

COMENIUS UNIVERSITY, BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND
INFORMATICS

**A New Algorithm for Using
External Information in Gene Finding**

MASTER'S THESIS

Bc. Marcel Kucharík

Bratislava, 2011



DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
COMENIUS UNIVERSITY, BRATISLAVA

A NEW ALGORITHM FOR USING
EXTERNAL INFORMATION IN GENE FINDING

(Master's Thesis)

BC. MARCEL KUCHARÍK

(Code: 195679d5-94ce-43c1-a505-b79068bb4708)

Branch of study: 9.2.1 Computer Science
Study programme: Computer Science

Supervisor: Mgr. Bronislava Brejová, PhD

Bratislava, 2011



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky


ZADANIE ZÁVEREČNEJ PRÁCE

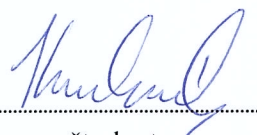
Meno a priezvisko študenta: Bc. Marcel Kucharík
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st.,
denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický

Názov: A New Algorithm for Using External Information in Gene Finding
Cieľ: Accuracy of computational gene finding can be increased by using additional information from biological experiments. The goal of the thesis is to design and implement a new algorithm for using complex information that relates to longer regions of the target DNA sequence.

Vedúci: Mgr. Bronislava Brejová, PhD.

Dátum zadania: 20.11.2009
Dátum schválenia: 18.02.2011


prof. RNDr. Branislav Rován, PhD.
garant študijného programu


.....
študent


.....
vedúci

Acknowledgments

I would like to thank my supervisor Broňa Brejová for her guidance and her priceless advice and also my friends and family for their support.

I hereby declare that I wrote this thesis by myself, only with the help of the reference literature, under the careful supervision of my thesis advisor.

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Abstrakt

Autor: Bc. Marcel Kucharík
Názov práce: Nový algoritmus pre využitie externej informácie pri hľadaní génov
Škola: Univerzita Komenského v Bratislave
Fakulta: Fakulta matematiky, fyziky a informatiky
Katedra: Katedra informatiky
Vedúci práce: Mgr. Bronislava Brejová, PhD

Cieľom hľadania génov je označiť význačné miesta v DNA sekvencii – gény. Programy určené na riešenie tohto problému používajú informáciu o DNA štruktúre z už ošetrovaných sekvencií a externú informáciu. V práci analyzujeme existujúce programy na hľadanie génov a popisujeme ich obmedzenia vo využívaní externej informácie. Tieto zvyčajne dokážu využiť iba jednoduché "hinty", pozostávajúce z jedného intervalu sekvencie, keďže tieto je relatívne ľahké zakomponovať do štandardných algoritmov. Existujúce hľadače génov si nevedia poradiť s komplexnejšími hintami, ktoré sa skladajú z viacerých intervalov. Pri delení informácie na menšie kúsky dochádza k strate informácie a preto sme vyvinuli algoritmus schopný pracovať aj s komplexnejšou externou informáciou. Tento algoritmus je založený na skrytých Markovových modeloch (HMM), podobne ako predchádzajúce hľadače génov, ale využíva iný prístup k spracovávaniu externej informácie. Naš algoritmus hľadá postupnosť označení stavov HMM s najlepším skóre, pričom postupnosť dostáva odmenu za každý uposlúchnutý hint. Tieto hinty sú modelované ako alternatívne cesty v grafovej reprezentácii problému.

Dokázali sme, že náš prístup zvýšil presnosť predikcie génov, hlavne na génovej úrovni. Časová zložitosť algoritmu je lineárna v závislosti od dĺžky vstupnej sekvencie a od dĺžky všetkých hintov dokopy, a kvadratická v závislosti od počtu stavov HMM a počtu hintov. Táto časová zložitosť je porovnateľná s časovou zložitosťou existujúcich hľadačov génov.

Kľúčové slová: hľadanie génov, skryté Markovove modely (HMM), externá informácia pri hľadaní génov, orientovaný acyklický graf.

Abstract

Author: Bc. Marcel Kucharík
Caption: A New Algorithm for Using External Information
in Gene Finding
University: Comenius University in Bratislava
Faculty: Faculty of Mathematics, Physics and Informatics
Department: Department of Computer Science
Supervisor: Mgr. Bronislava Brejová, PhD

The goal of gene finding is to mark important places in DNA sequence – genes. Programs solving this task are using information about DNA structure from already annotated sequences and external information. We study existing gene finding programs and find their limitations in using external information. Typically they can process only simple hints consisting of a single interval, because these are relatively easy to incorporate to standard algorithms, but cannot cope with complex hint consisting of multiple intervals. This is unwanted information loss, and therefore we developed an algorithm able to process complex information. It is based on Hidden Markov models and the Viterbi algorithm as previous gene-finders, but uses a different approach of adding external information into the model. Our algorithm finds the sequence of state labels of the HMM with the best score, where a sequence gets a bonus for each respected hint. Hints are modeled as alternative paths in a graph representation of the problem.

We proved that this approach increases the accuracy of gene prediction, especially at the gene level. Time complexity of the algorithm is linear in sequence length and in the total length of all hints, while quadratic in the number of hints and HMM states. This is comparable to time complexity of existing gene finders.

Key words: gene finding, Hidden Markov models (HMM), external information in gene finding, directed acyclic graphs.

Contents

Introduction	1
1 Bioinformatics and gene finding	3
1.1 Bioinformatics	3
1.2 Sequence annotation	4
1.2.1 The problem of sequence annotation	5
1.2.2 Hidden Markov models (HMM)	6
1.2.3 Hidden Markov models for gene finding	7
1.2.4 The Viterbi algorithm	10
1.3 Gene finding with external information	11
1.3.1 External information and its sources	11
1.4 Combination of external information with HMMs	14
1.4.1 Pointwise hints	14
1.4.2 Interval hints	15
1.4.3 Other approaches	16
2 Algorithm for using complex hints	17
2.1 Finding maximum consistent hint subset	17
2.1.1 Definitions	17
2.1.2 Hint path problem	20
2.1.3 Generalized label restrictions in hints	26
2.2 Algorithm for combining complex hints with an HMM	27
2.2.1 Joint graph description	27
2.2.2 Best score path	32
2.3 Time complexity	34

2.3.1	Computing the extension relation	35
2.3.2	Subset bonuses	35
2.3.3	Edge counting	37
2.3.4	Improvement	37
3	Implementation and results	39
3.1	Implementation	39
3.1.1	Inputs and usage	40
3.1.2	Hint set format description	40
3.1.3	Model training and data sets	41
3.2	Results	42
3.2.1	Real data hint sets	42
3.2.2	Artificial hint sets	46
	Conclusion	49
	Resumé	51

Introduction

Since the end of the Human Genome Project in 2003, a human DNA sequence has been determined to nearly 100%. Nowadays, we have also many sequences from different organisms. These sequences are very long strings over the alphabet $\{A, C, T, G\}$ and to use them effectively in research, further analysis is required to find parts of the sequence called genes which are of major importance. Genes are protein coding parts of a sequence, but they are not very easy to distinguish from surrounding noncoding sequence. Due to a vast size of DNA sequences, there is a need for an automated solution of this problem, since a manual gene finding would be very time consuming. Many gene finding programs have arisen to cope with this task using only genomic sequences; such an approach is called *ab initio* – from the beginning. Soon, they were expanded with the use of external information, and the gene prediction accuracy took a huge step forward. The external information gained from a variety of sources is typically in a preprocessing stage cut to simple intervals (or points), because gene-finders cannot process more complex types of external information.

We propose an algorithm able to cope with a more general case of external information. The algorithm is based on the same principle as existing gene-finders, but it uses a different approach in the processing of the external information. In particular, we first formulate and study a simpler problem of finding the subset of input complex hints that can be used simultaneously and have the highest possible total score. We then extend the algorithm to also incorporate information from a hidden Markov model representing statistical properties of genes and non-coding regions. The algorithm is fully described in Chapter 2.

Additionally, we experimentally evaluate the contribution of complex hints compared to simple interval hints and pointwise hints. We built a simple gene finder based on our algorithm, and created several hint sets based on both real and artificial data. Chapter 3 is dedicated to the implementation and evaluation of the contribution of complex hints.

Chapter 1

Bioinformatics and gene finding

1.1 Bioinformatics

Bioinformatics is a very young science discipline (first beginnings are dated back to the year 1979 [Ouzounis and Valencia, 2003]). It began as an application of computer science techniques to solve tasks of molecular biology. Its main subfields are genomics and genetics – disciplines, whose main interest is the processing of huge amounts of data, mainly DNA sequences. DNA sequences are simply sequences of molecules adenin(A), cytozin(C), tymin(T), and guanin(G). These small molecules are called *bases*. Similarly, protein sequences are composed of 20 different amino-acid molecules. One amino acid is coded by three bases from DNA – this triple is called a *codon*.

Since its birth, bioinformatics became a very complex field of science with a wide area of research. Today, its main activities include assembling and analyzing DNA and protein sequences, finding sequence similarities between species and creating phylogenetic trees, discovering 3D structures of proteins, and many others. It helped to acquire a huge amount of information from molecular biology in relatively short time. Many web-pages, (web-)databases, and journals are now dedicated only to bioinformatics and they provide tools and knowledge base for scientists from the field of bioinformatics and biology.

1.2 Sequence annotation

DNA sequence annotation is the process of marking genes and other biologically interesting parts of DNA sequences. A gene is a continuous part of DNA sequence, which encodes a protein (molecule necessary for living).

Genomes can be divided into two basic types: genomes of prokaryotic and eukaryotic cells.

Prokaryotic cells have very well defined and documented signals that control gene expression, the process of producing the proteins encoded by individual genes. Protein coding sequence in prokaryotic cells consists only of one continuous part – "open reading frame" (ORF). This ORF is often from hundreds to thousands bases long and it is ended with a so called stop codon. Statistical view on stop codon appearance in the sequence can give a good insight into the sequence structure. There are 3 stop codons out of 64 different codons, and therefore we can expect a stop codon in a random sequence once in approximately 21 codons, that is 63 bases. Beginnings and ends of the coding sequence can be estimated quite precisely with a help of the fact above and other periodicities in the coding sequence. Therefore, a process of annotation in prokaryotic cells is relatively straight-forward.

Gene finding in eukaryotic sequences is much more difficult, especially in higher, more complex organisms like for example human or other mammals. Regulatory signals are more complicated and less discovered than in prokaryotic cells and therefore it is much greater problem to find them in a given sequence. Furthermore, coding sequence is divided into several parts (exons) divided by non-coding sections (introns). A protein coding gene in the human DNA is usually divided into a several exons, most of them are 20 - 200 bases long [Sakharkar et al., 2004]. These are divided by introns with an average length of more than thousand bases. The goal of annotation in these genomes is not only to distinguish genes from other parts of DNA, but also delineate splice sites – sites, where exon ends and intron begins and vice versa.

1.2.1 The problem of sequence annotation

The process of annotating a DNA sequence is not well defined, since some control mechanisms in a cell are still unknown. Therefore, this problem is formulated probabilistically. The most simple formulation can be stated as:

Input: a sequence of characters from the alphabet $\Sigma_{in} = \{A, C, T, G\}$ of length k

Output: the most probable marking of each character of the input sequence with a label from the alphabet $\Sigma_{out} = \{X, I, E\}$, where X stands for an intergenic part of sequence, I for introns and E for exons.

The alphabet Σ_{out} is called a label set. To completely define this problem, we need to specify a probabilistic model assigning probabilities to different possible sequences of labels. To do so, we need additional information to help us distinguish exons and introns from intergenic regions.

How to get to know a gene?

Transcription of a gene is done by complex molecules, which bind to special short sequences in the DNA – **signals**. Signals appear on the splice sites, at the beginning and the end of genes, and at the other biologically significant places. A good example of the signal is the stop codon mentioned above, which is located in exons only at the end of the last exon.

The input character frequency in exons is also different from the other parts of the sequence. Even all three positions in codon have different character frequencies. These frequencies are usually stored in frequency tables for various parts of the sequence and we can construct these tables by analyzing sequences with known annotation.

We can also use already annotated sequence to gather statistical information about the structure of DNA sequences – the average number of exons in a gene, average number of genes, an average length of exons, introns, and genes.

We are now able to redefine the problem of sequence annotation by enhancing the input with the information about the signal structure, the DNA

structure, and the frequency tables.

Problem formulation II

Input: a sequence of characters from the alphabet $\Sigma_{in} = \{A, C, T, G\}$ of length k ; a list of signals, their structure and positioning in a DNA sequence; frequency tables for different sequence parts; statistical information about an unknown sequence

Output: the most probable marking of each character in the input sequence with a label from the alphabet $\Sigma_{out} = \{X, I, E\}$, where X stands for an intergenic part of sequence, I for introns and E for exons.

Although we have described the types of available statistical signals, we still need to define precise probability distribution over label sequences. This problem can be practically solved by Hidden Markov models.

1.2.2 Hidden Markov models (HMM)

Hidden Markov models are finite-state generative probabilistic automata, which generate a sequence of characters over some finite alphabet (in our case $\{A, C, G, T\}$). Emission and transition probabilities are defined in each state of the HMM (Figure 1.1). Emission probabilities capture frequency differences, while transition probabilities and the number of states capture signals and a structure of DNA. The HMM starts generating the sequence in a state chosen according to the start probabilities. Afterwards, it proceeds in the following way: one character is generated according to emission probabilities for the current state and then we move to the next state (not necessary different) according to transition probabilities. Each state has a label from the label set, this label represents the searched annotation.

Emission and transition probabilities sum up to 1 for each state and start probabilities sum up to 1 through all the states. The "hidden" property stands for the fact that after looking at the generated sequence, we do not know which states generated these characters. We can estimate only the most probable path through HMM that generates the sequence.

Formally, let $Q = \{q_1, \dots, q_n\}$ be HMM state set, $O = \{o_1, \dots, o_m\}$ be a generated alphabet. Next, let be defined [Paul E. Black, 2008]:

- **emission probabilities - E**

$$E = \{e_{i,k} = P(o_k|q_i)\}, \text{ where } o_k \in O \text{ and } q_i \in Q.$$

$$\text{such that: } \forall q_i \in Q : \sum_{o_k \in O} e_{i,k} = 1$$

- **transition probabilities - T**

$$T = \{t_{i,j} = P(q_j \text{ at time } t + 1 | q_i \text{ at time } t)\}, \text{ where } t \geq 1 \text{ is time (generation step) and } q_i, q_j \in Q$$

$$\text{such that: } \forall q_i \in Q : \sum_{q_j \in Q} a_{i,j} = 1$$

- **start probability - π**

$$\pi = \{p_i = P(q_i \text{ at time } t = 1)\}.$$

$$\text{such that: } \sum_{q_i \in Q} p_i = 1$$

A hidden Markov model is usually defined as a triple (T, E, π) , since the sets Q and O are known.

Alternative notation will be also used in the following text for clarity reasons: $p(q_x) = p_x$, $e(q_x, y) = e_{x,y}$ and $t(q_x, q_y) = t_{x,y}$.

Probability of generating a sequence

The probability that an HMM generates a sequence $S = (o_{s_1}, o_{s_2}, \dots, o_{s_k})$ by a sequence of states $F = (q_{f_1}, q_{f_2}, \dots, q_{f_k})$ in the first k steps is simply a product of a start probability of a state q_{f_1} and k emission probabilities and $k - 1$ transition probabilities.

$$P(F, S) = P(q_{f_1})e(q_{f_1}, o_{s_1}) \prod_{i=2}^k t(q_{f_{i-1}}, q_{f_i})e(q_{f_i}, o_{s_i}) \quad (1.1)$$

1.2.3 Hidden Markov models for gene finding

In order to solve the gene finding problem from Section 1.2.1, we need to create an HMM, which will characterize signals, DNA structure and it will

generate characters with proper emission frequencies. After creating the HMM, we need to find the most probable path (a state sequence) through this HMM, which generates the input sequence. A state sequence denotes annotation, because state labels represent parts of DNA.

Formally, sequence annotation problem is defined on HMM as:

For a given sequence S over alphabet Σ_{in} of length k and an HMM, find the most probable path F through states:

$$F = \arg \max_{F \in Q^k} P(F | S)$$

This problem can be solved by Viterbi algorithm described in Section 1.2.4.

The simplest HMM for gene finding would have three states: X , E and I , for intergenic regions, exons and introns, respectively. However, in order to add signals we extend the model with many new states. For example: a stop codon signal can be implemented by a sequence of three states with transition and emission probabilities allowing to generate only the desired stop codon sequence (Figure 1.1). Other signals can be implemented similarly.

Character frequency tables can be included into emission probabilities of the states X , E and I with the exception of E state, for which there exist three different frequency tables corresponding to different positions within a codon. This state is decomposed into three states E_1 , E_2 and E_3 connected to a directed cycle of length three. Thus, the length of exons in each output gene will be divisible by three, which agrees with our biological intuition. Transition probabilities are set according to statistic information from annotated sequences.

Note that in general by extending the HMM with more prior biological knowledge about signals and gene structure we increase the number of states, leading to increased time consumption for the Viterbi algorithm.

Existing *ab initio* gene finders include: HMMGENE [Krogh, 1997], GENESCAN [Burge and Karlin, 1997], AUGUSTUS [Stanke et al., 2006], and CONRAD [DeCaprio et al., 2007]. While HMMGENE is based on classical HMMs, GENESCAN and AUGUSTUS use generalized HMMs in which each state can generate sequence of characters in each step. Each state can have different emission probabilities and sequence length distribution. CONRAD

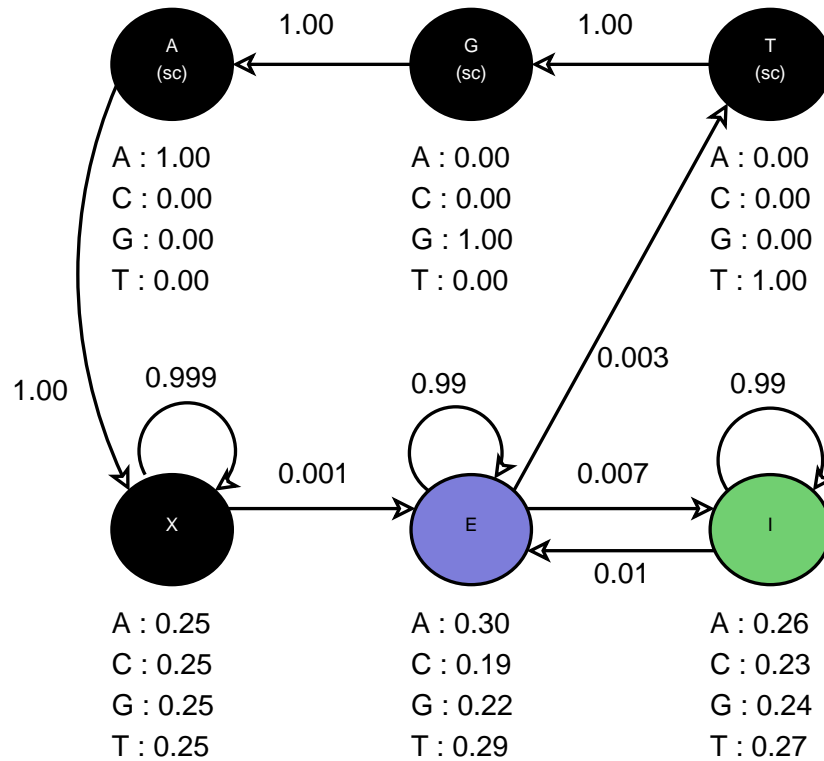


Figure 1.1: Example of an HMM – emission probabilities are written below states, transition probabilities are on the edges (states not connected by an edge have zero transition probability), states are colored according to their labels, states labeled with *(sc)* represent TGA stop codon signal

uses conditional random fields, which are further generalization of the hidden Markov models.

1.2.4 The Viterbi algorithm

The goal of the Viterbi algorithm is to find the most probable state path F that generated a given input sequence S :

$$F = \arg \max_F P(F|S) \quad (1.2)$$

This can be found by a simple dynamic programming algorithm named after Andrew Viterbi [Forney, 1973]. This algorithm can be described by recursion:

$$\forall q \in Q : v(1, q) = p(q)e(q, s_1) \quad (1.3)$$

$$\forall i, 1 < i \leq k, \forall q \in Q : v(i, q) = \max_{\bar{q}} \{t(\bar{q}, q)e(q, s_i)v(i-1, \bar{q})\} \quad (1.4)$$

Equation $v(i, j) = p$ means that the most probable path that generates a sequence s_1, \dots, s_i and ends in a state j has probability p . After computing values $v(k, q)$ for all states $q \in Q$, we get maximal probabilities of generating input sequence S and ending in different states and we can choose state q yielding the maximum value. This will be the last state of the resulting state path. Computation of this function can be seen as filling a table from left to right, where a cell in a row j and a column i has a value of $v(i, j)$. To compute entries of column i we need only entries from column $i-1$, which are already computed.

Now we have the probability and the end state, but for acquiring the whole state path we have to store the information about previous cells. This can be done without increasing the time complexity by computing values $f(i, j)$:

$$\forall q \in Q : f(1, q) = 0 \quad (1.5)$$

$$\forall i, 1 < i \leq k, \forall q \in Q : f(i, j) = \arg \max_{\bar{q}} \{t(\bar{q}, q)e(q, s_i)v(i-1, \bar{q})\} \quad (1.6)$$

Value $f(i, j)$ stores the row index of the cell from column $i-1$ used in computation of $v(i, j)$. Tracing backwards in table f from the most probable

end state yields the most probable state path and therefore the annotation.

$$g(k) = \arg \max_q v(k, q) \quad (1.7)$$

$$\forall i, 1 \leq i < k : g(i) = f(i + 1, g(i + 1)) \quad (1.8)$$

A function g represents the searched state path, $F = (g(1), g(2), \dots, g(k))$. The annotation will be $(label(g(1)), label(g(2)), \dots, label(g(k)))$.

This algorithm has a time complexity of $O(kn^2)$, where k is the length of the sequence and n is the number of states in HMM, because it has to go through the whole table of size $k \times n$ and it performs n operations for each cell. Viterbi algorithm implementations (and similar algorithms, which compute with probability products) have an underflow problem in floating point arithmetics. Therefore, probabilities are converted to logarithmic scale and the resulting numbers are added up. This type of computation is called "computation in log-space".

A time complexity of the Viterbi algorithm is linear in the sequence length, which enables its use for gene finding, because the sequence length k is huge in real data. For example, human genomes have approximately $3 \cdot 10^9$ bases.

1.3 Gene finding with external information

Algorithms using signals, DNA sequence structure, and frequency tables are looking for probable genes. This search is called *ab initio* search, the Latin phrase *ab initio* means *from the beginning, at first*. This search is without further knowledge – external information. However, such external information is often accessible and it can help to improve the accuracy of gene prediction.

1.3.1 External information and its sources

Many kinds of biological data can be used as external information. For example, we can search for *homologs* – similar genes, which appear in many different species, because they have arisen from common ancestor. Besides,

we can experimentally detect gene expression using biotechnology and thus acquire probable annotation of gene parts.

In this section we describe overall most commonly used sources of external information for gene finding, which served as motivation for this work devoted to generalization of frameworks for external information usage.

ESTs - "expressed sequence tags"

ESTs are short DNA sequences (usually 200 – 500 bases long), which are acquired by sequencing cDNA (complementary DNA to mRNA) [NCBI, 2011b]. In the process of gene expression, gene is first transcribed to RNA, introns are cut out, and the resulting mRNA is then translated to protein. The mRNA molecules contain only exons, since the introns have been cut out. It is possible to extract mRNA molecules from cells and sequence parts of them, obtaining EST sequences. Such sequences are then aligned to the input DNA sequence and perfect or at least the best alignment (because sequencing errors can occur in both sequencing events) denotes exon positions in the unknown sequence. EST database called **dbEST** is freely accessible and contains over 69 millions of EST, among them over 8 millions for human [NCBI, 2011a] [entry from April, 2011].

Proteins

Coding parts of genes are translated into proteins, which have specific functions in a cell. These proteins can be sequenced or a database of known proteins from closely related species can be used. A protein sequence can be aligned to a DNA sequence (Figure 1.2), because we know all codon to amino-acid transformations. Similarly to ESTs, this best alignment yields parts of annotation. Many protein databases exists: **PDB** contains only proteins with known 3D structure and has over 72 thousand proteins [NCBI, 2011c] [entry from April, 2011]. **UniProt** is a database of all proteins split into two separate databases (the more accurate sub-database Swiss-Prot contains over half million proteins [SIB, 2011]) [entry from April, 2011].

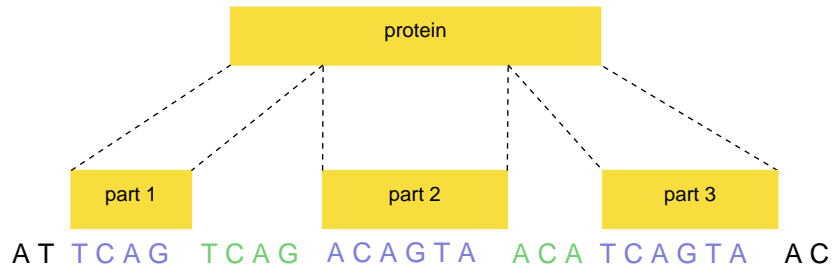


Figure 1.2: The protein alignment to target sequence – blue parts are exons, green parts are introns, and black are intergenic regions

Exon conservation

Exon conservation method is based on comparing DNA sequences from different, but relatively close, species – highly similar parts from this comparison are likely to be coding, because coding parts are evolutionary more conserved against mutations. Mutations in a coding part may turn the encoded protein dysfunctional, potentially leading to death of the organism.

External information can improve the accuracy of gene finding. However, external information has to be used with caution, because it is sometimes inaccurate due to errors in experimental methods used to gather the data.

EST, protein or DNA sequences need to be first aligned to the input sequence. Many programs exist for this purpose and gene finder developers have only to use the outputs of these programs as an input to theirs. I will mention at least some of these programs: BLASTZ and BLAT (ESTs, protein and other alignments), EXONIPHY and PHASTCONS (conservation of exons with use of phylogenetic trees), TRANSMAP (mapping known genes between species), and others.

1.4 Combination of external information with HMMs

External information can be used in many ways in the gene finding program. Here we will briefly mention some of the past approaches. Many of them convert external information to hints in the form of intervals covered by same label or even more extremely to a label indicated at each single position independently of others. Therefore, they introduce information loss, because external information is natively in a more complex form. For example, a hint from an EST alignment may start at position X and have representation: `exon(20 bases)-intron(20 bases)-exon(10 bases)`. Such a hint is in a preprocessing stage cut to 3 hints of interval type: `exon(20 bases)` starts at X , `intron(20 bases)` starts at $X + 20$, and `exon(10 bases)` starts at $X + 40$. State paths through the HMM that obey one of these intervals then receive a bonus (their probability is increased compared to other state paths). In our example, the state path that has exon states between X and $X + 50$ gets two bonuses for the first and third hint. Yet such a state path perhaps should not get any bonus, since it does not obey external information in its whole extent. The goal of this thesis is to overcome this shortcoming of existing methods. We will introduce our new algorithm in Chapter 2, which is able to handle the external information in a form of the sequence of intervals.

1.4.1 Pointwise hints

The easiest way to divide the external information is to cut it to single position hints. This approach was used in program EXONHUNTER [Brejová et al., 2005], where all external information is combined into one "superadvisor" object. This object defines the probability $P(A | ev)$, where A is annotation and ev is external information available for superadvisor. HMM model defines probability $P(A | S)$, where S is DNA sequence. The goal is to get $P(A | S, ev)$, which can be computed by Bayes' rule using assumption of independence between external and internal information. The most probable annotation in $P(A | S, ev)$ is computed by adaptation of Viterbi algorithm,

where emission probabilities on position i , in state j are multiplied by fraction $x_{label(j)}^{(i)}/prior(l(j))$, where $label(j)$ is label of state j , $x_j^{(i)}$ is superadvisor prediction for i^{th} character and label j , and $prior(j)$ is prior probability of label j .

The main advantages include:

- for each source of external information, ExonHunter allows to specify the information about one position in the form of a partition of the set Σ_{out} and probability distribution over all partition elements. Therefore, this approach natively uses hints with multiple labels at the same position.
- the ability to use every possible source of information – every type of complex information can be cut to single position hints with multiple labels.
- speed – this approach is only slightly more time consuming than *ab initio* gene finding using Viterbi algorithm
- this approach keeps the probabilistic nature of the Viterbi algorithm

The biggest limitation of this approach is surely the loss of information in the process of cutting the external information.

Pointwise hints were also used in other gene finders, such as GENOMES-CAN [Yeh et al., 2001] and CONRAD [DeCaprio et al., 2007].

1.4.2 Interval hints

External information can be cut to intervals, where each interval has one label associated with it. This is big generalization of the single position approach, but it gives rise to many disadvantages as well. Such an approach has been used in AUGUSTUS+ [Stanke et al., 2008], which is improvement of the *ab initio* program AUGUSTUS [Stanke et al., 2006]. This gene finder uses a generalized Hidden Markov models – GHMM. AUGUSTUS+ has augmented emission alphabet with hints by means of Cartesian product and these hints are generated by the GHMM similarly to DNA sequence. The most probable path through the GHMM is computed by adaptation of the

Viterbi algorithm; this adaptation allows to use only one hint at each position.

Advantages:

- usage of more general class of external information in the form of interval
- probabilistic nature

Limitations:

- problem of assigning the emission probability to hints
- dependence – absence of external information about some label in a part of the sequence decreases the probability of this label in this part
- speed – the running time is increased by an additive term proportional to the total length of hints

1.4.3 Other approaches

Some approaches do not use the HMM directly, but use the results of existing gene finders as evidence. One example is `EVIDENCEMODELER` [Haas et al., 2008]. This program uses evidence in the form of intervals with state restriction and it finds consensus of the evidence. The advantage of this approach is improved gene prediction compared to the gene finders, whose annotation is the input of EVM. The major limitation is in speed – firstly all evidence have to be gathered from many different programs and only afterwards consensus can be computed.

Chapter 2

Algorithm for using complex hints

In this chapter we propose a new algorithm for using external information in gene finding. This algorithm handles a more general class of external information in polynomial running time. The algorithm is fully described in following sections. At the end of the chapter, we analyze the time complexity of the algorithm, which is comparable to those used in existing gene finders.

2.1 Finding maximum consistent hint subset

We will first concentrate on a simpler version of the problem in which we ignore the HMM and consider only information obtained from the hints. We will now define several terms, which will be used in this chapter.

2.1.1 Definitions

A complex hint is a continuous sequence of intervals, each interval imposing a label restriction to states, formally:

Definition 2.1.1 (Complex hint) *A complex hint is a sequence of intervals $H_i = (h_1^{(i)}, h_2^{(i)}, \dots, h_{m_i}^{(i)})$, where each interval $h_j^{(i)}$ is defined as a triple*

$h_j^{(i)} = (b_j^{(i)}, e_j^{(i)}, l_j^{(i)})$. Values $b_j^{(i)} \in \mathbb{N}$, $e_j^{(i)} \in \mathbb{N}$, and $l_j^{(i)} \in \Sigma_{out}$ are the position of the beginning of an interval, position of the end of the interval, and a state restriction label, respectively ($b_j^{(i)} \leq e_j^{(i)}$). Furthermore, intervals in a complex hint must cover a continuous region without overlaps, that is $e_j^{(i)} + 1 = b_{j+1}^{(i)}$ must hold for $j = 1, 2, \dots, m_i - 1$. Position $b(i) = b_1^{(i)}$ is called the beginning of hint i and position $e(i) = e_{m_i}^{(i)}$ is called the end of hint i .

For every hint H_i , we introduce function $\ell_i : \mathbb{N} \rightarrow \Sigma_{out}$ which returns label restriction of the i -th hint at a particular position in the sequence. If hint H_i does not cover this position, the function is not defined. Formally:

$$\ell_i(x) = \begin{cases} l_j^{(i)} & \text{if } \exists j : b_j^{(i)} \leq x \leq e_j^{(i)} \\ \perp & \text{otherwise} \end{cases} \quad (2.1)$$

A complex hint set is a set of all available complex hints and their bonuses:

Definition 2.1.2 (Complex hint set) A complex hint set H is a pair $H = (B, \hat{H})$, where $B : \hat{H} \rightarrow \mathbb{R}^+$ is a bonus function and $\hat{H} = (H_1, H_2, \dots, H_c)$ is a sequence of available complex hints. Hints are ordered so that $b(i) \leq b(i+1)$ and if $b(i) = b(i+1)$ furthermore $e(i) \leq e(i+1) \forall i = 1, 2, \dots, c-1$.

For now on, we will use term "hint" for a complex hint and (simple) *interval hint* for hint with $m_i = 1$. We say that a *state path respects a hint* if it does not violate the hint label restrictions; formally:

Definition 2.1.3 (Respected hint) A complex hint $H_i = (h_1^{(i)}, h_2^{(i)}, \dots, h_{m_i}^{(i)})$ is respected by a state path $F = (f_1, f_2, \dots, f_k)$ if for every position $j \in \{b(i), \dots, e(i)\}$ we have $\text{label}(f_j) = l_i^{(i)}$, where $\text{label}(f_j)$ is the label of state f_j .

Two hints H_i and H_j ($i < j$) are intersecting if $b(j) \leq e(i)$.

Two hints are *compatible* if and only if they can be used in the same state path. This means that these two hints have the same label restrictions at every position in their intersection.

Definition 2.1.4 (Compatible hints) *Hints H_j and H_i are compatible if for every $p = b(i), \dots, e(i) : \ell_i(p) = \ell_j(p) \vee \ell_j(p) = \perp$.*

Definition 2.1.5 (Subset hint) *Hint H_j is subset of hint H_i if these two hints are compatible, $b(j) \geq b(i)$, and $e(j) \leq e(i)$.*

Two hints are equal if and only if they are subsets of each other. From now on we will assume that no two hints H_i and H_j ($i \neq j$) are equal, otherwise we can create a smaller hint set:

$$H_{new} = (B_{new}, \hat{H} - H_j) \quad (2.2)$$

$$B_{new}(H_i) = B(H_i) + B(H_j) \quad (2.3)$$

$$B_{new}(H_x) = B(H_x) \quad x \neq i \quad (2.4)$$

Definition 2.1.6 (Hint extension) *Hint H_j is an extension of hint H_i if these two hints are compatible, $b(i) \leq b(j) \leq e(i) + 1$, and $e(j) > e(i)$.*

Note that two compatible and intersecting hints are in either subset or extension relation, but not in both. Additionally, hint H_j can be an extension to hint H_i only if $j > i$, this condition does not apply to subset hints. If we augment the extension relation with non-intersecting hints, we get a transitive relation "weak extension". This transitivity is very important for polynomial-time running time of our algorithm.

Definition 2.1.7 (Weak hint extension) *Hint H_j is a weak extension of hint H_i if H_j is an extension of hint H_i or $i < j$ and hints H_j and H_i do not intersect.*

Lemma 2.1.1 (Transitivity of weak extension relation) *Consider three hints $H_i, H_j, H_k, i < j < k$. If hint H_j is weak extension of H_i , and hint H_k is weak extension of H_j then H_k is weak extension of H_i .*

Proof: Case analysis (two possibilities):

- $e(i) < b(k)$: then H_i and H_k do not intersect, and therefore H_k is clearly a weak intersection of H_i

- $e(i) \geq b(k)$: since $i < j < k$, we have $b(i) \leq b(j) \leq b(k)$. At the same time H_j is not a subset of H_i and therefore $e(j) \geq e(i)$. This implies that interval $\langle b(k), e(i) \rangle$, which is the intersection of hints H_i and H_k , is contained in H_j . Therefore this interval is in intersection of H_i and H_j and also in intersection of H_j and H_k . Thus hints H_i and H_k are compatible. Since $b(i) \leq b(k) \leq e(i) + 1$, $e(k) > e(j) > e(i)$ holds, hint H_k is an extension of H_i and therefore it is also its weak extension. \square

2.1.2 Hint path problem

These definitions give rise to the following natural question:

Problem 2.1.1 (Maximal bonus problem) *Given a complex hint set H find a set of hints that can be respected by the same path and that has maximal total bonus value.*

Problem 2.1.1 can be solved by an algorithm, which constructs a graph G called a *hint graph* (Figure 2.1). Every single position within each hint corresponds to one node in the hint graph. Thus, each node has assigned its sequence position and state restriction label. Edges can be divided into two groups: edges between hint positions inside a hint (*hint edges*) and edges between hints (*compatibility edges*). A hint edge goes from a position j inside hint H_i to the next position $j + 1$. Weight of the hint edge equals to added bonuses of all subset hints of hint H_i ending at position j . If no such subset hints exist, the weight is zero.

A compatibility edge goes from hint H_i to hint H_j that is its weak extension. Furthermore, if hints H_i and H_j intersect, then the edge is leading to the node at position $e(i) + 1$, otherwise leads to beginning of hint H_j . This compatibility edge has the weight equal to the bonus of the hint H_j plus sum of bonuses of all subset hints of hint H_i , ending at position $e(i)$.

Artificial *sentinel hints* H_0 and H_{c+1} are added to the hint set with zero bonus. These hints represent the beginning and the end of the graph respectively, so $H_0 = ((0, 0, \emptyset))$, $H_{c+1} = ((k + 1, k + 1, \emptyset))$. This allows us to count the bonus of the first respected hint and the bonus of subset hints, that end at position k .

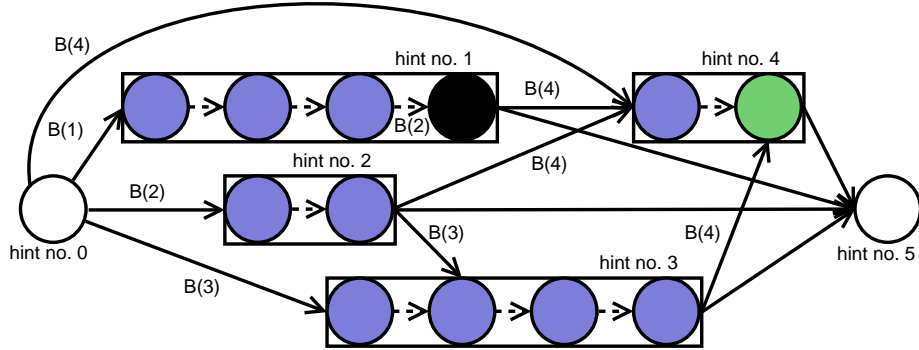


Figure 2.1: Hint graph – hint edges are dashed, compatibility edges are solid, edges have their weights on them, nodes are colored according to label restrictions. No edge leads from hint 1 to hint 3, because these hints are not compatible

It is easy to see that this graph is acyclic. Each possible path in this graph corresponds to a set of compatible hints (because of the forward transitive property of weak extension) and the weight of this path is equal to the total bonus of all these compatible hints. Therefore, the algorithm needs to find the longest possible path in this graph. Nodes on this path give the resulting respected hints, plus we must add every subset hint of these hints. Note that the longest path always starts in the start node, since bonuses are positive. Finding the longest path in a weighted directed acyclic graph can be done in time linear in the number of edges [Cormen et al., 1989]. Formally, we can describe this graph and algorithm as follows:

Hint graph description

Definition 2.1.8 (Hint graph) *A hint graph for hint set H is graph $G = (V + v^{(0)}, A)$, where:*

- $V = \{v_j^{(i)} \mid \forall i : H_i \in \bar{H}, \forall j : b(i) \leq j \leq e(i)\}$, where:
 - $\bar{H} = (\bar{B}, \hat{H} \cup \{H_0 = ((0, 0, \emptyset)), H_{c+1} = ((k + 1, k + 1, \emptyset))\})$

$$\circ \bar{B}(H_i) = \begin{cases} B(i) & \text{if } i = 1, 2, \dots, k \\ 0 & \text{otherwise} \end{cases}$$

- $A = A_h \oplus A_c$ (hint edges and compatibility edges)

$$A_h = \left\{ a_j^{(i)} = \left(v_j^{(i)}, v_{j+1}^{(i)} \right) \mid \forall i, j : v_j^{(i)}, v_{j+1}^{(i)} \in V \right\} \quad (2.5)$$

$$A_c = \left\{ a_{e(i)}^{(i,j)} = \left(v_{e(i)}^{(i)}, \max(b(j), e(i) + 1) \right) \mid \forall i, j : \text{weak}(i, j) \right\} \quad (2.6)$$

where $\text{weak}(i, j)$ is predicate, which is true if and only if hint H_i is weak extension of H_j .

- all weights reside on edges:

$$|a_{e(i)}^{(i,j)}| = B(j) + \sum_{H_x \text{ is subset of } H_i, e(x)=e(i)} B(x) \quad (2.7)$$

$$|a_j^{(i)}| = \sum_{H_x \text{ is subset of } H_i, e(x)=j} B(x) \quad (2.8)$$

We intentionally put the bonuses on edge weights (and not on nodes, which is more intuitive). We did not design it this way, because the hint graph and Algorithm 1 will be used with modifications in the final algorithm. Subscripts on the nodes and edges are their positions in the sequence, superscripts corresponds to the hint number of hint or hints that give rise to this node or edge.

Algorithm and proof

Algorithm 1 Find respected hints with maximal bonus value together in complex hint set H

1. Let $G = (V, A)$ be hint graph as defined in Definition 2.1.8.
 2. Let $P = (v_0^{(0)}, v_{r_1}^{(p_1)}, \dots, v_{r_q}^{(p_q)}, v_{k+1}^{(c+1)})$ be nodes of the longest path in graph G and B its length.
 3. For every node $v_{r_i}^{(p_i)} \in P - v_0^{(0)} - v_{k+1}^{(c+1)}$, include hint H_{p_i} and all its subsets to the set of respected hints R .
 4. Return R as the set of respected hints and B as their bonus value.
-

Proof: Firstly, we introduce few terms:

Compatible hint set: hint set $H = (B, \hat{H})$ is compatible if every two hints $H_i, H_j \in \hat{H}$ are compatible.

Hint respected by a path: hint H_i is respected by path P in G if every node $v_j^{(h)}$ ($b(i) \leq j \leq e(i)$, $1 \leq h \leq c$) on path P agrees with the state restriction label of the hint, so $\ell_i(j) = \text{label}(v_j^{(h)})$ must hold, where $\text{label}(x)$ is the state restriction label of node x .

Path describing a hint set: path P describes a hint set H if every hint H_i from the hint set is respected by the path. Note that one path can describe in general multiple hint sets and one hint set can be described by several paths.

Lemma 2.1.2 *For every compatible hint set $H = (B, \hat{H})$ (subset of the hint set that creates the graph) exists a path P in the hint graph that describes it such that $\text{length}(P) \geq \sum_{H_i \in \hat{H}} B(H_i)$. Conversely, for every path P exists a compatible hint set H that is described by the path P such that $\text{length}(P) = \sum_{H_i \in \hat{H}} B(H_i)$.*

Proof(Lemma 2.1.2): We split the \hat{H} to two hint sets $\hat{H} = R \oplus S$, where S is set of subset hints and R set of "path" hints:

$$S = \{H_i \mid \exists H_j \in \hat{H} : H_i \text{ is subset of } H_j\}$$

$$R = \hat{H} - S$$

We will prove that there exists a path P going through the nodes corresponding to hints from R , this path respects hints from \hat{H} , and $\text{length}(P)$ is at least the sum of hint bonuses $H_i \in \hat{H}$. We order the hints in $R = (H_{p_1}, \dots, H_{p_r})$ as in a hint set ($b(i) \leq b(i+1)$) and if $b(i) = b(i+1)$ furthermore $e(i) \leq e(i+1) \forall i = 1, 2, \dots, r-1$). We construct a path P going through nodes in the hint graph in R as follows. The path starts in the first node, continues through compatibility edge to node $v_{b(p_1)}^{(p_1)}$, then continues through hint edges through nodes $v_{b(p_1)+1}^{(p_1)}, \dots, v_{e(p_1)}^{(p_1)}$, and then through a compatibility edge to the next hint node $v_{\max(b(p_2), e(p_1)+1)}^{(p_2)}$. In the same way the path continues

through all hints to the node $v_{e(p_r)}^{(p_r)}$, then through compatibility edge to node $v_{k+1}^{(c+1)}$. This path clearly exists, since all described nodes and edges exist in the definition of the hint graph. This path respects hints from R inductively:

- The first hint is respected by the path, because the path goes through all hint nodes.
- If a compatibility edge enters a hint H_j at position $b(j)$, the path goes through the whole hint and therefore it respects the hint H_j .
- If a compatibility edge enters a hint H_j at position $e(i) + 1$, H_i is the hint, where the edge starts, H_j is respected by the path P if H_i is respected by this path, because $b(i) \leq b(j)$ and these hints are compatible, thus they have the same state restrictions in their intersection.

Each hint H_s from set S is also respected, because it is a subset of some hint $H_i \in R$ and hint H_i is respected by the path therefore subset hint H_s is also respected by path P . Now we prove that every hint $H_i \in H$ contributes at least $B(H_i)$ to $length(P)$:

- $B(H_{p_i})$ is included in the weight of compatibility edge for $H_{p_i} \in R, i \geq 1$
- $\sum_{H_i \in S} B(H_i)$ are included in hint and compatibility edges. Every hint $H_i \in S$ is a subset of at least one hint in R , let H_j be the superset of H_i with smallest number. Bonus of hint H_i is then included in the weight of the hint edge within the hint H_j or in the compatibility edge that starts in hint H_j (if $e(i) = e(j)$).

The length of the path is then $length(P) \geq \sum_{H_i \in S \oplus R} B(H_i)$, which concludes the first half of the proof.

The second half is to prove that for every path P in graph G exists compatible hint set H , described by the path. We can proceed reversely: every path describes some sets R and S , where R is a set of hints corresponding to nodes on the path and S is a set of all hint subsets of R . Labels on path P are compatible with the first hint $H_1 \in R$, because this path comes through all hint nodes. Other hints from R are compatible with the labels on the

path because every hint $H_{p_{i+1}} \in R$ is a weak extension of hint $H_{p_i} \in R$, due to compatibility edges. Therefore, using the transitive property of weak extension, hint H_{p_i} is compatible with every hint $H_{p_{i+j}}$, where $j \geq 1$. Therefore R is a set of compatible hints respected by path P . Hint $H_s \in S$ has superset hint $H_i \in R$ (due to graph construction) and subset hint is always compatible with its superset hint. Additionally, each hint compatible with superset hint is compatible with its subset hint (we lessen the requirements for compatibility). Therefore, hints $H_i \in R$ and $H_j \in S$ are compatible. The last thing is to prove that hints $H_i, H_j \in S$ are compatible:

- hints H_i, H_j do not intersect \Rightarrow compatibility
- hints H_i, H_j intersect \Rightarrow clearly exist hints $\bar{H}_i, \bar{H}_j \in R$, \bar{H}_i is superset of H_i , \bar{H}_j is superset of H_j , \bar{H}_i intersects with \bar{H}_j , and they are compatible and there is a path going through both hints, so the path is going through subsets and thus they are compatible.

Additionally, all bonuses from hint sets R and S are included in the path exactly once. Bonuses from R are on compatibility edges and the number of compatibility edges is equal to the size of the set R , because each hint $H_i \in R$ is represented by one compatibility edge, which starts it. This compatibility edge has weight corresponding to the bonus of hint H_i . Subset hint bonuses are on the edge weights in the path due to graph construction. No other subset hint bonus exists on the path P , because otherwise it must be subset of some hint from R . Therefore, $length(P) = \sum_{H_i \in S} B(H_i)$.

This concludes proof of the Lemma 2.1.2. □

We can now use the Lemma 2.1.2 to prove correctness of Algorithm 1. Let \hat{B} be the bonus of the optimal solution and P path found by the algorithm. We will prove that possibilities $B(P) > \hat{B}$ and $B(P) < \hat{B}$ cannot happen.

- $B(P) > \hat{B}$ – from Lemma 2.1.2 for every path P exists a compatible hint set respected by the path, but this cannot happen, because it will be solution better than optimal
- $B(P) < \hat{B}$ – let H be the optimal compatible hint set, again from Lemma 2.1.2 exists a path \hat{P} in the hint graph that describes this hint

set, clearly this path has length of $length(\hat{P}) \geq \hat{B} > B(P)$, so the path P is not the longest path in the hint graph.

□

2.1.3 Generalized label restrictions in hints

In the complex hint definition, we allow only one state restriction label for each interval. Hints of this form cannot be used to express some sources of evidence. For example, some experimental methods allow us to determine the whole extent of a gene, but not its splicing to exons [Ng et al., 2005]. To represent such information we may want to specify an interval and a set of allowed labels (in this case $\{E, I\}$).

Algorithm 1 will not work in this case, because multiple labels at one position makes compatibility relation non-transitive. Consider, for example, three hints (Figure 2.2): $H_1 = ((1, 4, \{E\}))$, $H_2 = ((2, 5, \{E, I\}))$, $H_3 = ((3, 6, \{I\}))$. Hint 1 is compatible with hint 2 (because a state path respecting both bonuses exists – the path through states labeled E), hint 2 is compatible with hint 3 (path through states labeled I), but hint 1 is not compatible with 3 – at positions 3 and 4 we have two incompatible state restrictions – I and E .

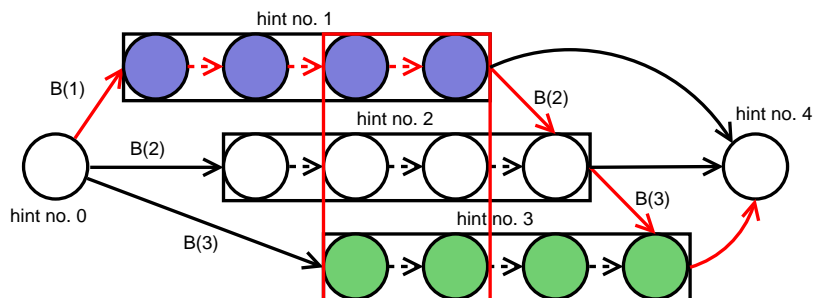


Figure 2.2: Loss of transitivity of weak extension relation – hints 1 and 3 are not compatible on area with red border, but the algorithm will find path through all hints (marked red)

Since this problem is a simplified version of the problem of implementing

complex hints into the Viterbi algorithm, we will consider only single state restriction label per interval in the main algorithm. However the complexity of the problem with label sets remains open.

2.2 Algorithm for combining complex hints with an HMM

After solving problem stated as Problem 2.1.1, we are ready to extend the Viterbi algorithm with complex hints. Formally, the goal of our algorithm is to find the most probable state path F that generated a given input sequence S , while it takes into account the bonuses of hints from a complex hint set H respected by the state path F :

Problem 2.2.1 (Sequence annotation problem) *Given an input sequence S , a complex hint set H , and an HMM model, find the state path F such as:*

$$\begin{aligned}
 F &= \arg \max_F \left(\log(P(F | S)) + \sum_{H_i \in H \text{ is respected by } F} B(H_i) \right) \\
 &= \arg \max_F \left(\log(P(F, S)) + \sum_{H_i \in H \text{ is respected by } F} B(H_i) \right) \quad (2.9)
 \end{aligned}$$

Our solution is based on creating a special graph called a *joint graph*. The first part of the joint graph represents the Viterbi algorithm, while the second is derived from the hint graph. Hints are implemented as alternative paths to the first graph part. These alternative paths carry a bonus, but need to obey state restrictions. Similarly to Algorithm 1, we are trying to find the longest path in this graph; this path gives the resulting annotation.

2.2.1 Joint graph description

If we view the Viterbi algorithm as filling the dynamic programming table, then each cell in this table can represent one node in a graph and directed edges represent possible ways of reaching a certain cell. Edge weights now consist of emission and transition probabilities in log space and finding the

longest path from an artificial start node to an artificial end node yields the same result as the Viterbi algorithm with backward pass. This algorithm has the same time complexity as the Viterbi algorithm ($O(kn^2)$), because we are searching for the longest path, what is solvable in time linear in the number of edges. Edges are between each two table columns – $O(n^2)$ edges per one column, k columns, thus $O(kn^2)$ is the total time complexity. These nodes and edges will be called *base nodes and edges*.

Nodes have their sequence position and label from the HMM. Edges are always going from a node at position i to a node at position $i + 1$. This edge property is preserved in whole graph, ensuring consistency in computing HMM probabilities, but it brought a need to modify the hint graph.

The second part of the joint graph consists of a modified hint graph, nodes of this part are called *hint nodes*. For each position within each hint we may have several nodes corresponding to HMM states consistent with the hint label at that position. Weights of all edges within the graph will also include transition and emission probabilities of the HMM. Edges starting the hint (*starting edges*) will go now only from base nodes and edges leaving the hint are divided into two parts: edges returning to base nodes (*ending edges*) and edges between hints (*compatibility edges*). Compatibility edge from hint H_i to hint H_j will go only if the hint H_j is an extension of the hint H_i . Thus a compatibility edge will always go to position $e(i) + 1$ in the hint H_j . Requirement of extension rather than weak extension is needed, because we cannot skip a sequence of HMM probability edges.

Finally, artificial *start node* and *final node*, edges from the start node to nodes at the position 1 (*beginning edges*), and edges from nodes at the position k to the final node (*final edges*) have been added to the joint graph.

Formally we can describe the joint graph as follows:

Definition 2.2.1 (Joint graph) *A joint graph for hint set H and sequence S is graph $G = (V, A)$, where:*

- $V = V_b \oplus V_h \oplus v_s \oplus v_f$
 - V_b – *base nodes*
 - V_h – *hint nodes*

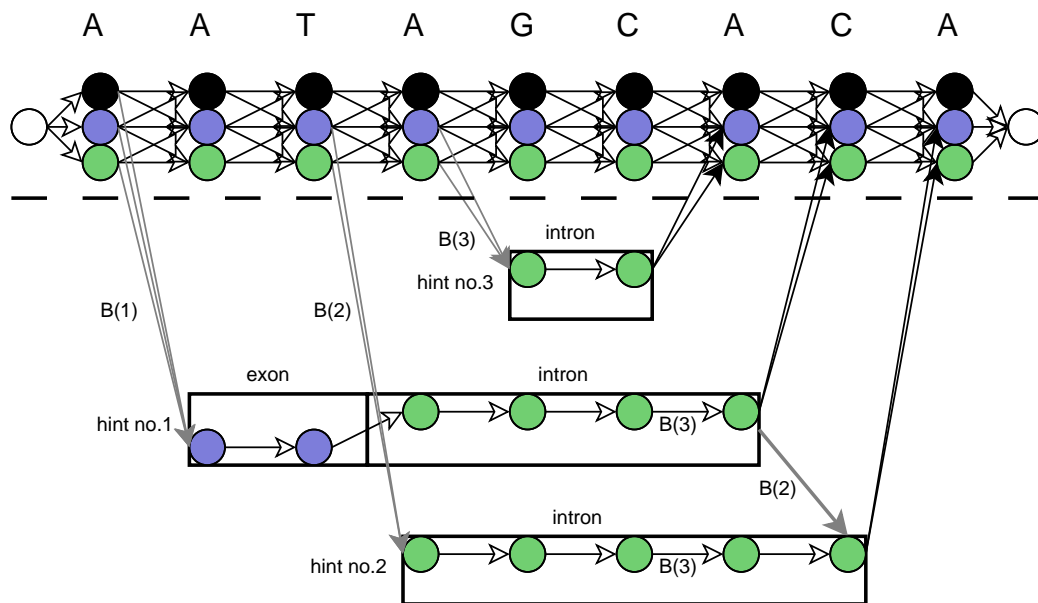


Figure 2.3: Joint graph – all edges contain transition and emission probabilities of underlying HMM. Nodes are colored according to their labels (start and final nodes are white). Starting edges and few hint edges contain hint bonuses (labeled with $B(1)$, $B(2)$, and $B(3)$). Dashed line divides the base part and the hint part of the joint graph

- v_s – *start node*
- v_f – *final node*
- $A = A_b \oplus A_h \oplus A_c \oplus A_s \oplus A_e \oplus A_{bs} \oplus A_{hs} \oplus A_{bf} \oplus A_{hf}$
 - A_b – *base edges*
 - A_h – *hint edges*
 - A_c – *compatibility edges*
 - A_s – *starting edges*
 - A_e – *ending edges*
 - A_{bs} – *edges from start node to base nodes at the first position (beginning base edges)*
 - A_{hs} – *edges from start node to hint nodes (only to hints starting at position 1) (beginning hint edges)*
 - A_{bf} – *edges from base nodes at position k to the final node (final base edges)*
 - A_{hf} – *edges from hint nodes at position k to the final node (final hint edges)*

Nodes:

$$V_b = \left\{ v_{s,i}^{(0)} \mid \forall s, i : 1 \leq s \leq k, 1 \leq i \leq n \right\} \quad (2.10)$$

$$V_h = \left\{ v_{s,i}^{(h)} \mid \forall s, i, h : 1 \leq h \leq c, b(h) \leq s \leq e(h), i \in Q_{\ell_h(s)} \right\} \quad (2.11)$$

Edges and their weights (we use alternative notation for base nodes:

$v_{s,i}^{(0)} = v_{s,i}$), $|a|$ is weight of hint a :

$$A_b = \{a_{s,i,j} = (v_{s,i}, v_{s+1,j}) \mid t_{i,j} > 0, v_{s,i}, v_{s+1,j} \in V_b\} \quad (2.12)$$

$$|a_{s,i,j}| = \log(t_{i,j}e_{j,s_{s+1}}) \quad (2.13)$$

$$A_h = \left\{ a_{s,i,j}^{(h)} = \left(v_{s,i}^{(h)}, v_{s+1,j}^{(h)} \right) \mid t_{i,j} > 0, v_{s,i}^{(h)}, v_{s+1,j}^{(h)} \in V_h \right\} \quad (2.14)$$

$$|a_{s,i,j}^{(h)}| = \log(t_{i,j}e_{j,s_{s+1}}) + \text{subsets}(h, s) \quad (2.15)$$

$$\text{subsets}(h, s) = \sum_{\text{hint } H_u \text{ is subset of hint } H_h} [e(u) = s]B(u)$$

$$A_c = \left\{ a_{i,j}^{(h_1, h_2)} = \left(v_{e(h_1), i}^{(h_1)}, v_{e(h_1)+1, j}^{(h_2)} \right) \mid \right. \\ \left. h_2 \text{ is extension of } h_1, t_{i,j} > 0, v_{e(h_1), i}^{(h_1)}, v_{e(h_1)+1, j}^{(h_2)} \in V_h \right\} \quad (2.16)$$

$$|a_{i,j}^{(h_1, h_2)}| = \log(t_{i,j}e_{j,s_{s+1}}) + B(h_2) + \text{subsets}(h_1, e(h_1)) \quad (2.17)$$

$$A_s = \left\{ a_{i,j}^{(h)} = \left(v_{b(h)-1, i}^{(h)}, v_{b(h), j}^{(h)} \right) \mid \right. \\ \left. t_{i,j} > 0, v_{b(h)-1, i} \in V_b, v_{b(h), j}^{(h)} \in V_h \right\} \quad (2.18)$$

$$|a_{i,j}^{(h)}| = \log(t_{i,j}e_{j,s_{s+1}}) + B(h) \quad (2.19)$$

$$A_e = \left\{ \bar{a}_{i,j}^{(h)} = \left(v_{e(h), i}^{(h)}, v_{e(h)+1, j} \right) \mid \right. \\ \left. t_{i,j} > 0, v_{e(h), i}^{(h)} \in V_h, v_{e(h)+1, j} \in V_b \right\} \quad (2.20)$$

$$|\bar{a}_{i,j}^{(h)}| = \log(t_{i,j}e_{j,s_{s+1}}) \quad (2.21)$$

$$A_{bs} = \{a_j = (v_s, v_{1,j}) \mid p_i > 0, v_{1,j} \in V_b\} \quad (2.22)$$

$$|a_j| = \log(p_j e_{j,s_1}) \quad (2.23)$$

$$A_{hs} = \left\{ a_j^{(h)} = \left(v_s, v_{1,j}^{(h)} \right) \mid p_i > 0, v_{1,j}^{(h)} \in V_h \right\} \quad (2.24)$$

$$|a_j^{(h)}| = \log(p_j e_{j,s_1}) + B(h) \quad (2.25)$$

$$A_{bf} = \{\bar{a}_i = (v_{k,i}, v_f) \mid v_{k,i} \in V_b\} \quad (2.26)$$

$$|\bar{a}_i| = 0 \quad (2.27)$$

$$A_{hf} = \left\{ \bar{a}_i^{(h)} = \left(v_{k,i}^{(h)}, v_f \right) \mid v_{k,i}^{(h)} \in V_h \right\} \quad (2.28)$$

$$|\bar{a}_i^{(h)}| = \text{subsets}(h, k) \quad (2.29)$$

Used indexing:

- s – position in the input sequence $S = (s_1, s_2, \dots, s_k)$
- i – HMM state of the node, edge starts here
- j – HMM state of the node, edge ends here
- h, h_1, h_2 – hint numbers
- $p_i, t_{i,j}, e_{j,s_x}$ – start, transition, and emission probabilities (defined in 1.2.2)
- Q_l – set of all HMM states with label l
- $\ell_h(s)$ – label restriction of hint h at position s (defined in 2.1.1)

2.2.2 Best score path

Similarly to the hint path problem in 2.1.2, we are searching for a path with the highest score from the start node to the final node. Note that unlike the graph in the Section 2.1.2 here we can have negative edge weights (the logarithm of probability). Nonetheless, the algorithm for directed acyclic graphs works for negative edge weights as well. This path gives the state path, because all nodes go precisely one position in the sequence forward and thus labels belonging to nodes on the path with the highest score (except for the start and final node) create the resulting annotation.

Formally: Let $P = (v_s, v_{1,r_1}^{(h_1)}, v_{2,r_2}^{(h_2)}, \dots, v_{k,r_k}^{(h_k)}, v_f)$ be nodes on the longest path from v_s to v_f . Number of nodes in P is $k+1$, because all edges are going one position forward. States $F = (r_1, r_2, \dots, r_k)$ form the resulting state path and $F_l = (label(r_1), label(r_2), \dots, label(r_k))$ is the sequence annotation. We say that path P describes the annotation F_l and the state path F . A set of primarily respected hints is defined as $R_h = \{H_{h_i} \mid \exists j : v_{j,r_j}^{(h_i)} \in P, h_i \neq 0\}$. Let R_s be set of all subsets of hints in F_h . The goal of the algorithm is to find the annotation F_l and respected hints $F_h \cup F_s$. We can summarize this as Algorithm 2.

Algorithm 2 Given sequence S and hint set H , find a sequence annotation with the best score

1. Let $G = (V, A)$ be joint graph as defined in Definition 2.2.1.
 2. Let $P = \left(v_s, v_{1,r_1}^{(h_1)}, v_{2,r_2}^{(h_2)}, \dots, v_{k,r_k}^{(h_k)}, v_f \right)$ be nodes of the longest path from v_s to v_t in graph G
 3. For every node $v_{i,r_i}^{(h_i)} \in P$, include hint H_{h_i} and all its subsets to the set of respected hints R .
 4. Return R as the set of respected hints and $F_l = (\text{label}(r_1), \text{label}(r_2), \dots, \text{label}(r_k))$ as the sequence annotation.
-

Proof: We will use the following Lemma to prove the correctness of our algorithm:

Lemma 2.2.1 (Existence of a path in the joint graph) *For every state path $F = (r_1, r_2, \dots, r_k)$ of length k that has non-zero probability to generate sequence S there exist path P from v_s to v_t in the joint graph such that P describes the state path and $\text{length}(P) = \log(P(F, S)) + \sum_{H_i \in H \text{ is respected by } F} B(H_i)$.*

Proof: We will construct path $P = \left(v_s, v_{1,r_1}^{(h_1)}, v_{2,r_2}^{(h_2)}, \dots, v_{k,r_k}^{(h_k)}, v_f \right)$ as follows: the path will go through nodes corresponding to hints respected by the state path F and through states from F . This path is unambiguous due to the joint graph construction. All bonuses of hints respected by state path F are in weights of the path, here we can use the same arguments as in the simplified version of this problem in Section 2.1.2. HMM probabilities in log space are on every edge weight along the path. Every edge a that is not beginning nor final edge ($a \in A - A_{b_s} - A_{h_s} - A_{b_s} - A_{h_s}$) contains $\log(t_{i,j} e_{j,s_{s+1}})$, where $i, j \in Q$ are HMM states of the node where edge starts and ends, respectively. Beginning edge a ($a \in A_{b_s} \oplus A_{h_s}$) contains $\log(p_j e_{j,s_1})$. The sum of these partial edge weights is $\log(p_{r_1} e_{r_1,s_1}) \sum_{i=2}^k \log(t_{r_{i-1},r_i} e_{r_i,s_i}) = \log \left(p_{r_1} e_{r_1,s_1} \prod_{i=2}^k t_{r_{i-1},r_i} e_{r_i,s_i} \right) = \log(P(F | S))$. Adding up hint bonuses and the sum of partial edge weights gives the searched length. \square

Lemma 2.2.2 (Length of v_s - v_f path) Let $P = (v_s, v_{1,r_1}^{(h_1)}, v_{2,r_2}^{(h_2)}, \dots, v_{k,r_k}^{(h_k)}, v_f)$ be a path in the joint graph. Let F is a state path described by the path P . Then $length(P) \leq \log(P(F | S)) + \sum_{H_i \in H \text{ is respected by } F} B(H_i)$.

Proof: As in Lemma 2.2.1 we can easily prove that $length(P)$ contains all HMM probabilities ($\log(P(F, S))$), we also need to prove that it does not contain bonus of any hint that is not respected by the state path F . Bonus of a hint H_j is either on a edge leading to hint H_j or this hint is a subset of hint H_i . Thus the path goes through hint H_j or through its superset H_i . In both cases, this path describes state path F that respects hint H_j . Also it is easy to see that bonus of no hint is counted twice in the length of P . Since bonuses are positive, these two claims lead to desired inequality $length(P) \leq \log(P(F | S)) + \sum_{H_i \in H \text{ is respected by } F} B(H_i)$. \square

Now we can use the Lemma 2.2.1 and 2.2.2 to prove the correctness of our algorithm. For the optimal state path \hat{F} exists path \hat{P} through the joint graph that describe it since the optimal state path does not have zero probability of generating S (Lemma 2.2.1). Let this path be $\hat{P} = (v_s, v_{1,r_1}^{(h_1)}, v_{2,r_2}^{(h_2)}, \dots, v_{k,r_k}^{(h_k)}, v_f)$. We have $length(\hat{P}) = \log(P(\hat{F}, S)) + \sum_{H_i \in H \text{ is respected by } \hat{F}} B(H_i)$.

Thus the optimal state path is among paths considered by the algorithm. Now we have to prove only that no v_s - v_f path P such as $length(P) > length(\hat{P})$ exists in the joint graph. If such a path would exists, from Lemma 2.2.2, it describes some state path F and $\log(P(F, S)) + \sum_{H_i \in H \text{ is respected by } F} B(H_i) \geq length(P) > length(\hat{P})$, which violates the optimality of \hat{F} . \square

2.3 Time complexity

We will discuss the time complexity of whole algorithm in this section, which can be divided into two main parts: constructing the joint graph and computing the longest path in the graph. The construction of the graph is straight-forward with two exceptions: finding all extensions to the hint and counting subset bonuses. Computing the longest path is well known problem

and its best solution in directed acyclic graphs is in time linear in the number of edges. Therefore, this problem collapses into edge counting.

2.3.1 Computing the extension relation

Problem can be formulated as:

Problem 2.3.1 *Given hint H_i , find all hints that are its extensions.*

Testing if hint H_j is an extension of hint H_i can be done by the algorithm 3. Time complexity of this algorithm is $O(u)$, where u is the maximum number of intervals in a hint. We have to find all extensions for all hints therefore total time complexity is raised to $O(uc^2)$.

2.3.2 Subset bonuses

Finding subsets to a hint is very similar to finding hint's extensions. The only difference is that the extension is (by definition) not a subset and subsets have to intersect. Thus, intersecting and compatible hint H_j with the higher number than hint H_i is either an extension or a subset of the hint H_i . In fact, Algorithm 3 becomes the subset checking algorithm only by the change of the comparison mark at line 6 to " $>$ " (line 3 is then unnecessary). However, after computing subsets to hints, bonuses of these hints have to be added to the right edge weights in the graph. This can be easily done by iterating through the subset hints U_h of a hint H_h and adding their bonus to edges: for every i, j add $\sum_{u \in U_h} B(u)$ to weight of edge $a_{e(u), i, j}^{(h)}$. Time complexity: finding subsets – $O(ic^2)$, assign bonuses to edges – $O(c^2n^2)$. In fact we can assign bonuses in time $O(c^2 + cn^2 + h)$, because we first gather for every bonus all its subset bonuses, sort them to bins according to their endpoint $e(j)$ and compute a sum of bonuses in each bin. Then the precomputed bonus is simply added to each appropriate edge, which takes $O(cn^2)$ time overall.

Algorithm 3 Find if hint H_i is extension of hint H_j

Require: H_i, H_j are complex hints

```
1: if  $b(j) < b(i)$  then // wrong order of hints
2:   return false
3: end if
4: if  $b(j) > e(i) + 1$  then // not intersecting and  $b(j) \neq e(i) + 1$ 
5:   return false
6: end if
7: if  $e(j) \leq e(i)$  then //  $H_j$  is subset of  $H_i$ 
8:   return false
9: end if
10:  $pi \leftarrow 1, pj \leftarrow 1$ 
11: while  $e_{pi}^{(i)} < b_{pj}^{(j)}$  do // find first label combination
12:    $pi \leftarrow pi + 1$ 
13: end while
14: while  $pi < m_i \wedge pj < m_j$  do
15:   if  $l_{pi}^{(i)} \neq l_{pj}^{(j)}$  then // labels are not compatible
16:     return false
17:   end if
18:   if  $e_{pi}^{(i)} = e_{pj}^{(j)}$  then // move to the next label combination
19:      $pi \leftarrow pi + 1, pj \leftarrow pj + 1$ 
20:   else if  $e_{pi}^{(i)} > e_{pj}^{(j)}$  then
21:      $pj \leftarrow pj + 1$ 
22:   else
23:      $pi \leftarrow pi + 1$ 
24:   end if
25: end while
26: return true
```

2.3.3 Edge counting

The most time consuming part of the algorithm is finding the longest path and its time complexity is linear in the number of edges.

Edge type	Number
A_b – base edges	$O(kn^2)$
A_h – hint edges	$O(hn^2)$
A_c – compatibility edges	$O(c^2n^2)$
A_s – start edges	$O(cn^2)$
A_e – end edges	$O(cn^2)$
A_{bs} – from start node to bases	$O(n)$
A_{hs} – from start node to hints	$O(cn)$
A_{bf} – from bases to final node	$O(n)$
A_{hf} – from hints to final node	$O(cn)$

Table 2.1: Edge types and their numbers (c – number of hints, h – total added length of hints, n – number of HMM states, k – length of input sequence)

After summing up all the edges we get $O((k+h+c^2)n^2)$, which is the time complexity of finding the longest path as described above. Under assumption $i < n^2$ (in the real data we have $i \approx 10$ and $n \approx 260$) we can take this complexity as the total complexity of the algorithm.

2.3.4 Improvement

Total complexity can be improved to $O((k+h)n^2 + c^2n)$, which is only slight improvement, because the main part of the complexity is the underlying Viterbi algorithm ($O(kn^2)$ part) and alternative paths through hints ($O(hn^2)$ part) and these cannot be easily reduced. Nevertheless, improvement is based on reducing the A_c , A_s , and A_e sets to depend only linearly on the number of HMM states. This can be achieved by changing the starting (or end) point of these edges, so they will stay at the same sequence position. Their weight consist now only from the hint bonus (in compatibility and start edges) or it is zero (in end edges). This modification makes the joint graph cyclic in the case of the hint with the length of 1. Problem can be easily fixed by

including bonuses of hints of length 1 directly to base or compatibility edges intersecting them.

Chapter 3

Implementation and results

We have implemented the algorithm described in chapter 2 and tested it. In this chapter we describe details of the implementation, data set preparation and experimental results. The goal was to compare the contribution of different classes of the external information to accuracy of the gene finding.

3.1 Implementation

The algorithm was implemented in C++ as a standalone console application called `grapHMM`. In order to process long DNA sequence in a reasonable time and memory, we have used the improvement described in 2.3.4 as well as other small adjustments. For example, the graph is generated "on the run" and unnecessary information is immediately deleted to clean up the memory. The program has several inputs described below and one output – a sequence of state labels. However, these state labels are not corresponding 1-to-1 to searched annotation, because we used a more detailed labeling of the states for compatibility with gene finder EXONHUNTER. In this chapter we will use notation "state label" for this labeling and "hint label" for hint labels and the annotation. In general, for every hint label we have several state labels.

3.1.1 Inputs and usage

The program is used classically by running `graphHMM <arguments>` in the console. Possible arguments include:

- `-i "filename"` – input sequence in *fasta* format
- `-m "filename"` – HMM parameters in a text file format
- `-o "filename"` – the output file name (default: "out_graphmm.txt")
- `-h "filename"` – hint set (described below)
- `-l "filename"` – file with state labels for hint label conversion
- `-r "filename"` – filename for listing the set of respected hints (default: console)
- `-b "filename"` – bonuses of hints (if not found, bonuses have to be supplied in the hint set file)
- `-int` – switch, which changes the bonus for each hint by multiplying it by number of intervals in the hint
- `-sqrt` – switch; which multiplies the bonus for each hint by square root of the number of its intervals

Program writes its status messages to the console window.

3.1.2 Hint set format description

Since complex hints have not been used yet, we had to come up with a good file format for their representation. We have chosen a simple text file, where each line represents one hint. Hints from different chromosomes are divided by line with '>' sign followed by the name of chromosome, similarly as in the *fasta* format for storing DNA sequences. Hint are written in the following form:

```
<number> <bonus> <count> -  
  [<begin of interval> <hint label>](count)
```

<one position after end of last interval>

where **number** is the hint number used for testing and evaluation, **bonus** can be a string defined in the bonus file or a real number. After hyphen sign, intervals (the number of them is determined by **count**) are described with beginnings and labels. The end of the last interval is determined by the position of the following nucleotide, because we wanted to keep the description of each interval consistent: a label is surrounded by the beginning and the position one after the end of the interval. The example of the hint file can look as follows:

```
>dm3.chr2L
1 10.5 3 - 10 e 25 i 28 e 31
2 bonus 1 - 15 x 51
```

Both hints are from the chromosome `dm3.chr2L`. Hint number 1 has bonus of 10.5 and contains 3 intervals: exon from position 10 to 24, then intron from 25 to 27, and lastly, exon from 28 to 30. Hint number 2 has bonus defined under string `bonus` in the bonus file and it contains only one interval: intergenic from position 15 to 50 (example of an interval hint). If string `bonus` is not defined in the bonus file, program writes error message and quits.

This format enables to assign the same bonuses to a group of hints, what can be useful for training of the bonus values.

3.1.3 Model training and data sets

The program was tested on genomic data from *Drosophila melanogaster* (fruit fly). This species is an important model organism in genetics and its genome is relatively well sequenced and annotated. We used the same sequences and annotations as work of [Kováč, 2010]. Briefly, *Drosophila* genomic sequences and reference annotations produced by the Berkeley Drosophila Genome Project were downloaded from the UCSC genome browser [UCSC, 2006]. These sequences have 44 MB and contain 5164 genes. They were split to 2 MB parts and divided to training (≈ 28 MB, 3368 genes) and testing (≈ 16 MB, 1796 genes) sequences.

The underlying HMM has been taken from the work [Šrámek et al., 2007] and retrained on the training sequence from the *Drosophila melanogaster* genome by standard procedures [Durbin et al., 1998]. This model is described in a text file in a simple format – listing emission and transition matrices and states. We used modified HMM allowing states of higher order. In a state of order k the emission probability depends not only on the current state, but also on k last emitted symbols. Implementing higher order states into the Viterbi algorithm is straight-forward, but they require training of more parameters.

3.2 Results

We used two different approaches in creating hint sets: **hints from real data** and **artificial hint sets**.

3.2.1 Real data hint sets

The real data hint sets are from protein sequences from insect species *Drosophila Melanogaster*, *D. Simulans*, *D. Pseudoobscura*, *Anopheles Gambiae*, and *Bombyx Mori* – together 67893 protein sequences. These protein sequences were downloaded from the NCBI Protein database [NCBI, 2011c] and aligned to the target sequence using the BLAT program [Kent, 2002]. Alignments were postprocessed by scripts from ExonHunter distribution [Brejová et al., 2005, Kováč, 2010]. Namely, introns were inserted between exons and untrustworthy alignments (big number of gaps, start of an exon not on a splice site, ...) were thrown out. If the whole protein was aligned with high accuracy, a single intergenic position is put to the beginning and to the end of the complete gene hint. The resulting file is converted to our hint format by our program **HinteR**. Coverage of the hint set is displayed in Table 3.1. The filtering reduces the sum of length of alignments more than 20 times and reduces the amount of sequence covered by hints by a factor of about 7.6. Approximately 11% of the training and test sequence is covered by hints. Genes create almost 45% of the correct annotation in the test sequence, so the hints were filtered quite strictly. On

the other hand, the strict filtering raised the specificity of the search.

Type	Training	Testing
Exon – sum	6,400,436	2,710,857
Exon – union	1,814,016	1,007,198
Intron - sum	2,345,827	1,370,425
Intron – union	744,514	393,163
Intergenic – sum	14,244	3,747
Intergenic – union	3,768	1,851
All – sum	8,739,255	4,085,029
All – union	2,516,800	1,389,895
All before filtering – sum	225,181,378	87,598,103
All before filtering – union	19,276,754	10,506,537
Sequence	28,307,322	15,789,189

Table 3.1: Hint set coverage – sum represents total length of hints of some type, union represents amount of the sequence covered by these hints

We made four different hint sets. All hint sets are created from the same information set, only in some sets each hint is divided to several pieces, corresponding to the forms of external information used in existing gene finders:

- *complex hint set* – complex hints (15,057 training hints, 5,545 testing hints)
- *interval hint set* – each complex hint was divided to individual interval hints (39,900 training hints, 14,358 testing hints)
- *pointwise hint set* – interval hints with length of 1 (this file was huge due to redundant text representation, 8,651,653 training hints, 4,042,685 testing hints)
- *combined hint set* – complex hints plus their interval subparts (50,618 training hints, 17,920 testing hints)

Distribution of the number of intervals in complex hints is displayed in the Table 3.2. Two hints with 21 and 31 intervals in the training test set

are not displayed due to table size. Odd numbers of intervals are dominant, because complete gene hints have to have odd number of intervals (x exons, $x - 1$ introns, and two intergenic intervals). Even numbers (and a portion of the odd numbers) represent imperfect alignments, where only part of this alignment has been used.

Intervals	1	2	3	4	5	6	7	8	9	10	11	13
Training	4,339	1,752	7,148	205	979	57	405	8	99	1	52	10
Test	1,983	687	2,008	85	501	25	177	2	60	1	11	5

Table 3.2: Distribution of the number of intervals in complex hints

Bonuses of hints have great influence on the gene prediction accuracy: if they are too low, the hints will not help at all; if they are too high, wrong hints will decrease the gene prediction accuracy. Therefore, hint bonuses have to be trained to the optimal value. We wanted to keep the model as simple as possible, because training of a high number of parameters would require huge amount of time. Therefore, all hints in a set have the same bonus and only one parameter is trained. It is obvious that this bonus has to be trained separately for each hint set, because the bonus of, for example, pointwise hint cannot be compared to the bonus for a whole interval, or a whole complex hint. Nevertheless, we tried different modes of bonus assignment for complex hints depending on the number of their intervals:

- (no mode) – bonus is used for the whole hint
- `-int` mode – bonus is multiplied by the number of intervals
- `-sqrt` mode – bonus is multiplied by a square root of the number of intervals

The combined hint set was used only with `-int` mode, because we wanted to assign greater bonus to the longer intervals. Therefore, we performed 6 different hint bonus trainings on the training data set with exploring a range of values and choosing the value achieving the highest prediction accuracy on the training sequences. Prediction accuracy has been evaluated using the evaluation software from [Keibler and Brent, 2003]. There are different levels of evaluation:

- nucleotide (base) level – we count how many exon bases have been correctly as inside exon
- exon level – how many exons have been correctly found with both boundaries
- gene level – how many whole genes have been correctly found with all splicing sites

We evaluate sensitivity and specificity of the search at each level. Sensitivity is the ratio of the correctly predicted features (bases, exons, or genes) and the features in the reference annotation. Specificity is the ratio of the correctly predicted features and all predicted features. Average of gene specificity and gene sensitivity was chosen as the training criterion.

Evaluation on the test sequence of these different approaches is shown in Table 3.3. Difference between pointwise bonuses and interval bonuses is about 1% on the gene level and little less difference is between interval and complex hints with `-int` mode. Therefore, complex hints can help gene prediction, especially on the gene level. However, this difference is only slight compared to the jump from *ab initio* prediction to the gene prediction with external information.

Hint set	no hints	point	int	comb	compl	compl	compl
Mode	–	–	–	-int	–	-sqrt	-int
Bonus	–	3	8	7	12	6	6
Gene Sn.	48.83%	60.08%	61.08%	61.19%	60.97%	61.14%	61.86%
Gene Sp.	40.40%	45.76%	46.39%	46.14%	46.50%	46.95%	46.62%
Exon Sn.	71.18%	75.76%	76.17%	76.08%	75.99%	75.98%	76.14%
Exon Sp.	69.01%	71.72%	72.32%	72.06%	72.30%	72.47%	72.30%
Base Sn.	92.81%	94.35%	94.15%	94.20%	94.03%	93.93%	94.14%
Base Sp.	91.86%	91.98%	92.09%	92.13%	92.03%	92.07%	92.08%

Table 3.3: Comparison of different approaches on real data

3.2.2 Artificial hint sets

The second test was using artificial hint sets created from the reference annotation. We have created only two hint sets: complex and interval, because the previous test proved that other hint sets yield inferior results. Hints are chosen randomly with a fixed *length* over the sequence to achieve a fixed estimated total *coverage*. All hints are correct in the sense that they respect the known reference annotation; therefore, we expect the accuracy to grow with the bonus. Parameter *length* was set to 1000, because hints with this length seem to have the same distribution of the number of intervals as our real hint set (except for a high portion of single interval, intergenic hints). We tried different *coverage* values: 0.25, 0.5, and 2.0. Coverage of 0.25 corresponds to the real data set coverage, where only 11% of genes were covered (whole genome coverage of 0.25 in a genome with 45% gene content gives approximately 11% gene coverage).

The aim of this test is to find out the relation between bonus value and the accuracy of gene prediction. Mode `-int` has to be used for complex intervals to achieve equivalence between sums of bonuses in both hint sets. We tried bonus in range 0 to 25 with step of 1. Gene sensitivity and specificity are plotted in the graphs on Figures 3.1-3.3.

We can see, that gene prediction accuracy increases with increasing coverage and the difference between complex hints and interval hints becomes more apparent. While for 0.25 coverage, the difference is at most only about one percent (which agrees with the real data test), in greater coverage the difference raises to 2%-3% for realistic bonus value.

Finally, we assigned huge bonus to the hints and observe the maximal accuracy allowed by our model. This accuracy was not 100%, because the HMM does not allow some genomic structures (for example, exons and introns with very small length). This accuracy is in Table 3.4.

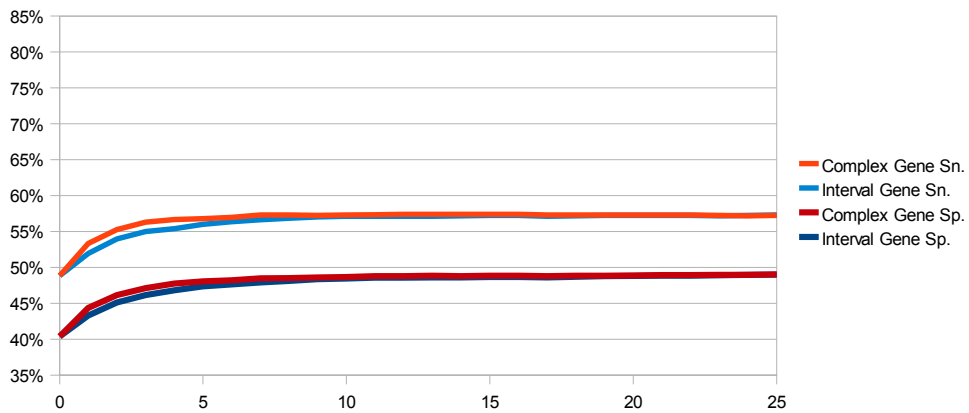


Figure 3.1: Artificial hint sets (*coverage* = 0.25) – bonus is on X axis, prediction accuracy is on Y axis

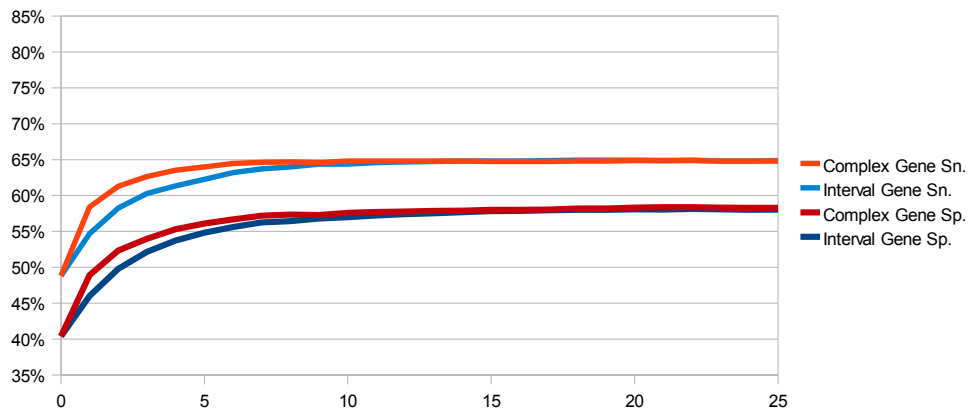


Figure 3.2: Artificial hint sets (*coverage* = 0.5) – bonus is on X axis, prediction accuracy is on Y axis

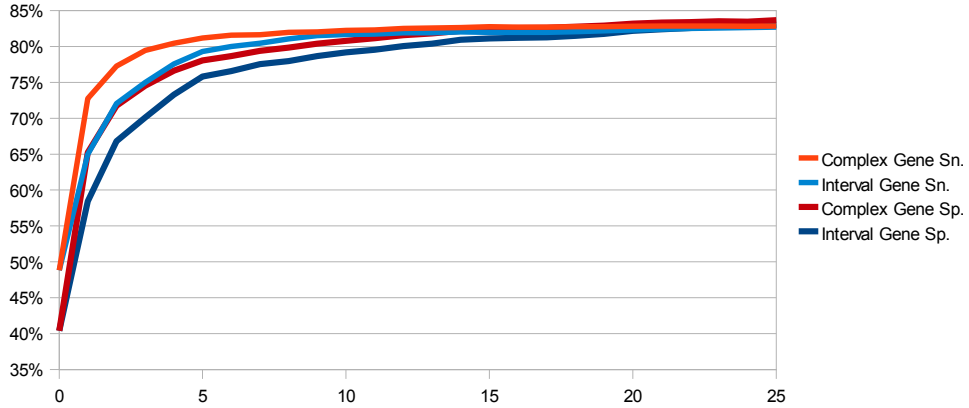


Figure 3.3: Artificial hint sets ($coverage = 2.0$) – bonus is on X axis, prediction accuracy is on Y axis

Hint set	complex	interval
Gene Sn.	89.70%	89.81%
Gene Sp.	94.28%	93.95%
Exon Sn.	87.24%	87.80%
Exon Sp.	98.28%	97.87%
Base Sn.	94.66%	94.79%
Base Sp.	99.93%	99.88%

Table 3.4: Maximal accuracy of our model achieved for very large bonus

Conclusion

We proposed a new algorithm for using external information in gene finding, which allows the use of more complex hints. The algorithm is based on finding the longest path in a special directed acyclic graph called the joint graph. We implemented a gene finder based on our algorithm and tested it. Several hint sets were created to evaluate the contribution of different forms of external information to the accuracy of gene finding. Although our implementation cannot be compared to the accuracy of existing gene finders, both artificial and real sets of hints lead to a slight increase in the accuracy for complex hints compared to simpler hint types used before. Therefore, we strongly believe that a full scale implementation of proposed algorithm with all biology motivated enhancements will result in a global gene prediction improvement.

Time complexity of this algorithm is within reasonable limits, which has been proved by both time complexity analysis and experiments on realistic data sets.

There is still space to further generalize the class of allowed external information, but some of these generalizations leads to NP-hard problems [Kováč et al., 2009]. There are open questions for future work including the following:

- Is gene finding using complex hints with multiple state restriction labels NP-hard problem? (we showed that our algorithm does not solve this case)
- If it is NP-hard, can we solve efficiently some special cases with further restrictions on hints? For example pointwise hints with multiple state

restrictions are clearly in P (such hints were used in a slightly different way in ExonHunter [Brejová et al., 2005]).

- Is gene finding with negative bonuses NP-hard problem?

Resumé

V práci popisujeme nový algoritmus na hľadanie génov, využívajúci externú informáciu. Tento algoritmus umožňuje spracovanie komplexnejšej informácie, aká bola doteraz využívaná. Popisujeme tu princípy a presnú tvorbu špeciálneho acyklického grafu, kde časti externej informácie tvoria alternatívne zvýhodnené cesty s obmedzeniami. Tento algoritmus sme aj implementovali a otestovali. Za účelom možnosti porovnania presnosti predikcie génov sme vytvorili niekoľko množín "hintov", ktoré zodpovedajú rôznym triedam externej informácie. Naša jednoduchá implementácia sa nemôže rovnať komerčným programom na hľadanie génov, avšak porovnanie rôznych množín hintov ukázalo, že komplexné hinty čiastočne zvyšujú presnosť predikcie hlavne na génovej úrovni. Tento výsledok bol dosiahnutý či už s reálnou alebo s umelo vytvorenou externou informáciou, preto veríme, že plnohodnotná implementácia programu na hľadanie génov založená na našom algoritme zvýši presnosť génovej predikcie globálne.

Navrhovaný algoritmus má časovú zložitosť porovnateľnú s dnes používanými programami, čo bolo dokázané analýzou časovej zložitosti aj experimentmi na reálnych dátach.

Bibliography

- [Brejová et al., 2005] Brejová, B., Brown, D. G., Li, M., and Vinař, T. (2005). Exonhunter: a comprehensive approach to gene finding. In *ISMB (Supplement of Bioinformatics)*, pages 57–65.
- [Burge and Karlin, 1997] Burge, C. and Karlin, S. (1997). Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.*, 268:78–94.
- [Cormen et al., 1989] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1989). *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company.
- [DeCaprio et al., 2007] DeCaprio, D., Vinson, J. P., Pearson, M. D., Montgomery, P., Doherty, M., and Galagan (2007). Conrad: gene prediction using conditional random fields. *Genome research*, 17(9):1389–1398.
- [Durbin et al., 1998] Durbin, R., Eddy, R., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis*. Cambridge Univ Pr.
- [Forney, 1973] Forney, G.D., J. (1973). The viterbi algorithm. *Proceedings of the IEEE*, 61:268 – 278.
- [Haas et al., 2008] Haas, B. J., Salzberg, S. L., Zhu, W., Pertea, M., Allen, J. E., Orvis, J., White, O., Buell, C. R., and Wortman, J. R. (2008). Automated eukaryotic gene structure annotation using EVIDENCEModeler and the Program to Assemble Spliced Alignments. *Genome Biol.*, 9(1).
- [Keibler and Brent, 2003] Keibler, E. and Brent, M. R. (2003). Eval: A software package for analysis of genome annotations. *BMC Bioinformatics*, 4:50.

- [Kent, 2002] Kent, W. J. (2002). BLAT—the BLAST-like alignment tool. *Genome Res.*, 12(4):656–664.
- [Kováč et al., 2009] Kováč, J., Vinař, T., and Brejová, B. (2009). Predicting gene structures from multiple rt-pcr tests. *Algorithms in Bioinformatics*, 5724:181–193.
- [Kováč, 2010] Kováč, P. (2010). Implementácia externých zdrojov dát v hľadani génov.
- [Krogh, 1997] Krogh, A. (1997). Two methods for improving performance of a hmm and their application for gene finding. In *ISMB*, pages 179–186.
- [NCBI, 2011a] NCBI (2011a). National Center for Biotechnology Information: database of Expressed Sequence Tags.
http://www.ncbi.nlm.nih.gov/dbEST/dbEST_summary.html .
- [NCBI, 2011b] NCBI (2011b). National Center for Biotechnology Information: ESTs Factsheet.
<http://www.ncbi.nlm.nih.gov/About/primer/est.html> .
- [NCBI, 2011c] NCBI (2011c). National Center for Biotechnology Information: Protein Data Bank (Known proteins database).
<http://www.pdb.org> .
- [Ng et al., 2005] Ng, P., Wei, C.-L., Sung, W.-K., Chiu, K. P., Lipovich, L., Ang, C. C., Gupta, S., Shahab, A., Ridwan, A., Wong, C. H., Liu, E. T., and Ruan, Y. (2005). Gene identification signature (GIS) analysis for transcriptome characterization and genome annotation. *Nature Methods* 2, pages 105–111.
- [Ouzounis and Valencia, 2003] Ouzounis, C. A. and Valencia, A. (2003). Early bioinformatics: the birth of a discipline - a personal view. *Bioinformatics*, 19(17):2176–2190.
- [Paul E. Black, 2008] Paul E. Black (2008). hidden Markov Model.
<http://www.nist.gov/dads/HTML/hiddenMarkovModel.html>.

- [Šrámek et al., 2007] Šrámek, R., Brejova, B., and Vinař, T. (2007). On-line Viterbi Algorithm for Analysis of Long Biological Sequences. In Giancarlo, R. and Hannenhalli, S., editors, *Algorithms in Bioinformatics: 7th International Workshop (WABI)*, volume 4645 of *Lecture Notes in Computer Science*, pages 240–251, Philadelphia, PA, USA. Springer.
- [Sakharkar et al., 2004] Sakharkar, M. K., Chow, V. T., and Kanguene, P. (2004). Distributions of exons and introns in the human genome. *In Silico Biology* 4.
- [SIB, 2011] SIB (2011). Swiss Institute of Bioinformatics: UniProt Knowledgebase – protein sequence database.
<http://expasy.org/sprot/> .
- [Stanke et al., 2008] Stanke, M., Diekhans, M., Baertsch, R., and Haussler, D. (2008). Using native and syntenically mapped cdna alignments to improve *de novo* gene finding. *Bioinformatics*, 24(5):637–644.
- [Stanke et al., 2006] Stanke, M., Schoffmann, O., Morgenstern, B., and Waack, S. (2006). Gene prediction in eukaryotes with a generalized hidden markov model that uses hints from external sources. *BMC bioinformatics* 2006, 7:62.
- [UCSC, 2006] UCSC (2006). *Drosophila melanogaster* draft assembly (BDGP Release 5), provided by the Berkeley *Drosophila* Genome Project(BDGP). *Online*.
<http://genome.ucsc.edu/cgi-bin/hgGateway?clade=insect&org=0&db=0>.
- [Yeh et al., 2001] Yeh, R.-F., Lim, L. P., and Burge, C. B. (2001). Computational inference of homologous gene structures in the human genome. *Genome Res.*, 11:803–816.