictions and an improved global alignment which should also highlight conserved subsequences of yet unknown function, see (5) in Figure 1.7. The mathematical concept of pair HMMs can be used in a fully probabilistic way and sequence signals such as splice site scores and start codon scores can be fully integrated.

Depending on the definition of states, the pair HMM can also be set up to be able to align more diverged pairs of genes which are related by events of exon-fusion or exon-splitting. The main advantage of pair HMM based methods is that the gene prediction process is not separated from the alignment process and the similarity information between the sequences is fully used to aid the gene prediction process and vice versa. The heuristical ideas and assumptions used in the multi-step methods which may impose unjustified prejudices and restrictions on the gene finding procedure, are essentially not needed within pair HMMs and the prediction process relies only on a few very basic assumptions. However, the integrated alignment and gene prediction approach has the disadvantage that the time and memory requirements of the prediction process scale quadratically with the length of the input sequence which limits the applicability to rather short DNA sequences or makes the implementation technically challenging. We solve this problem by introducing the Stepping Stone algorithm whose memory and time requirements scale linearly with the length of the input sequence, see Chapter **2.**

It remains to be seen how well multi-step methods perform in comparison to each other and to pair HMM based methods, how each method performs on more diverged pairs of genes and how readily it can be adapted to successfully analyse other pairs of genomes. It will be crucial to see how the performance of each method scales when going from nucleotide level to gene level **as** this should be a good indicator of how well and in which way similarity information is utilised within each method (except for TWINSCAN and PRO-GEN, the gene level performance of the above mentioned comparative methods is not reported).

1.5 Theoretical background

Traditionally, ***ab*** initio gene prediction deals with one DNA sequence at a time. Among the most successful methods are hidden Markov models **as** exemplified by GENSCAN [BK97] and HMMGENE [Kro97]. In order to extend gene prediction to work on two DNA sequences simultaneously, we employ an extension of hidden Markov models, called pair hidden Markov models (pair HMMs) [DEKM98, KZOO].

1.5.1 Pair hidden Markov models

In analogy to the previously introduced Markov models, see Section 1.3.2, the definition of a state can be extended to deal with two sequences instead of only one sequence. All previously given definitions and remarks which were made for variants of Markov models in Section 1.3.2 also apply to pair hidden Markov models.

The difference between a pair HMM and an HMM is that a pair HMM deals with two input sequences instead of only one. The states of a pair HMM read letters from only one of the two input sequences or from both of them. As for HMMs, a state of a pair HMM assigns an emission probability to the letters it reads. The pair HMM then passes to one of the states to which the current state is connected by directed transitions and assigns a transition probability to this action. This procedure is repeated until all letters of both sequences have been read. The sequence of states passed through is called the state path.

Each state assigns labels to the letters it reads, as for example 'intron' or 'exon'. A state path can therefore be translated into annotations for both DNA sequences.

1.5.2 Alignment algorithms

Once the transition and emission probabilities of the pair HMM have been specified, the pair HMM can be used to predict an annotation for the two input DNA sequences by finding the optimal state path, S_{opt}. To any chosen state path S and a given pair of sequences X and Y, the pair HMM assigns the following probability:

$$P(X,Y,S) = e_{s_1}(k_1,p_1) \cdot \prod_{i=1}^{Z-1} t_{s_i}(s_{i+1}) \cdot e_{s_{i+1}}(k_{i+1},p_{i+1})$$

The sequence of states encountered on the state path is S $= (s_1, s_2, \ldots, s_Z)$, Z being the length of the chosen state path. $t_{s_i}(s_{i+1})$ is the transition probability to go from the i-th state s_i to the i $+$1-th state s_{i+1}. $e_s(k,p)$ is the emission probability of state s to read $\Delta_x(s)$ letters from sequence X, namely letters $x_{k-\Delta_x(s)}, \ldots, x_{k-1}$, and to read $\Delta_y(s)$ letters from sequences Y, namely letters $y_{p-\Delta_y(s)}, \ldots, y_{p-1}$. After the i-th step in the state path, we are therefore in state s_{i+1} at position k_{i+1} in sequence X and at position p_{i+1} in sequence Y. At the end of the state path, i.e. after Z steps in the pair HMM, all letters of the two sequences have been read.

As with a single sequence HMM, it is clear that there is a multitude of possible state paths for a given pair HMM and a given pair of input sequences. The aim is to find the state path which corresponds to a correct annotation for both sequences. The assumption is that, with appropriately chosen emission and transition probabilities, the state path with the highest probability $P(X, Y, S)$, denoted S_{opt}, corresponds to a correct annotation. The task is then to find this optimal state path, S_{opt}, that maximises $P(X, Y, S)$.

The basic method to retrieve the optimal state path is the Viterbi algorithm [Vit67]. We also introduce here the Hirschberg algorithm [Hir75] which finds the optimal state path with linear memory requirements.

The Viterbi algorithm

The optimal state path can be found using the Viterbi algorithm [Vit67]. This algorithm solves the optimisation problem in two steps. In the first step, the elements of a three dimensional matrix, the ***Viterbi matrix,*** are iteratively calculated. In the second step, a traceback process through the matrix retrieves the optimal state path.

Let N be the number of states and T the number of transitions in the pair HMM and L_x and L_y the lengths of the two input sequences X and Y, respectively. The value of each element in the Viterbi matrix, denoted $v(s, i, j)$, corresponds to the probability of a state path which ends in state s and which has so far read i letters from sequence X and j letters from sequence Y. By definition, every state path starts in the ***begin*** state, $s = 0$, and finishes in the ***end*** state, $s = N - 1$.

The elements of the Viterbi matrix are calculated as follows:

- Initialisation step:

  Set $v(0, 0, 0) = 1$ and all other $v(s, i, j) = 0$. This forces every state path to start in the ***begin*** state, $s = 0$.

- Recurrence relation:

  The $v(s, i, j)$ are iteratively calculated by looping over all $i \in \{1, \ldots, L_x\}$, all $j \in \{1, \ldots, L_y\}$ and all states $s \in \{1, \ldots, N - 2\}$ (the ***begin*** state, $s = 0$, and the ***end*** state, $s = N - 1$, need not be considered as they are only used at the start and end of each state path):

$$v(s,i,j) = \max_{s' \in \{1,\dots,N-2\}} \{v(s', i - \Delta_x(s), j - \Delta_y(s)) \cdot t_{s'}(s) \cdot e_s(i,j)\}$$

where

- $t_{s'}(s)$ is the transition probability to go from state s' to the state s.
- $e_s(i,j)$ is the emission probability of state s to read $\Delta_x(s)$ letters from sequence X , namely letters $x_{i-\Delta_x(s)}, \dots, x_{i-1}$, and to read $\Delta_y(s)$ letters from sequences $\boldsymbol{Y}$, namely letters $y_{j-\Delta_y(s)}, \dots, y_{j-1}$.
- Note that instead of maximising over all states $s' \in \{1, \dots, N-2)$ only those states s' for which a transition to s exists have to be considered.

- Termination step:

  The constraint that every state path has to end in the ***end*** state, s $= N - 1$, is implemented by setting

  $v(N-1, L_x, L_y) = \max_{s' \in \{1,\dots,N-2\}} \{v(s', L_x - \Delta_x(s), L_y - \Delta_y(s)) \cdot t_{s'}(N-1)\}$

  This probability can be shown [Vit67] to be equal to the probability of the optimal state path, S_{opt}.

At this state, the probability of the optimal state path is known, but the path itself has still to be retrieved. Once the elements of the Viterbi matrix have been calculated, the optimal state path, S_{opt}, is retrieved by starting at the matrix element $v(N-1, L_x, L_y)$ whose value is equal to $P(X, \mathbf{Y}, S_{opt})$ and by recursively determining the state from which the maximum at the current state was derived. Using this traceback method, the sequence of states of the optimal state path is retrieved. The annotations of the two DNA sequences as well as the conserved subsequences can be deduced from this state path.

For a pair HMM with N states and T transitions and two sequences of length L_x and L_y, respectively, the memory requirement for the Viterbi algorithm is of order $O(N \cdot L_x \cdot L_y)$, as this is the number of elements in the Viterbi matrix. The time requirement is of order $O(T \cdot L_x \cdot L_y)$, which is essentially the time consumed to calculate the elements of the Viterbi matrix.

It is clear that the quadratic dependency on the sequence length imposes serious restrictions on the applicability of the Viterbi algorithm on long sequences. For example, two sequences of 10^3 base pairs length and a pair HMM with 50 states would need about 400 MB memory (numbers saved in double format) to save the Viterbi matrix. The same pair HMM used on two sequences of 10^4 base pairs length would need a hundred times more memory and time to complete the calculation of the Viterbi matrix.

The Hirschberg algorithm

The dependency of the Viterbi's memory requirement on the product $L_x \cdot L_y$ imposes a serious constraint on the analysis of long sequences. The Hirschberg algorithm [Hir75] linearises the memory requirement while still retrieving the optimal state path.

The key idea is to make use of the following underlying symmetry: instead of starting the calculation at the start of the two sequences, i.e. sequence positions (x_1, y_1), we may as well start it at their ends, (x_{L_x}, y_{L_y}). This can be done by using a mirrored model which is created from the original pair HMM by reversing the directions of all arrows and by permuting the ***begin*** and ***end*** state with respect to the original pair HMM. This reversed pair HMM does not admit a probability interpretation any more because the probabilities of the transitions emerging from each state do no longer add up to one (instead, the probabilities of the transitions leading into each state add up to one). In the following, this model is called the ***mirror model.***

The Hirschberg algorithm divides the Viterbi matrix, see (1) in Figure **1.8,** into two halves which can each be calculated independently. One sub-matrix is calculated using the pair HMM starting at (x_1, y_1) and proceeding towards higher values of the sequence index i, the other sub-matrix is calculated using the mirror model starting at (x_{L_x}, y_{L_y}) and proceeding towards lower values of the sequence index i, see (2). Instead of storing the whole Viterbi matrix, only the values in a narrow strip like volume are stored because only these are needed to continue the calculation, see the hatched areas in (2). The minimum strip width is equal to the maximum number of letters which are read from a sequence by a state in the pair HMM plus one, to store the row of new values. The process is stopped when the two strips overlap, see **(3).** The probability of the optimal state path, $P(X, Y, S_{opt})$, is then found by multiplying the appropriate values in the two strips and by searching for their maximum which is equal to $P(X, Y, S_{opt})$. We then not only the know the probability of the optimal state path, S_{opt}, but also the coordinates (s, i, j) where the optimal state path crosses the two superimposed strips, see **(4).** The same procedure is then applied to the two emerging

sub-matrices whose boundaries are now known, one with sequence coordinates from (x_1, y_1) to (x_i, y_j) and the other one from (x_i, y_j) to (x_{L_x}, y_{L_y}), see (5) and (6), until the adjacent coordinates of the optimal state path, (7), are at most separated by a sub-matrix of some predefined maximum size, (8). These coordinates are then used as boundary conditions to run the Viterbi algorithm separately on all the small sub-matrices. In the end, the state paths of the small sub-matrices are concatenated into the optimal state path from start state $s = 0$ at (x_1, y_1) to the end state $s = N - 1$ at (x_{L_x}, y_{L_y}).
Using the Hirschberg algorithm, the memory requirement reduces to $O(N \cdot min\{L_x, L_y\})$. As each iteration halves the volume of the matrices that have to be calculated, the time used by the Hirschberg algorithm is at most twice the time used by the Viterbi algorithm, i.e. still of order $O(T \cdot L_x . L_y)$.
The benefits of the Hirschberg algorithm are:

- The memory requirement of the Viterbi algorithm can be reduced to $\mathcal{O}(N \cdot min\{L_x, L_y\})$, i.e. the memory required to save the two strips which each have length $min\{L_x, L_y\}$, minimal width and height N.
- As all sub-matrices have known boundary conditions, they can be calculated independently, possibly in parallel and on several computers.

To summarise, the Hirschberg algorithm finds the optimal state path and the optimal score with a memory requirement which scales only linearly with the sequence length and in a time at most twice the time needed by the Viterbi algorithm (as each iteration halves the area which has to be calculated, the time used by the Hirschberg algorithm is at most $t + t/2 + t/4 \ldots = 2 \cdot t$, i.e. twice the time t taken by the Viterbi algorithm). However, this time requirement of the Viterbi algorithm scales with $L_x \cdot L_y$ and still imposes a serious constraint on the analysis of long DNA sequences. In Chapter 2, we introduce a new algorithm which reduces both time and memory requirements to effectively linear dependence on the sequence length.

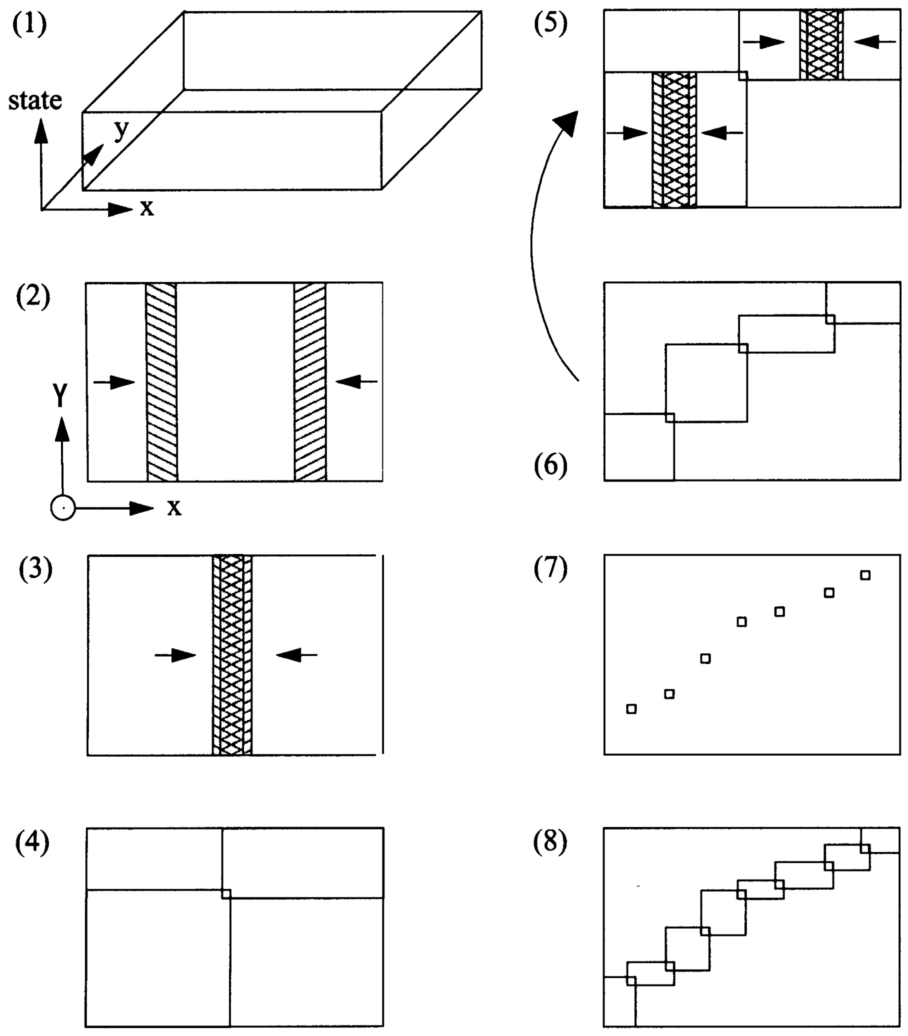

Figure 1.8: The Hirschberg algorithm. (1) shows the three-dimensional Viterbi matrix. (2) to (8) show only its two-dimensional projection onto the plane spanned by the two sequences *X* and *Y*. See text for details.