Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Gene Finding Using Rt-pcr Tests

Master's Thesis

Jakub Kováč

Supervisor: Mgr. Broňa Brejová, PhD.

Bratislava, 2009

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.

. . . .

Acknowledgements

I am very grateful to Tomáš Vinař and Broňa Brejová for their advice, support and introducing me to the world of biology.

Abstract

We formulate and study a new problem in bioinformatics. This problem can be formalized as finding the highest score s–t path in a graph. We are given lengths for certain pairs of vertices. If the path passes through a pair of vertices and the length of the subpath between these vertices has given length, we get bonus (otherwise we are penalized). We prove that several variants and subproblems are NP-hard and for some special cases we propose efficient algorithms

The problem is (weakly) NP-hard even for a single pair, but can be solved in pseudopolynomial time; we also propose an approximative solution. Then we study a problem of finding an s–t path that avoids all the "forbidden" pairs of vertices or finding a path that collects "useful" pairs of vertices. These are the special cases of our problem (if we forget about scores and lengths) and turn out to be the hard core of our problem. Complexity of these problems depends to a large extent to the mutual positions of the pairs. We introduce 7 natural classes of sets of pairs for which we try to solve the problems. We prove the problem (strongly) NP-hard for some classes and we propose efficient algorithms for other classes (thus we study the borderline between the NP-hardness and efficient solvability).

Contents

Acknowledgements **v**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Previous Work and Statement of the Problem . . . 2
1.3 Our results . . . 2

2 Biological problem **5**
2.1 Brief introduction to molecular biology . . . 5
2.1.1 Proteins . . . 5
2.1.2 DNA . . . 5
2.1.3 Gene . . . 6
2.1.4 Protein synthesis . . . 7
2.2 Size and complexity of the human genome . . . 9
2.3 Gene prediction . . . 9
2.3.1 Statement of the Problem . . . 9
2.4 Introduction to RT-PCR tests . . . 10
2.4.1 Polymerase Chain Reaction and RT-PCR . . . 10
2.4.2 RT-PCR as a black box . . . 11

3 Formulation of the Problem **15**
3.1 Definitions of the Basic Terms . . . 15
3.2 Exon graph . . . 17

4 Algorithms and Hardness Results **19**
4.1 Single RT-PCR Test . . . 19
4.1.1 Bad News . . . 19
4.1.2 Good News . . . 20
4.2 Multiple RT-PCR Tests and Alternative Splicing . . . 23
4.2.1 Multiple RT-PCR Tests . . . 23
4.2.2 Length Inference . . . 24
4.3 Useful and Forbidden Pairs . . . 27
4.3.1 Bad news . . . 31
4.3.2 Good news . . . 35
4.4 More on Multiple RT-PCR Tests . . . 40

5 Conclusion **45**

Chapter 1

Introduction

1.1 Motivation

Since April 2003, 99% of gene-containing part of human sequence has been determined to 99.99% accuracy. However, without further analysis, this is just a string of 3 billion A's, C's, G's and T's. Gene finding i.e., finding the regions of DNA that encode proteins is then one of the first and most important steps in deriving some meaningful knowledge from a DNA sequence.

Due to its vast size, it would be very time consuming to analyze whole genome manually. Therefore algorithms and methods have been developed that can predict the location and exon structure of genes. A gene is a region or a substring of DNA, which encodes a protein or a functional RNA molecule. However, only some parts of the substring, called exons, are actually translated to a protein; the remaining parts, called introns, are discarded in the process of protein synthesis. In the process of gene prediction we are interested in discovering exons as disjoint intervals of the genome (we explain more of the biology in Chapter 2). Methods to predict exons and introns are based largely on probabilistic models of gene structure, particularly hidden Markov models (HMMs). Such a model can for example capture known differences between the frequencies of some letters in coding and non-coding parts of DNA, certain patterns that are usually present at the boundaries between introns and exons, etc. These so called *ab initio* ("from the beginning") gene finders use only the genomic sequence, without additional extrinsic information. They can be further extended to incorporate various forms of external evidence (for example database of known proteins). There are even methods for combining outputs of different gene finders.

The gene finders are evaluated by comparing the predictions with experimentally validated known genes and it is done on three levels – on the nucleotide level we look at the individual letters and check how many of them were classified correctly as coding or non-coding; on the exon level we look at the individual exons and count how many of the predicted exons are exactly identical to the real ones and at the gene level we find how many genes were predicted entirely correctly. Although gene finders are very good at a nucleotide level, it is still very hard to predict entire gene with all exon–intron boundaries exactly right (Guigo et al., 2006).

1.2 Previous Work and Statement of the Problem

In this thesis we try to improve the process of gene prediction by considering an additional external information – results of an RT-PCR test, which basically tell us the sum of lengths of the exons in a certain region of DNA. It was inspired by a paper by Agrawal and Stormo (2006), which used a simple heuristic to find good transcripts of given length even in presence of alternative splicing. In a small experiment with 151 regions of DNA with alternative splicing they compared their solution with an ab initio gene prediction program called Genefinder. Genefinder predicted one of the two correct product 65% of the time. Compared to this, the Locus software ("Length Optimized Characterization of Unknown Spliceforms") by Agrawal and Stormo (2006) predicted at least one of the two products 97% of the time and both products 64% of the time. Moreover, when considering also the second through fourth best solutions, the ability to predict both products increased to 72–75% and considering the ten best solutions to 80% (in this case, although at the cost of lower specificity, 89% of all gene products are accurately predicted).

We try to improve on this result by

- solving the underlying algorithmic problems exactly and
- generalizing it to more RT-PCR test results.

We propose to use a gene finder that outputs putative exons in a DNA sequence. We generate enough putative exons so that with high probability they will contain all the true ones; we also expect an abundance of false positives, which are to be filtered. These exons may be thought of as vertices in a graph where edges represent possible introns and paths represent possible transcripts. Each vertex has an associated length and each edge has an associated score. The length of a transcript is the sum of lengths of its exons and score of a transcript is the sum of scores of its introns. We generally try to find paths with the highest scores. The RT-PCR test result as an additional information tells us, what length should a path have, if it goes through some pair of vertices.

1.3 Our results

We state and study a new problem in computational biology. The bad news are that the problem in its full generality and even several subproblems are NP-hard. In Section 4.1 we prove that the problem is NP-hard (but not strongly NP-hard) even for a single RT-PCR test result; in Section 4.2 we prove that more RT-PCR tests and alternative splicing makes situation even worse.

On the other hand, we show that there is a pseudopolynomial algorithm for a single RT-PCR test. Furthermore, the lengths cannot be measured very precisely anyway and in Section 4.1 we give an algorithm that finds high score paths (with certain guarantees) of approximate length.

In Section 4.3 we study the problems of avoiding forbidden or passing useful vertex pairs in a directed acyclic graph, which seems to be at the core of the complexity of our problem. The problem of avoiding forbidden pairs is to find a path between two vertices in a graph that avoids all the forbidden pairs of vertices (i.e., a path that contains at most one vertex from each forbidden pair). A very similar problem is that of passing useful pairs: Given a graph with a set of useful pairs the problem is to find a path that “collects” or passes through as many useful pairs as possible. It turns out that the complexity of the problem depends largely on the mutual positions of the pairs. We study these problems in depth proving some subproblems to be NP-hard and showing effective algorithms for some special cases.

The effective algorithms for certain special cases of the useful pairs problem can then be generalized to a pseudopolynomial solution for some special cases of our problem.

Chapter 2

Biological problem

In this chapter we will introduce necessary background from molecular biology and describe the problem of gene finding which is the focus of our thesis. We will also introduce RT-PCR technology, which is typically used for verification of computational gene predictions but which we will use to guide such predictions.

2.1 Brief introduction to molecular biology

2.1.1 Proteins

Proteins are the main building blocks and functional molecules of cells. They take up almost 20% of an eukaryotic cell's weight, which is the largest contribution after water (70%) (Brazma et al., 2001).

Different types of proteins play a crucial role in almost every biological process. Proteins called enzymes catalyze (increase the rate of) biochemical reactions. They are able to alter, join or chop up other molecules. There are structural proteins such as collagen, which strengthens skin and bones. Transmembrane proteins are responsible for maintaining cellular environment and regulating cell volume. Proteins actin and myosin provide for muscle contraction; hemoglobin transports oxygen to tissues, certain proteins control growth and cell differentiation, start or repress DNA expression and so on and so forth.

Proteins are polymers – macro-molecules consisting of a chain of simple units (amino acids) connected by peptide bonds. There are 20 different amino acids that can be found in proteins, each having its one-letter code. Thus if we do not take into consideration the three-dimensional structure of proteins, we can encode them as strings over the alphabet $\Gamma = \{\mathtt{A}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{V}, \mathtt{W}, \mathtt{Y}\}$.

2.1.2 DNA

DNA, deoxyribonucleic acid, stores an organism's genetic information and controls the production of proteins. It is thus responsible for the biochemistry of an organism.

A DNA strand is a long chain of small molecules, called nucleotides, linked by phosphate ester bonds. Each nucleotide consists of phosphate, sugar (a 2-deoxyribose in case of DNA) and an amine base. The alternating phosphate and sugar residues form a backbone, and there is an amine base attached to each sugar. There are four amine bases: adenine, cytosine, guanine and thymine. Nucleotides are denoted by letters A, C, G and T, respectively according to the bases. Despite the complex three-dimensional structure of DNA, it appears that the genetic material it stores only depends on the sequence of nucleotides. Thus it can be modelled as a string over the alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. This string is refered to as a DNA sequence and may be obtained by a process called DNA sequencing.

Usually DNA does not exist as a single molecule, but rather as a pair of complementary DNA strands winding around each other forming the well-known double helix (a right-handed spiral). Two DNA strands are complementary (or antiparallel), if one can be obtained from the other by mutually exchanging all the A's with T's and C's with G's, and changing the direction of the molecule to the opposite. The A–T and C–G pairs are called (complementary) base pairs (abbreviated bp). This complementarity is exploited e.g. during DNA replication – two strands of the DNA unwind and new complementary strands are synthesized.

2.1.3 Gene

In the cells, DNA is "packaged" in several chromosomes (e.g. humans have 24 chromosomes). The total complement of genetic material (DNA from all chromosomes) is called a genome. The genome of an organism contains templates of molecules that are necessary for life – proteins and RNA (however, we will consider only proteins). A region of the genome that encodes a single protein is called a gene (see Fig. 2.1). Proteins are created from genes by a complicated process called protein synthesis (or more generally, gene expression).

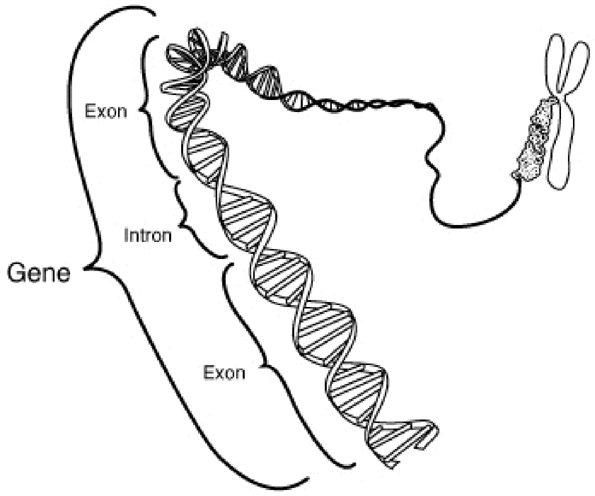


Figure 2.1: Gene is a region of a chromosome encoding one protein. Chromosome is made of two complementary DNA strands. Source: http://en.wikipedia.org/wiki/File:Gene.png

2.1.4 Protein synthesis

The process of protein synthesis can be divided into three phases: transcription, splicing and translation (see Fig. 2.2).

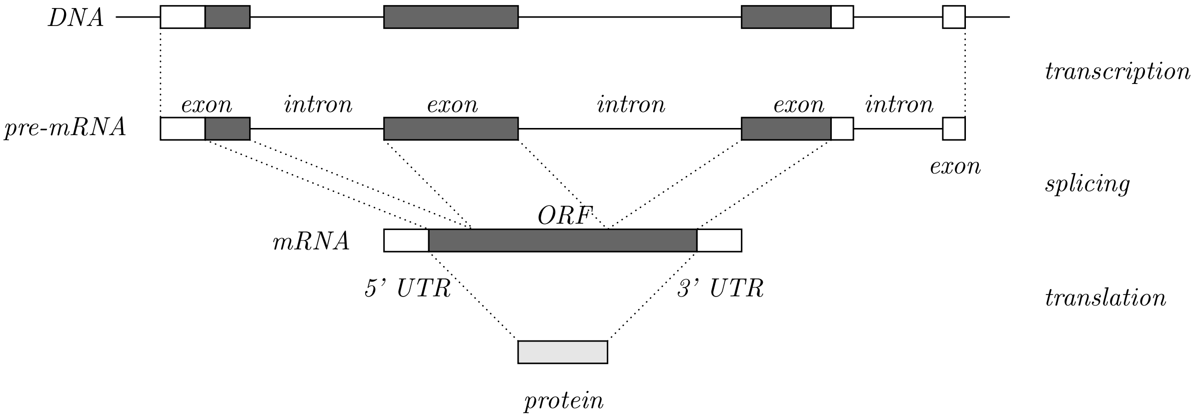


Figure 2.2: Protein synthesis: In the first step a region of DNA is transcribed onto a pre-mRNA. The introns are discarded in a process called splicing. Term *alternative splicing* then refers to the fact that this may be done in several ways. The last step is translation, where the resulting protein is constructed based on the triplets of nucleotides in mRNA. The white block represents a region in the beginning and at the end of mRNA called UTR – untranslated region. The part of sequence that encodes protein is called open reading frame (ORF). Although technically, exons may contain a part of a UTR, by exons we will mean simply the coding parts of exons (the gray blocks).

In the first phase the DNA is temporarily unwound and the region encoding a protein, the gene, is copied (transcribed) into pre-mRNA (prefix "pre-" is for "preliminary"). This pre-mRNA contains the same information as the gene (although in complementary form). However, in eukaryotic organisms (such as animals and plants) protein is usually not encoded by the whole gene – the gene consists of coding parts called exons and non-coding parts called introns. Introns are cut out in the second phase in a process called splicing. The introns are removed and exons are joined together forming an mRNA. The "m" is for "messenger", since in eukaryotic cells the DNA is contained in cell's nucleus and proteins are made in cytoplasm outside the nucleus. Thus mRNA is used for transporting the information.

Then, in cytoplasm, the mRNA is translated into protein by cell machinery (by protein-RNA complex ribosome, other RNA, etc.). Recall that RNA is a chain of nucleotides (string over 4 letters alphabet) and protein is a chain of amino acids (string over 20 letters alphabet). Each triplet of nucleotides in mRNA, called codon, encodes one amino acid according to the so called genetic code (see Tab. 2.1 and Fig. 2.3). Since there are $4^3 = 64$ codons and only 20 amino acids, many amino acids are encoded by several different triplets. Each protein sequence starts with methionine (`M`) encoded by `AUG` and ends with one of the stop codons `UAA`, `UAG`, or `UGA` (denoted by asterisk in Tab. 2.1).

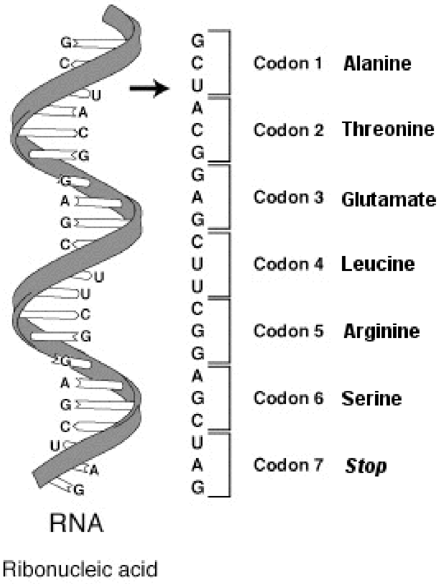


Figure 2.3: During the translation a protein encoded in mRNA is constructed; each triplet of the mRNA bases encodes one amino acid of a protein. Thus the mRNA code is "translated" into the "protein" code. Table 2.1 summarizes which triplets code for which amino acids. Source: `http://en.wikipedia.org/wiki/File:Rna-codons-protein.png`

UUU	→	F	UUC	→	F	UUA	→	L	UUG	→	L
UCU	→	S	UCC	→	S	UCA	→	S	UCG	→	S
UAU	→	Y	UAC	→	Y	UAA	→	*	UAG	→	*
UGU	→	C	UGC	→	C	UGA	→	*	UGG	→	W
CUU	→	L	CUC	→	L	CUA	→	L	CUG	→	L
CCU	→	P	CCC	→	P	CCA	→	P	CCG	→	P
CAU	→	H	CAC	→	H	CAA	→	Q	CAG	→	Q
CGU	→	R	CGC	→	R	CGA	→	R	CGG	→	R
AUU	→	I	AUC	→	I	AUA	→	I	AUG	→	M
ACU	→	T	ACC	→	T	ACA	→	T	ACG	→	T
AAU	→	N	AAC	→	N	AAA	→	K	AAG	→	K
AGU	→	S	AGC	→	S	AGA	→	R	AGG	→	R
GUU	→	V	GUC	→	V	GUA	→	V	GUG	→	V
GCU	→	A	GCC	→	A	GCA	→	A	GCG	→	A
GAU	→	D	GAC	→	D	GAA	→	E	GAG	→	E
GGU	→	G	GGC	→	G	GGA	→	G	GGG	→	G

Table 2.1: The genetic code specifies which triplets of RNA bases (called codons) code for which amino acids. Codon `AUG` is refered to as start codon; codons `UAG`, `UAA` and `UGA` are the stop codons which signify the end of translation.

If we map the regions of an mRNA that are translated back to the original DNA, we get several non-overlapping intervals called coding regions. We will call them simply exons (even if not technically correctly).

2.2 Size and complexity of the human genome

To get an idea of the size of human genome, let us list some of its parameters.

- Human genome consists of 24 chromosomes (22 types of autosome and two sex chromosomes, X and Y).
- The chromosomes have variable length ranging from 46 to 245 MB (read "mega bases", that is 10^6 bases). In total, human DNA consists of approximately 3 billion ($3 \cdot 10^9$) base pairs (Strachan and Read, 1999).
- It is estimated that the human genome contains 20 000–25 000 genes; only about 1.5% of the DNA sequence is coding proteins. The rest are RNA genes, regulatory sequences, introns and highly repetitive sequences.
- The average number of exons in a human gene is about 8–10, with the mean value of 8.8 exons per gene. The average exon length is 170 base pairs (with standard deviation 192–397 on different chromosomes), while 80–85% exons on each chromosome were found to be less than 200bp (base pairs) in length (Sakharkar et al., 2004).
- The average intron size is 5419bp (however, with much higher standard deviation 4741–23528). About 5% of introns are more than 200 000bp. On the other hand less than 0.01% of introns are less than 20bp long (Sakharkar et al., 2004).
- An average human gene contains about 6–9 introns, with the mean value 7.8 introns per gene; this number varies from 0 in about 3 362 genes (the single exonic genes) to 147 introns of nebulin (Sakharkar et al., 2004).

2.3 Gene prediction

2.3.1 Statement of the Problem

The goal of gene prediction is to detect the coding regions of a given DNA sequence. The sequence usually contains several genes on each strand – the goal is to predict coding regions of every gene on both strands. Each coding region can be unambiguously specified by giving the strand, position, length and reading frame. The reading frame specifies the position (modulo 3) of the first base within a codon – this is needed because introns can separate one codon into two exons; thus a coding region does not need to start with a full codon.

The gene prediction problem is usually formalized as follows:

Problem 0 (The gene prediction.). Given a DNA sequence $x = x_1 x_2 \dots x_n \in \Sigma^*$, find the labeling $\lambda = \lambda_1 \lambda_2 \dots \lambda_n \in \Lambda^*$, corresponding to the correct gene structure of all genes in x. $\Lambda = \{\texttt{0}, \texttt{1}, \texttt{2}, \texttt{i}, \texttt{x}\}$, where `0`, `1` and `2` denote the reading frame in a coding region (exon), `i` denotes an intron and `x` denotes an intergenic region (non-coding region between two genes).

However, the problem of gene finding is actually more complicated than the definition suggests. Previously scientists believed in a ‘one gene → one protein’ dogma. This turned up not to be true.

- First, genes encode not only proteins but also RNA (tRNA, ribosomal RNA, etc.).
- Second, the synthesized proteins and even the pre-mRNA (after the first phase) may be modified in several ways (so called post-translational and post-transcriptional modification).
- Third, two genes may overlap – this is prevalent in the short genomes of viruses and bacteria, however, hundreds of overlapping genes were discovered even in the human genome.
- And fourth, sometimes the mRNA can be spliced in more than one way – when a gene is being expressed, different regions may be cut out (as introns), leading to different mRNA molecules and ultimately different proteins. This is called alternative splicing.

Most of the today’s software solves Problem 0, predicting only one splicing variant per gene and ignoring overlapping genes. However, in the future, both computer scientists and biologists will have to cope with the above-mentioned problems – under these conditions, even the definition of a gene is unclear (Gerstein et al., 2007).

2.4 Introduction to RT-PCR tests

Gene finding software tools often make errors in their predictions and therefore we need techniques for experimentally verifying predicted genes. RT-PCR tests described in this section are one of the standard ways for achieving this goal. In this thesis we propose ways for using RT-PCR not only for verification but also for improving quality of gene prediction.

2.4.1 Polymerase Chain Reaction and RT-PCR

The polymerase chain reaction (PCR) is a widely used technique of molecular biology. With PCR it is possible to make millions or more copies of a piece of DNA, starting with only one or few copies of this piece.

A basic PCR setup requires several components and reagents, including

- DNA template that contains the DNA region (target) to be amplified,
- many copies of two primers (usually short strands of complementary RNA, which serves as a starting point for DNA replication),
- DNA polymerase – an enzyme that assists in DNA replication,
- dNTPs – the building blocks from which a new DNA strand is synthesized,
- and several other components.

The procedure consists of alternating heating and cooling: when DNA is heated, the hydrogen bonds that hold the two strands together melt and DNA unwinds. Then the temperature is lowered allowing annealing of the primers to the single-stranded DNA template. The DNA polymerase binds to the primer-template hybrid and begins DNA synthesis. It synthesizes a new DNA strand complementary to the DNA template strand by adding dNTP's from the environment. This makes up one cycle, which is repeated 20–35 times. Note, that in each cycle the number of copies of the DNA template doubles (if all goes well); thus the region between the two primers is exponentially amplified (see Fig. 2.4; Cooper and Hausman (2004)).

Reverse Transcription PCR (RT-PCR) is a variant of polymerase chain reaction where an RNA strand is first reverse transcribed into its DNA complement (cDNA) using the enzyme reverse transcriptase. Typically, RT-PCR is applied to a mixture of mRNA molecules extracted from some tissue with the goal to confirm expression of a particular gene containing the two primers in its exons.

After the amplification, the RT-PCR product is loaded onto an agarose gel for electrophosphoresis. The electromotive force is used to move the products through the gel matrix. This technique is based on a principle that the molecules with lower length have lower weigth and thus (with the same momentum) travel longer distances in the gel matrix (the distance they travel is approximately inversely proportional to the logarithm of their size). We can make the molecules visible (see Fig. 2.5) and using *markers* – a mixture of molecules of known size – the lengths of the RT-PCR products may be estimated by comparison with the markers.

2.4.2 RT-PCR as a black box

In the previous section the principles behind RT-PCR were explained. However, it is sufficient for us to treat an RT-PCR experiment as a black box.

Actually we may treat an RT-PCR test in three different ways as a gradually stronger and stronger black box:

1. Given a pair of primers (or simply two positions on a DNA strand), an RT-PCR test says `YES`, if there is a transcript with both primers contained in some of its exons; otherwise the test says `NO` (in other words, the test says `YES`, if there is a product of some transcript at the end of the test). This is the weakest RT-PCR black box; we shall actually use this model in Section 4.3.

2. Given a pair of primers, an RT-PCR test outputs the lengths of all the products (estimated from the electrophosphoresis). This is the middle RT-PCR black box, which is our main concern in this thesis.

3. The strongest version of an RT-PCR black box sequences all the products, so we get to know not only lengths of the products, but the specific nucleotides from which the products are comprised. After aligning such a sequence to the genome, we will obtain exact boundaries of all exons between the two primers. However, the subsequent sequencing incurs further expenses and we do not consider it in this thesis.

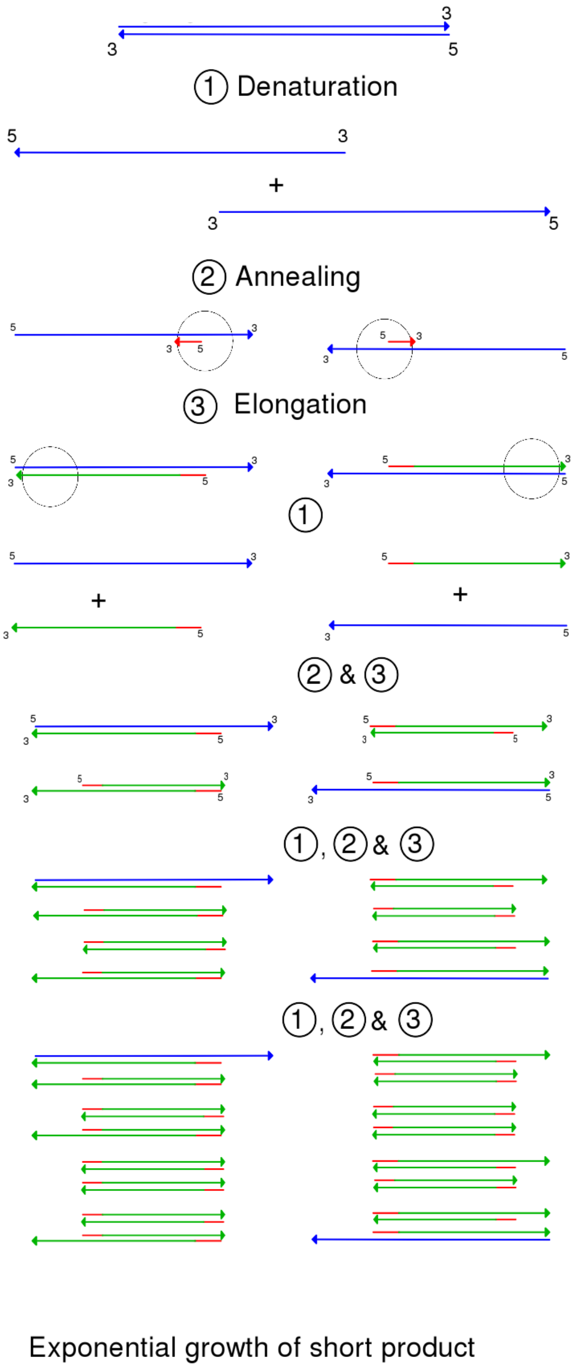


Figure 2.4: Polymerase Chain Reaction: In phase 1 the mixture is heated and the double helix of DNA unwinds; in phase 2 it is cooled down and the primers adhere to the DNA strands; in phase 3 the DNA is copied by DNA polymerase; these 3 steps are repreated several times; in each iteration the number of copies containing the region between the two primers doubles. Source: `http://upload.wikimedia.org/wikipedia/commons/8/87/PCR.svg`

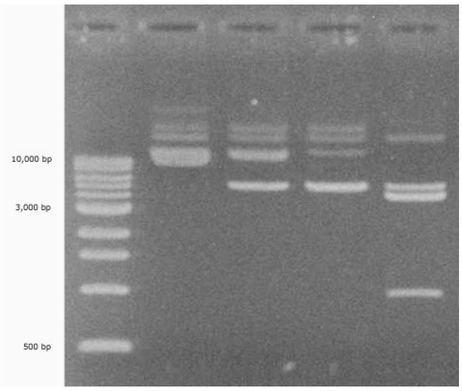


Figure 2.5: Electrophosphoresis: The molecules are moved by electromotive force; the shorter they are, the longer distance (from the top) they travel. In the first lane there is a so called DNA ladder containing markers of known size; other four lanes show variously-sized DNA fragments. By comparison with the first lane we can estimate their sizes. Source: `http://en.wikipedia.org/wiki/Image:AgaroseGel.jpg`

Chapter 3

Formulation of the Problem

3.1 Definitions of the Basic Terms

In this chapter we translate the problem into the language of computer science and formulate the biological problem as an algorithmic problem. In particular, we define a result of an RT-PCR test and we model the putative exons and introns as a graph and transcripts as paths in this graph. Thus we state the problem as a graph-theoretical one.

Exons (the coding regions) are basicaly substrings of a DNA sequence so if the DNA sequence is fixed, we can represent them as intervals, giving their starting and end positions. Then a transcript is composed of several such exons (at least one).

Definition 3.1 (Transcript). *Transcript* is a nonempty sequence of disjoint intervals. Intervals of a transcript are called *exons* and intervals between exons (from the end of one exon to the beginning of the following exon) are called *introns*.

For example Fig. 3.1 shows 4 transcripts. The first and the last ones consist of 4 exons and 3 introns, the middle ones have just 3 exons (gray bars are exons, lines between them are introns).

As we mentioned in Chapter 2, we treat an RT-PCR test as a black box, which given position of the two primers in a DNA sequence returns the length of a trancript between the two primers. More generally, if we take alternative splicing into account, there may be more such transcripts which may (and may not) have different lengths. On the other hand there may be no such transcript in which case we do not get any length.

Definition 3.2 (Result of an RT-PCR test). The *result of an RT-PCR test* is a tuple $(p_1, p_2, \langle m_1, M_1\rangle, \langle m_2, M_2\rangle, \ldots, \langle m_k, M_k\rangle)$, where p_1 and p_2 are positions of the primers ($p_1, p_2 \in \mathbb{N}$, $p_1 < p_2$) and pairs $\langle m_i, M_i\rangle$ are the measured lengths (minimum and maximum). We treat the measured lengths as intervals to model uncertainty stemming from imprecision of measurement. Lengths $\langle m_i, M_i\rangle$ and $\langle m_j, M_j\rangle$ from the same RT-PCR test are disjoint intervals for $i \neq j$. On the other hand we treat primers as points for simplicity.

We will consider mostly results without alternative splicing having at most one measured length, i.e. (p_1, p_2) or $(p_1, p_2, \langle m, M\rangle)$.

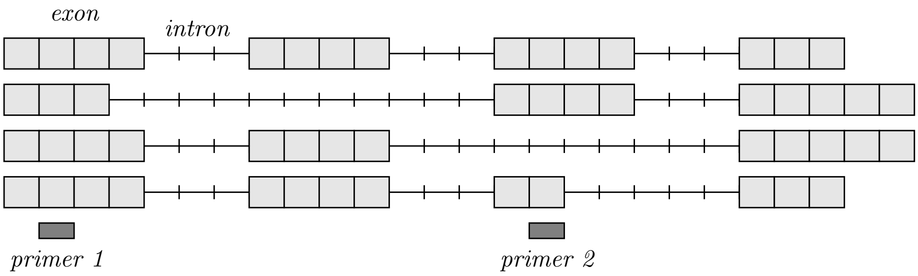


Figure 3.1: This figure shows 4 transcripts; the gray bars represent exons, the lines between them are introns; one tick represents 10 bases. If we carry out an RT-PCR test with the positions of primers as shown, we get products of two different lengths. The first and the last transcripts will both yield a product of length approximately 90 bases, the second transcript will have a product cca 40 bases long and the third one does not produce any product, since the second primer is not contained in any of its exons. Thus the result of the RT-PCR test may be (if we account for imprecission of measurement), say $(2, 16, \langle 37, 42\rangle, \langle 83, 105\rangle)$.

Example 3.1. Fig. 3.1 shows an RT-PCR test where we place the primers at positions 2 and 16. Assume that the only possible transcripts are the ones shown. Then the third transcript does not have any product, because it does not contain the second primer. Imagine that one tick or one bar represents 10 bases in the DNA sequence. Then the length of the product from the second transcript is approximately 40 bases so the measured length may be say $\langle 37, 42\rangle$. The length of the product from the first and the last transcript is approximately 90 bases and we may measure e.g. $\langle 83, 105\rangle$. Thus we can write the result of this RT-PCR test as $(2, 16, \langle 37, 42\rangle, \langle 83, 105\rangle)$.

Note that this test will not differentiate between the first and the last transcript and will not even indicate that there is more than one transcript with the given length. All we know is that there is at least one transcript of length approximately 90 bases. The same applies to the second transcript: we cannot be sure that there is only one transcript of the given length. On the other hand, another RT-PCR test with different positions of primers may succeed in discriminating between the two.

Remark 3.1. Note that all the positions and lengths in a DNA sequence are integers, so what we really mean by an interval $\langle a, b\rangle$ in the definitions of exons and results of RT-PCR tests is $\langle a, b\rangle \cap \mathbb{Z} = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$.

Now let us define, what does it mean to say that certain transcript *explains* some result of an RT-PCR test and what does it mean to say that some transcript is *consistent with* some result of an RT-PCR test.

Definition 3.3 (Explanation, Consistency). We say that a transcript *explains* length $\langle m_i, M_i\rangle$ from the result of some RT-PCR test, if it contains both primers and sum of lengths of exons between the primers is in the interval $\langle m_i, M_i\rangle$. Formally: transcript $\langle e_1, e'_1\rangle, \langle e_2, e'_2\rangle, \ldots, \langle e_n, e'_n\rangle$ explains the i-th length from the test $(p_1, p_2, \langle m_1, M_1\rangle, \ldots, \langle m_k, M_k\rangle)$, if there are indices q and

r such that $p_1 \in \langle e_q, e'_q \rangle$, $p_2 \in \langle e_r, e'_r \rangle$ and

$$m_i \le (e'_q - p_1) + (p_2 - e_r) + \sum_{j=q+1}^{r-1} (e'_j - e_j) \le M_i.$$

We say that a transcript is *consistent* with the result of an RT-PCR test, if it either explains some length of the test, or if it does not contain one of the primers (otherwise it is *inconsistent*).

Example 3.2. The first and the last transcript on Fig. 3.1 both explain length $\langle 83, 105 \rangle$. Transcript 3 is consistent with that RT-PCR test result, because it does not match the second primer.

In the problems we solve we are usually given a set of *putative* exons and the transcripts are unknown. Naturaly not every combination of exons forms a transcript consistent with all RT-PCR test results. We seek to find transcripts that explain at least some results, but they should be consistent with every result.

3.2 Exon graph

Let us formulate the problem in the language of graph theory. We consider a graph (which we call an *exon graph*), where vertices represent possible exons and edges represent possible introns. For example if two exons are not disjoint intervals in the DNA sequence, there is no edge between them. On the other hand, if two exons are too far appart, it is improbable that the first one is directly followed by the second one in some transcript, so again there is no edge between the two. Since the exons are ordered on the DNA from “left to right” (e.g. by the starting position), the graph is naturally directed and acyclic – there is an edge going from exon e to e' if the two exons are disjoint (as intervals), e is to the left of e' and e' may directly follow e in some transcript. In this graph any path corresponds to a putative transcript.

Moreover, for each vertex we have length $\ell(v)$ which is the length of the corresponding exon and for each vertex and edge we have its score. Scores of the exons are determined from the result of a gene finder – vertices with higher scores are more probable (more probably correct). We can have also scores for introns based e.g. on the distribution of their lengths (introns of certain length are more probable than too short or too long ones). Actually it is sufficient (and simpler) to have scores just for edges and carry the scores of vertices over to their out-going edges.

Since the graph is directed and acyclic, we may sort it topologically. For simplicity we also add special starting and terminal vertices s and t (as the first and the last in topological order) with edges into/from each vertex that represents an exon which can serve as a possible first or last exon of a transcript, respectively (this information is also obtained from a gene finder). Consequently we can focus solely on the s–t paths.

Definition 3.4 (Exon graph). *Exon graph* is a directed acyclic graph $G = (V, E, \ell, S, s, t)$, where $\ell : V \to \mathbb{N}$ is a function which assigns to each vertex v the *length* $\ell(v)$ of the exon it represents and $S : E \to \mathbb{Q}^+$ is a function which assigns to each intron – edge e – its *score* $S(e)$.

We will assume that the graph is topologically sorted and all its vertices are $s = v_0, v_1, \ldots, v_n, v_{n+1} = t$ in that order. In this setting a *result of an RT-PCR test* is a tuple $(p_1, p_2, \langle m_1, M_1\rangle, \langle m_2, M_2\rangle, \ldots, \langle m_k, M_k\rangle)$, where $p_1, p_2 \in V$ are vertices corresponding to the primers. *Transcript* is an s–t path in G and its length is a sum of lengths of its vertices and its score is sum of the scores of its edges.

We would like to find transcripts with high score which explain some lengths and which are consistent with as many RT-PCR tests as possible. To model this, transcript gets bonus $B \in \mathbb{Q}^+$ (which is added to its score) for each length of RT-PCR test result that it explains and gets penalty $P \in \mathbb{Q}^+$ (which is subtracted from the score) for each RT-PCR test result with which it is inconsistent.

Thus we can formulate the problem (in its full generality) as follows:

Problem 1. Given an exon graph $G = (V, E, \ell, S, s, t)$, results of RT-PCR tests $T_1, \ldots, T_R$, bonus B for explaining measured length and penalty P for inconsistency with result of an RT-PCR test, find the transcript (s–t path in G) with the highest score.

Unfortunatelly, this problem is way too hard (actually several its subproblems and special cases turn out to be NP-hard). In the next chapter we study the following subproblems and variants:

- In Section 4.1 we are concerned with just one RT-PCR test result. We develop a pseudopolynomial algorithm and an approximate solution.
- In Section 4.2 we try to generalize the results from Section 4.1. However, we show that even a slightly simpler problem (without the graph structure) with multiple RT-PCR test results and alternative splicing is strongly NP-hard.
- In Section 4.3 we drop all the lengths and scores and study two problems: finding a transcript that does not contain any of the forbidden pairs of vertices (consistency) and a transcript that contains as many useful pairs of vertices as possible (explaining results). These are special cases of Problem 1 and we prove them NP-hard (thus these may be thought of as the hard core of the problem). We further study several classes of forbidden or useful sets of vertex pairs (by their mutual positions) and either prove them NP-hard or give efficient algorithms.
- Finally in Section 4.4 we return to our problem and generalize some results of Section 4.3 into some special cases of Problem 1.

Chapter 4

Algorithms and Hardness Results

4.1 Single RT-PCR Test

4.1.1 Bad News

Before studying the problem in full generality we analyze the problem with just a single RT-PCR test result with only one length. All the algorithms in this section can be easily extended for a single result with more measured lengths (alternative splicing). Also if the test gives more products, we may want to select those lengths or that length that is not yet explained by well-known transcripts.

Problem 2 (Single RT-PCR test). Given an exon graph $G = (V, E, \ell, S, s, t)$ with two distinguished vertices s and t (starting and terminal) and a single RT-PCR test result $T = (s, t, \langle m, M \rangle)$ find an s–t path π (transcript) of length $m \leq \ell(\pi) \leq M$ with the highest score $S(\pi)$.

Note that in this setting we seek to explain the result of the RT-PCR test. Also we can ignore the part of the exon graph outside of the two vertices chosen as primers, since the transcript length in this part is not considered and the score can be simply optimized.

Theorem 4.1. *The problem of finding even a feasible s–t path π such that $m \leq \ell(\pi) \leq M$ is NP-complete.*

Proof. Reduction from subset sum problem: for set $X = \{x_1, \dots, x_n\}$ and target sum S we construct a complete directed acyclic graph where lengths of vertices are the x_i's and $m = M = S$ is the desired sum.

Actually we do not need a complete graph: we can construct graph $G = (V, E)$, where $V = \{s = a_0, v_1, a_1, \dots, v_n, a_n = t\}$ and $E = \{(a_{i-1}, v_i), (v_i, a_i), (a_{i-1}, a_i) \mid 1 \leq i \leq n\}$, i.e. we add auxilliary vertices between v_i's, where $\ell(a_i) = 0$ and $\ell(v_i) = x_i$ (for all i) and from all auxilliary vertices we can either go through vertex v_i or jump over to the next auxilliary vertex. Thus any s–t path naturally corresponds to a subset of X and vice versa. □

4.1.2 Good News

However, the problem is not strongly NP-complete: we show a pseudopolynomial algorithm:

Theorem 4.2. *The problem of finding the best transcript of a given length can be solved in $O(M(V+E))$ where M is the maximum length in the RT-PCR test result.*

Proof. We use dynamic programming approach: Let $\mathcal{H}[i,l]$ be the highest score we can achieve by some s–v_i path of length l in G or $-\infty$ if there is no such path. Then $\mathcal{H}[0,\ell(s)] = 0$ and $\mathcal{H}[0,l] = -\infty$ for all $l \neq \ell(s)$. If we know $\mathcal{H}[j,l']$ for all $j < i$ and all $l' \leq M$, we can calculate $\mathcal{H}[i,l]$ for all $l \leq M$ by formula

$$\mathcal{H}[i,l] = \max\{\mathcal{H}[j, l-\ell(i)] + S(j,i) \mid (v_j, v_i) \in E\}.$$

We can improve this algorithm by considering only achievable lengths l for each vertex i. To do so we store the highest scores in a map $\mathcal{H}_i$ and go the other way around: When we process i-th vertex, for all achievable lengths l and for all out-neighbours[1] v_j of v_i we set

$$\mathcal{H}_j[l+\ell(v_j)] \leftarrow \max(\mathcal{H}_j[l+\ell(v_j)],\ \mathcal{H}_i[l] + S(i,j)).$$

If we implement map $\mathcal{H}_i$ as a hash table for each i, or even use a common hash table $\mathcal{H}$ with key being a pair (vertex, length), we get the same expected time complexity and possibly lower space requirements (see Algorithm 1).

Algorithm 1: Pseudopolynomial algorithm for a single RT-PCR test

```
Input: Exon graph $G = (V, E, \ell, S)$, RT-PCR test result
       $T = (s, t, \langle m, M\rangle)$
Output: The highest score of an $s$–$t$ path $\pi$ of length $m \leq \ell(\pi) \leq M$

mind[$n+1$] ← 0; maxd[$n+1$] ← 0;
for $i \leftarrow n$ to 0 do
    mind[$i$] ← $\ell(i) + \min\{$mind[$j$] $\mid v_j$ is an out-neighbour of $v_i\}$;
    maxd[$i$] ← $\ell(i) + \max\{$maxd[$j$] $\mid v_j$ is an out-neighbour of $v_i\}$;
end
for $i \leftarrow 0$ to $n+1$ do  $\mathcal{H}_i \leftarrow \emptyset$;
$\mathcal{H}_0[\ell(s)] \leftarrow 0$;
for $i \leftarrow 0$ to $n$ do // process vertex $v_i$
    foreach out-neighbour $v_j$ of $v_i$ and $(l, S) \in \mathcal{H}_i$ do
        $l' \leftarrow l + \ell(v_j)$; $S' \leftarrow S + S(v_i, v_j)$;
        if ($m \leq l +$ maxd[$j$]) and ($l +$ mind[$j$] $\leq M$) and
          ($\mathcal{H}_j[l']$ is unset or $\mathcal{H}_j[l'] < S'$) then
          $\mathcal{H}_j[l'] \leftarrow S'$;
        end
    end
end
return $\max\{\mathcal{H}_{n+1}[l] \mid m \leq l \leq M\}$;
```

[1] we call v an out-neighbour of u, if there is an edge $(u, v) \in E$.

We can further improve the algorithm by eliminating lengths that cannot achieve the target length. We can calculate for each vertex v the minimum distance $\mathrm{mind}(v)$ and the maximum distance $\mathrm{maxd}(v)$ from v to t and ignore all lengths l such that $l + \mathrm{mind}(v) > M$ or $l + \mathrm{maxd}(v) < m$ (i.e. even if we went by the shortest path from now on, the s–t path would be too long or even if we went by the longest path, the s–t path would be too short). □

Thus we can solve the simplified problem exactly in pseudopolynomial time. We will now introduce an algorithm that can find a transcript with a very good score of slightly worse length: not in $\langle m, M\rangle$, but, say $\langle m/(1+\varepsilon), M(1+\varepsilon)\rangle$. Moreover we can do this for each $0 < \varepsilon$ in time polynomial in $1/\varepsilon$ and the size of input. This is very similar to a fully polynomial time approximation scheme, although technically it is not even an approximation algorithm: instead of returning a feasible solution with almost the best score it returns an *almost feasible* solution with at least as good score as the best possible.

Theorem 4.3. *Let π^* be the optimal transcript of length $m \leq \ell(\pi^*) \leq M$ with the highest score $S(\pi^*)$. It is possible to find transcript π of length $m/(1+\varepsilon) \leq \ell(\pi) \leq M(1+\varepsilon)$ with score $S(\pi) \geq S(\pi^*)$ for every $0 < \varepsilon \leq 2$ in time polynomial in the size of input and $1/\varepsilon$. That is, we can find a transcript of almost the given length with a score that is not less than the score of the best transcript.*

Proof. We use the dynamic programming solution and adapt the idea of FPTAS for the subset sum problem (Ibarra and Kim, 1975). We use map $\mathcal{H}_i$ of highest scores for each length again, but this time we just approximate the length of the transcript so we do not keep lengths that are very close to each other. If ε is big, we keep only few lengths that are further apart, so the algorithm is fast but not so precise. On the other hand, if ε is small, we keep more lengths and the algorithm is more similar to the pseudopolynomial solution.

More precisely, in each $\mathcal{H}_i$ we keep only lengths $l, l' \in \{0, 1, \ldots, M\}$, $l < l'$ such that $l(1+\delta) \leq l'$, where $\delta = \frac{\varepsilon}{2|V|}$. The algorithm works as follows: For each vertex, map $\mathcal{H}_i$ can be thought of as a list of pairs (length, highest score of any path of this length). We begin with $\mathcal{H}_0 = \{(\ell(s), 0)\}$. When processing vertex v_i we sort $\mathcal{H}_i$ by scores and filter the lengths that are too close to each other. More precisely, we store the lengths that survive in $\mathcal{F}$; we start with $\mathcal{F} = \{-\infty, +\infty\}$ (just two sentinels). Then we take the lengths l one by one (from the highest to the lowest score), find their predecessor l_p and successor l_s in the set $\mathcal{F}$ and keep them only if the predecessor and successor are far enough – if $l_p(1+\delta) \leq l \leq l_s/(1+\delta)$. After we filter the lengths that are too close to each other we update the maps $\mathcal{H}_j$ of the neighbouring vertices as in the dynamic programming solution (see Algorithm 2).

We now have to show that

- the algorithm can be implemented in time polynomial in $|V|$, $|E|$, $\log M$ and $1/\varepsilon$;
- and if there is a path π^* of length $m \leq \ell(\pi^*) \leq M$ with maximal score $S(\pi^*)$, the algorithm returns an s–t path π of length $m/(1+\varepsilon) \leq \ell(\pi) \leq M(1+\varepsilon)$ with score $S(\pi) \geq S(\pi^*)$.

Let us count the maximal number of elements in $\mathcal{H}_i$ after filtering: in the worst case $\mathcal{H}_i$ contains all the lengths

$$0, 1, (1+\delta), (1+\delta)^2, \ldots, (1+\delta)^k \leq M.$$

```
Algorithm 2: 1 RT-PCR test with approximate length
  Input: Exon graph $G = (V, E, \ell, S)$, RT-PCR test result
         $T = (s, t, \langle m, M\rangle)$ and parameter $\varepsilon$
  Output: Score $S(\pi) \geq S(\pi^*)$ of an $s$–$t$ path $\pi$ of length
          $m/(1+\varepsilon) \leq \ell(\pi) \leq M(1+\varepsilon)$

  for $i \leftarrow 1$ to $n+1$ do  $\mathcal{H}_i \leftarrow \emptyset$;
  $\mathcal{H}_0[\ell(s)] \leftarrow 0$;
  $\delta \leftarrow \varepsilon/(2 \cdot |V|)$;
  for $i \leftarrow 0$ to $n+1$ do // process vertex $v_i$
      sort $\mathcal{H}_i$ by scores;
      $\mathcal{F} \leftarrow \{-\infty, +\infty\}$;
      foreach $(l, S) \in \mathcal{H}_i$ do
          $l_s \leftarrow$ successor of $l$ in $\mathcal{F}$;
          $l_p \leftarrow$ predecessor of $l$ in $\mathcal{F}$;
          if $l_p(1+\delta) \leq l \leq l_s/(1+\delta)$ then  $\mathcal{F} \leftarrow \mathcal{F} \cup \{l\}$;
                                          else delete $(l, S)$ from $\mathcal{H}_i$;
      end
      foreach out-neighbour $v_j$ of $v_i$ and $(l, S) \in \mathcal{H}_i$ do
          $l' \leftarrow l + \ell(v_j)$; $S' \leftarrow S + S(v_i, v_j)$;
          if $l + \ell(v_j) \leq M(1+\varepsilon)$ and ($\mathcal{H}_j[l']$ is unset or $\mathcal{H}_j[l'] < S'$) then
              $\mathcal{H}_j[l'] \leftarrow S'$;
          end
      end
  end
  return $\max\{\mathcal{H}_{n+1}[l] \mid m/(1+\varepsilon) \leq l \leq M(1+\varepsilon)\}$;
```

Thus the number of lengths is $k+2$ where k satisfies

$$k \leq \log_{1+\delta} M = \frac{\ln M}{\ln(1+\delta)} \leq 1/\delta \left(\frac{1}{1-\delta/2}\right) \ln M \leq \frac{2}{\delta} \ln M = \frac{4|V|}{\varepsilon} \ln M,$$

(using $\ln(1+x) \geq x - x^2/2$ and $1/(1-x) \leq 2$ for $x \leq 1/2$). Thus the number of lengths in each $\mathcal{H}_i$ is polynomial in the size of input and $1/\varepsilon$.

Before filtering each $\mathcal{H}_i$ contains $O(k \deg_{\text{in}} v_i)$ elements (where $\deg_{\text{in}} v_i$ is the in-degree of v_i) and we can sort them in $O(k \deg v_i \log kV)$ time. We can filter lengths that are too close to each other in $O(k \deg v_i \log V)$ time using for example some kind of balanced binary tree for $\mathcal{F}$. Finally updating the values of all out-neighbours can be done in $O(k \deg_{\text{out}} v_i)$ time. Thus the whole algorithm runs in time $O(k(V+E)\log kV)$, where $k = O(1/\varepsilon \cdot V \ln M)$, i.e. in time polynomial in the size of input and $1/\varepsilon$ as stated.

Now let us prove that the algorithm returns a solution as guaranteed by the statement of the theorem. By induction on the number of edges r we can prove that

> for every path $\pi = w_0 w_1 \dots w_r$, where $w_0 = s$ and $w_r = v_i$ of length $\ell(\pi) \leq M$ there is an s–v_i path π' of length $\ell(\pi)/(1+\delta)^r \leq \ell(\pi') \leq \ell(\pi)(1+\delta)^r$ and score $S(\pi') \geq S(\pi)$ in $\mathcal{H}_i$ (after filtering).

This is certainly true for $r = 0$, when $\pi = \pi'$ contain just the starting vertex s. Now let π be an s–v_i path and $\mathring{\pi}$ be the same path without the last vertex. By the induction hypothesis there is a path $\mathring{\pi}'$ such that $\ell(\mathring{\pi})/(1+\delta)^{r-1} \leq \ell(\mathring{\pi}') \leq \ell(\mathring{\pi})(1+\delta)^{r-1}$ and $S(\mathring{\pi}') \geq S(\mathring{\pi})$. By linking vertex v_i to $\mathring{\pi}'$ we get a path π' of length

$$\frac{\ell(\pi)}{(1+\delta)^{r-1}} \leq \ell(\pi') \leq \ell(\pi)(1+\delta)^{r-1} \tag{1}$$

with $S(\pi') = S(\mathring{\pi}') + S(w_{r-1}, w_r) \geq S(\mathring{\pi}) + S(w_{r-1}, w_r) = S(\pi)$. Either π' is in $\mathcal{H}_i$ and there is nothing to prove or π' has been deleted from $\mathcal{H}_i$. However, then there is an s–v_i path π'' such that

$$\frac{\ell(\pi')}{(1+\delta)} \leq \ell(\pi'') \leq \ell(\pi')(1+\delta) \tag{2}$$

and $S(\pi'') \geq S(\pi') \geq S(\pi)$. The statement is proved by combining (1) and (2).

Notice that $\delta = \frac{\varepsilon}{2n}$ and

$$(1+\delta)^n = \left(1 + \frac{\varepsilon/2}{n}\right)^n \leq \exp(\varepsilon/2) \leq 1 + \varepsilon$$

for $\varepsilon \leq 2$. Thus we have proved that if π^* is the optimal s–t path with length in $\langle m, M\rangle$, there exists path π of length in $\langle m/(1+\varepsilon), M(1+\varepsilon)\rangle$ with score $S(\pi) \geq S(\pi^*)$ in $\mathcal{H}_{n+1}$ which is returned by the algorithm. □

4.2 Multiple RT-PCR Tests and Alternative Splicing

Thus we have answered the question for a single RT-PCR test result. What about multiple RT-PCR tests?

4.2.1 Multiple RT-PCR Tests

If all the tests are (as intervals, from the first to the second primer) disjoint, we can run the dynamic programming or approximation algorithm for each test and find each part of transcript independently. Afterwards we can find the s–v_i path with the highest score consecutively for each $0 \leq i \leq n+1$: we extend the highest score s–v_j path for some neighbour v_j before v_i, or if v_i is an end of some RT-PCR test, we can use the path we calculated in the first phase (if such path exists) and get bonus B. Actually for every vertex inside some RT-PCR test we calculate two values: the highest score provided that we went through the first primer and the highest score if we did not pass the first primer. Then at the vertex corresponding to the second primer, we either take the highest score path of required length and get bonus B (if such path exists) or take the highest score path that does not pass the first primer or the best path that goes through the first primer, although we get penalty P. We always choose an option with the highest possible score. Thus we have a corollary:

Corollary 4.4. *If there are R disjoint RT-PCR test results (i.e., the intervals corresponding to the tests with primers being endpoints are disjoint), the problem can be solved by adapted versions of Algorithms 1 and 2 from Theorems 4.2 and 4.3 with the same time and space complexity.*

Actually the $|V|$ and $|E|$ in complexity of these algorithms is the number of vertices and edges inside of some RT-PCR test.

On the other hand, if the RT-PCR tests overlap, it is not sufficient to calculate the lengths for just a single test – for example if p_1 and p_1' are the first primers of two overlapping tests, we can get the same length l from p_1 to p_2 with different lengths l' from p_1' to p_2. An obvious way to generalize our algorithm is to store pairs (l, l'), or generally k-tuples of lengths (measured from k different first primers) in places where k RT-PCR tests overlap.

Corollary 4.5. *If there are multiple RT-PCR test results, we can find a transcript the highest score in time $O(M^k(V + E))$, where k is the highest number of tests that overlap some exon.*

Naturally, this is not an algorithm we are seeking for.

4.2.2 Length Inference

In the previous section we have proved that the problem is (weakly) NP-complete for one RT-PCR test. Here we prove that the situation with multiple RT-PCR tests and alternative splicing is even worse. If the algorithm is to solve Problem 1, it should also decide whether there is a path that explains *all* the results (this is the case when the bonuses and/or penalties are very high and say, scores $S(e)$ are all zero). In this section we consider an even "simpler" problem: If we forget about the graph structure and imagine the RT-PCR tests as intervals, for each interval we know (possibly several) lengths of a product in this interval. Then determining the lengths in the little subintervals composed of the endpoints of the RT-PCR tests or merely deciding whether this can be done in a consistent way is what we call a length inference problem. If there is an algorithm that can find a transcript that explains all the results, then by cutting this transcript into the given intervals it can surely solve the length inference problem (in a graph formulation it would be a special case where all exons are of unit length and the graph is complete).

Problem 3 (The length inference problem (LI)). Let $T_1, \ldots, T_R$ be RT-PCR tests (tuples of the form $(p_1, p_2, \langle m, M\rangle)$, where p_1 and p_2 are positions of the primers and the measured length is between m and M). The set of all primer positions p_1, p_2 divides interval $(0, N)$ into smaller subintervals – it forms a partition $0 = y_0 < y_1 < y_2 < \cdots < y_n < y_{n+1} = N$; denote these intervals $I_i = (y_i, y_{i+1})$.

Given the RT-PCR tests $T_1, \ldots, T_R$ find all possible lengths of products in the subintervals I_i i.e., find all sequences $l_0, l_1, \ldots, l_n$ such that $0 \leq l_i \leq y_{i+1} - y_i$ and for each test T the sum of the lengths between p_1 and p_2 is between m and M: $m \leq \sum_{I_i \subseteq (p_1, p_2)} l_i \leq M$.

If we consider alternative splicing, the tests are of of the form $(p_1, p_2, \langle m_1, M_1\rangle, \ldots, \langle m_k, M_k\rangle)$ and we seek lengths of products $l_0, l_1, \ldots, l_n$ such that for each test T the sum of lengths between p_1 and p_2 is in $\langle m_j, M_j\rangle$ for some $1 \leq j \leq k$. We call this problem (and its decision version, whether such a sequence of lengths exists) a length inference problem with alternative splicing (LIAS).

Theorem 4.6. *The LIAS problem is NP-complete even if there are at most 2 alternatives in every test.*

Proof. The problem is in NP. We show the completeness by reduction from 3-SAT. Given a formula φ in 3-CNF we show how to construct an input to the LIAS problem such that there are product lengths satisfying the RT-PCR tests if and only if there is a valuation of the variables satisfying φ.

One of the problems we have to address is to avoid some undesirable interference between different RT-PCR tests. We solve this by constructing input from *blocks*: block will be an interval of known fixed size.

Intervals of length 1 will correspond to the literals. Literal x will always be followed by its negation $\overline{x}$ as shown in Fig. 4.1(a). Since the length in intervals denoted by x and $\overline{x}$ must sum to 1, they must be indeed complementary. This way we get a block of length 1 that will not interfere with other RT-PCR tests. Constructing an input from such blocks will enable us for example to copy the literals as shown in Fig. 4.1(c): if we know, that the length of a product inside the lightly-shaded block is n, the intervals denoted by $\overline{x}$ on the left and x on the right must sum to 1 and hence must be complementary.

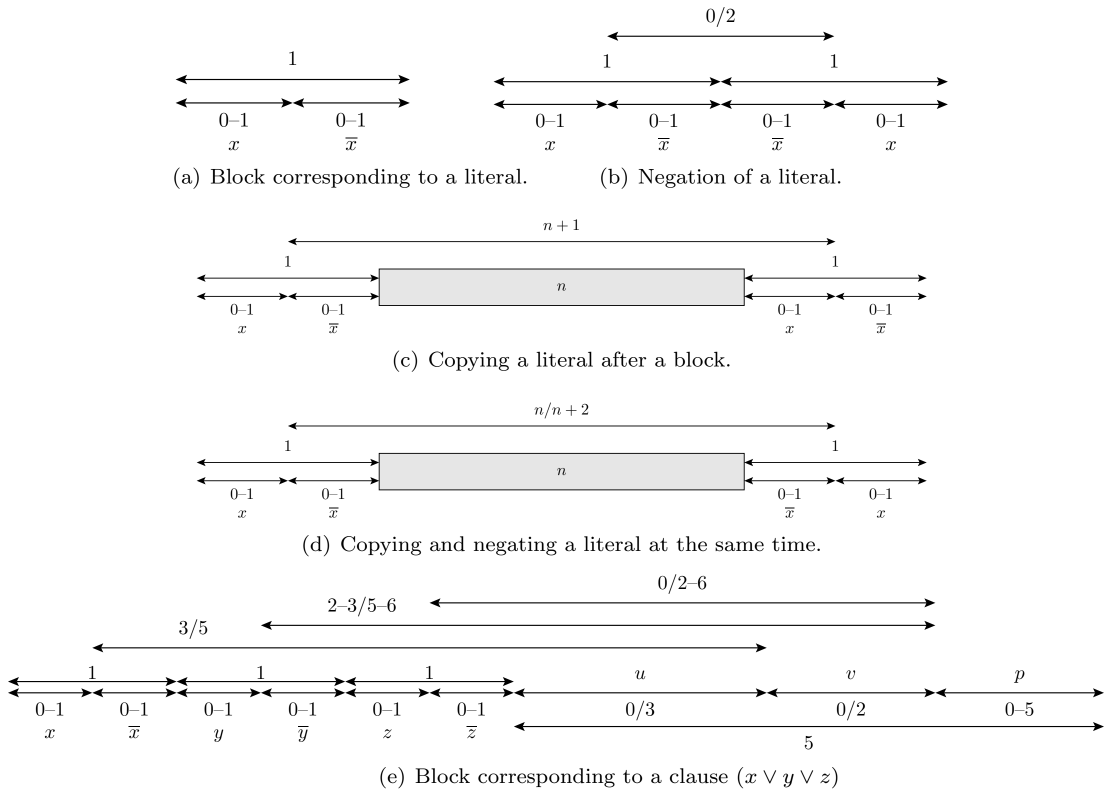
We can also swap the lengths in two intervals as shown in Fig. 4.1(b), which corresponds to negation: since the two intervals denoted by $\overline{x}$ in the middle must sum to 0 or 2 (but not to 1), they must be both 0 or both 1. So if the lengths in the first two intervals were 0 and 1, there will be 1 and 0 in the second two intervals and vice versa. Actually, if we know that there is a block of length n, we can copy a literal x after the block negated as shown in Fig. 4.1(d). The only change we have to make is to consider a test with results $n/n+2$ instead of 0/2 in Fig. 4.1(b).

Finally Fig. 4.1(e) shows a block corresponding to a clause $(x \vee y \vee z)$. We argue that some product lengths satisfy this block if and only if x or y or z is true i.e., product length inside at least one of the intervals denoted as x, y, z is 1. Let us overload the notation and denote the product lengths in the corresponding intervals as x, $\overline{x}$, y, $\overline{y}$, z, $\overline{z}$, u and v. Observe that $\overline{x}+y+\overline{y}+z+\overline{z}+u$ must be 3 or 5. Since $y+\overline{y}+z+\overline{z}$ form a block of length 2, $\overline{x}+u$ must be 1 or 3. If $x=1$, then $\overline{x}=0$ and u must be 3. On the other hand, if $x=0$, $\overline{x}=1$ and u must be 0; so actually $u=3\times x$. Also notice that $\overline{y}+z+\overline{z}+u+v$ must be 2, 3, 5 or 6, so $\overline{y}+u+v$ must be 1, 2, 4 or 5. Now if $y=1$, $\overline{y}=0$ and $u+v$ must be 1, 2, 4, or 5. Length u is 0 or 3, but either way v must be 2. On the other hand, if $y=0$, $\overline{y}=1$ and $u+v$ must be 0, 1, 3, or 4; either way v must be 0. So actually $v=2\times y$. Finally $\overline{z}+u+v$ must be 0 or at least 2. Obviously if x or y is true, $u=3$ or $v=2$, so no matter what is z, this test is satisfied. If both x and y are false, $u+v=0$ and $\overline{z}$ must be 0 too (it cannot be 2 or more), so z must be 1. Thus we have proved that there exist lengths $x, \overline{x}, \ldots, u, v$ that satisfy all the tests if and only if x or y or z is true (has length 1).

Notice that we do not know lengths u and v – therefore we juxtapose an interval p with product length from 0 to 5 which will serve as a padding. We add a constraint $u+v+p=5$ (which is always satisfiable by a suitable choice of p) turning the whole construction into a block of length 8.

Now we show how to build an input from the blocks for the whole formula φ. The input will start with all the variables (followed by their negations). Product lengths in this part will correspond one-to-one with valuations of φ. Then for every clause of φ_i we copy the variables in φ_i normally or negated and apply the block from Fig. 4.1(e). For example if $\varphi_i = (x_2 \vee \overline{x_5} \vee x_{11})$, we copy x_2, then copy x_5 negated (Fig. 4.1(d)) and after that we copy x_{11} normally.

If φ has m variables and n clauses, this construction creates $m+12n$ RT-



(a) Block corresponding to a literal.

(b) Negation of a literal.

(c) Copying a literal after a block.

(d) Copying and negating a literal at the same time.

(e) Block corresponding to a clause $(x \lor y \lor z)$

Figure 4.1: Blocks of intervals from which we construct an input corresponding to a 3-SAT formula.

PCR tests (if we do not count intervals with lengths 0–1 and 0–5, since these lengths are implicit). We conjecture that we can combine the block for clause (Fig. 4.1(e)) with copying or negative copying and improve this to $m+6n$ tests. (As we have shown, this is true for unnegated variables, however, we did not try all 8 possibilities of negated and unnegated variables in a clause.) □

4.3 Useful and Forbidden Pairs

In Section 4.1 we solved the problem for a single RT-PCR test, however we were unable to come up with an efficient algorithm for more RT-PCR tests. In this section we show, that there is no such algorithm (polynomial in M, $|V|$ and $|E|$), unless P = NP.

We study two problems: useful and forbidden pairs that are at the core of our problem. We consider the mutual positions of the RT-PCR primer pairs and prove that for some classes the problem is still hard and for some it is efficiently solvable by dynamic programming.

Both problems are essentially the Problem 1 without lengths and scores (say $\ell(v) = 0$ and $S(e) = 0$ for every $v \in V$, $e \in E$):

Problem 4 (The path avoiding forbidden pairs problem (PAFP)). Given a directed acyclic graph G with two distinguished vertices s, t and a set $F \subseteq V \times V$ of forbidden pairs of vertices, find an s–t path π which does not contain any forbidden pair i.e., π contains at most one vertex from each pair $(u, v) \in F$.

We will also refer to the decision problem, whether such path π exists, as path avoiding forbidden pairs problem (PAFP).

This is a special case of Problem 1: Recall that an RT-PCR test for which there is no transcript which contains both primers (in some of its exons), gives no product (formally, the result of the test is just a pair (p_1, p_2)). In the language of graph theory this means that an s–t path may go through p_1 or through p_2 (or none), but not through both p_1 and p_2 (the transcript has to be consistent with the test result). Thus from now on we call a pair of primers (p_1, p_2) from an RT-PCR test that gave no product a *forbidden pair*. In Problem 4 all the RT-PCR tests give no product.

We can also study the reverse problem where the length measurements of RT-PCR results have such a wide margin that it is not really difficult to explain some test result (or any combination of them). Then we get a similar problem but with different motivation: we search for an s–t path that does not avoid but passes through or “collects” the pairs of primers:

Problem 5 (The path passing useful pairs (PPUP)). Given a directed acyclic graph G with two distinguished vertices s, t and a set $U \subseteq V \times V$ of useful pairs of vertices, find an s–t path π which contains the maximum number of useful pairs.

We will also refer to the decision problem, whether there is a path π containing at least k useful pairs, as path passing useful problem (PPUP).

This is also a special case of Problem 1, where all the RT-PCR test results are of the form $(p_1, p_2, \langle 0, L\rangle)$ where L is sufficiently large, e.g. $L = \sum_v \ell(v)$. This way every transcript that passes both p_1 and p_2 has length in $\langle 0, L\rangle$ and thus explains the length. From now on we call pairs of primers (p_1, p_2) from such an RT-PCR test result *useful pairs*.

The problem of finding a path passing useful pairs turns out to be more interesting for us, since the efficient algorithms for some special cases can be generalized into special cases of Problem 1.

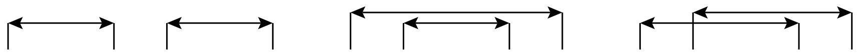
It turns out that the complexity of the problems depends to a large extent on the mutual positions of the pairs and several significant subproblems are solvable in polynomial time. We study both problems for different types of mutual positions of the pairs and thus we analyze the boundary between NP-hardness and efficient solvability.

To define special cases of interest, we extend the topological order into a linear order of vertices and we fix one such order. We say that vertex u is *before* another vertex v (we write $u \prec v$), if it is less in this linear order. We recognize three possible types of mutual position of pairs (p_1, p_2) and (p'_1, p'_2) (let p_1 be before p'_1):

1. *disjoint* – p_2 is before p'_1 i.e., $p_1 \prec p_2 \prec p'_1 \prec p'_2$, see Fig. 4.2(a)

2. *nested* – p_2 is after p'_2 i.e., $p_1 \prec p'_1 \prec p'_2 \prec p_2$, see Fig. 4.2(b)

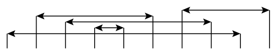
3. *halving* – p_2 is before p'_2, but after p'_1 i.e., $p_1 \prec p'_1 \prec p_2 \prec p'_2$, see Fig. 4.2(c)



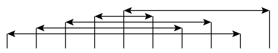
(a) Disjoint pair (b) Nested pair. (c) Halving pair.

Figure 4.2: Different mutual positions of RT-PCR tests (forbidden or useful pairs).

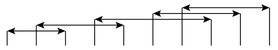
We define 7 classes of RT-PCR test sets, according to the mutual positions of the pairs that occur in the set:



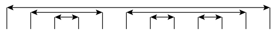
1. *general* – there are no constraints on the positions of pairs; any set of pairs belongs to this class; this corresponds to the Problems 4 and 5



2. *halving structure* – there may be nested pairs and halving pairs, but no two pairs are disjoint; as a consequence all the first primers are before all the second primers i.e., if we number the first primers $1, 2, \ldots, R$ and denote the corresponding second primers $1', 2', \ldots, R'$, then a set of pairs has halving structure, if the primers are ordered $1, 2, 3, \ldots, R, \sigma(1'), \sigma(2'), \sigma(3'), \ldots, \sigma(R')$ for some permutation σ

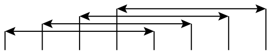


3. *ordered* – there may be disjoint and halving pairs, but no two pairs are nested; as a consequence all the second primers are in the same order as the first primers i.e., we say that a set of pairs is ordered, if and only if the primers are ordered so that $1 \prec 2 \prec \cdots \prec R$ and $1' \prec 2' \prec \cdots \prec R'$

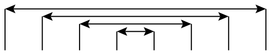


4. *well-parenthesized* – there may be disjoint and nested pairs, but no two pairs are halving; if we order the primers and substitute different parentheses for different pairs, say $(_i$ for the first primer and $)_i$ for the second primer of the i-th pair, we get a sequence that is well-parenthesized

5. *ordered halving* – there may be only halving pairs (any two pairs halve each other); this is an intersection of the class of ordered pairs and pairs with the halving structure; thus this class inherits the atributes of both: all the first primers are before all the second primers and at the same time the first primers are in the same order as the second primers; thus the primers are ordered $1, 2, 3, \ldots, R, 1', 2', 3', \ldots, R'$



6. *nested* – there are only nested pairs i.e., the first and second primers are ordered $1, 2, 3, \ldots, R, R', \ldots, 3', 2', 1'$; this is a special case of well-parenthesized pairs



7. *disjoint* – all the pairs are pairwise disjoint; the primers are ordered $1, 1', 2, 2', 3, 3', \ldots, R, R'$; problems for this class are easily solved and we did so in the previous section

The previous work and our own results are summarized in Tables 4.1 and 4.2. The problem of avoiding forbidden paths was already studied as early as in the 70's in connection with an automatic software testing and validation (Krause et al. (1973), Srimani and Sinha (1982)). In 1976 Gabow et al. proved the general problem with forbidden pairs NP-complete. The PAFP problem was further studied by Kolman and Pankrác (2008). We improve these results by showing a simpler and direct proof of NP-hardness for halving pairs and a simple and efficient algorithm for the problem with well-parenthesized pairs (that does not use advanced data structures). Furthermore we prove that the PAFP problem with ordered set of pairs is still NP-hard (this result is new).

Since nested pairs are a special case of well-parenthesized forbidden pairs, there is an $O(P(V+E)+P^3)$ algorithm for nested pairs. It remains an open problem whether this can be improved. Furthermore, Kolman and Pankrác (2008) prove that the ordered halving case can be solved in polynomial time. It remains an open problem whether this can be done more efficiently.

Furthermore we study the problem of passing the useful pairs. This problem has not been studied before and we solved it for each subclass of useful pairs.

We end this section by a simple observation. We defined the disjoint, nested and halving with a strict inequality, or in other words, the definition does not allow for pairs with common endpoints. However in practice this may be the case nevertheless. Also in the following section we construct graphs and pairs with common endpoints. The following lemma says that the problems with and without pairs with common endpoints are basically equivalent. This lemma will be often used in the following sections to simplify the proofs and algorithms.

Lemma 4.7. *Graph $G = (V, E)$ and a set of either forbidden or useful pairs given in Problems 4 and 5 (or their subproblems) can be modified in such a way that there is at most one pair starting or ending in each vertex. This can be done efficiently and a solution can be trivially retrieved from the solution for the modified input.*

Proof. It suffices to substitute a vertex with k beginning or ending pairs by a path P_k of k vertices. We redirect all the incoming and out-going edges to/from the different vertices on the path. This way passing a vertex (in the former graph) is equivalent to passing the corresponding path (in the modified graph). Furthermore, we assume that the number of pairs is $O(V)$, which is true for

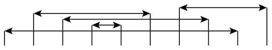
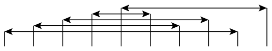
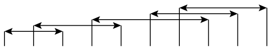
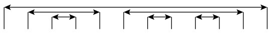
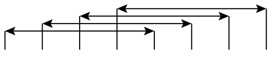
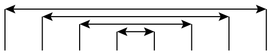
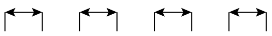
Problem	Complexity	Reference	Example
general problem	NP-hard	Thm. 4.8*	
halving structure	NP-hard	Cor. 4.14†	
ordered	NP-hard	Thm. 4.15	
well-parenthesized	$O(P(V+E)+P^3)$	Thm. 4.19†	
ordered halving	in P	†	
nested	in P	Thm. 4.19	
disjoint	$O(V+E)$		

Table 4.1: Complexity of the path avoiding forbidden pairs problem, where V and E are the number of vertices and edges of the input graph and P is the number of forbidden pairs (we generally expect that $P = O(V)$); NP-hardness of the general problem was proved by Gabow et al. (1976); results marked by † were first proved by Kolman and Pankrác (2008), we present our own proofs.

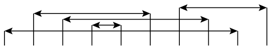
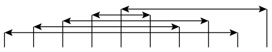
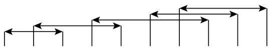
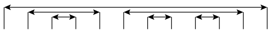
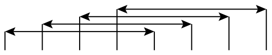
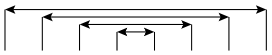
Problem	Complexity	Reference	Example
general problem	NP-hard	Thm. 4.11	
halving structure	NP-hard	Cor. 4.14	
ordered	NP-hard	Thm. 4.15	
well-parenthesized	$O(P(V+E)+P^3)$	Thm. 4.17	
ordered halving	$O(P(V+E)+P^3)$	Thm. 4.20	
nested	$O(P(V+E))$		
disjoint	$O(V+E)$		

Table 4.2: Complexity of the path passing useful pairs problem; V and E are the number of vertices and edges, P is the number of useful pairs (we assume that $P = O(V)$).

e.g. well-parenthesized pairs (there are at most $2|V|-3$ pairs) or ordered pairs[2] (actually in practice we expect it to be $o(V)$ or much less than the number of vertices) so the graph does not grow very much. Also this can be done so that it does not destroy the structure of the input (i.e., being ordered or well-parenthesized). □

4.3.1 Bad news

Here we present the proofs of NP-hardness. We proved that the general problem is NP-hard independently, although it has been known since as early as 1976:

Theorem 4.8. *(Gabow et al., 1976) The PAFP problem is NP-complete.*

Proof. The problem is obviously in NP. We will show the completeness by reduction from 3-SAT.

Let φ be an instance of 3-SAT i.e., $\varphi = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$ is a conjunction of clauses $\varphi_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$, where $\ell_{i,j}$ are literals – variables or their negations ($\ell_{i,j} \in \{x_1, \overline{x_1}, \ldots, x_m, \overline{x_m}\}$). We will construct graph $G = (V, E)$ such that φ is satisfiable if and only if there is an s–t path π in G that avoids all forbidden pairs.

We will have one vertex for each literal $\ell_{i,j}$. Clauses φ_i will form layers of graph G and each literal from φ_i will be connected to every literal from φ_{i+1} (see Fig. 4.3). We add a distinguished starting vertex s connected to literals of the first clause and a terminal vertex t connected to literals of the last clause. Formally: $V = \{s,t\} \cup \{\ell_{i,j} \mid 1 \le i \le n, 1 \le j \le 3\}$ and $E = \{(s, \ell_{1,j}) \mid 1 \le j \le 3\} \cup \{(\ell_{i,j}, \ell_{i+1,k}) \mid 1 \le i < n, 1 \le j,k \le 3\} \cup \{(\ell_{n,j}, t) \mid 1 \le j \le 3\}$.

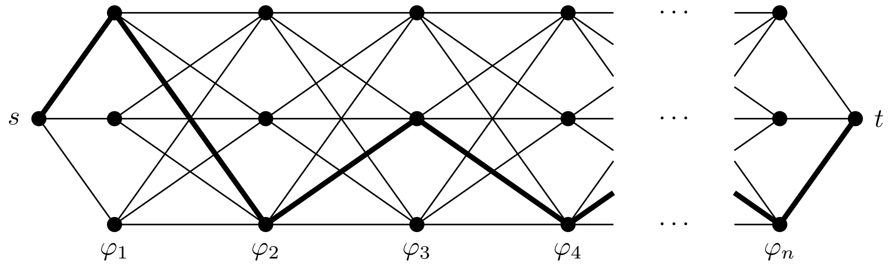


Figure 4.3: Input for the PAFP problem for formula $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$

Every s–t path π in G goes through exactly one vertex from each layer, so it can be interpreted as choosing one literal from each clause that we would like to satisfy. Naturally, not every path will correspond to a correct valuation – we cannot set both a variable and its negation true; but this can be ensured by forbidden pairs. Formally $F = \{(\ell_{i,j}, \ell_{i',j'}) \mid \ell_{i,j} = \neg \ell_{i',j'}\}$

Now if $\pi = s, \ell_{1,j_1}, \ell_{2,j_2}, \ldots, \ell_{n,j_n}, t$ is a path in G that contains at most one vertex from each forbidden pair, then it does not contain a pair $\ell_{i,j_i}, \ell_{i',j_{i'}}$ such that $\ell_{i,j_i} = \neg \ell_{i',j_{i'}}$. Set variable x true, if x is on path π ($x = \ell_{i,j_i}$ for some i) and false, if $\overline{x}$ is on path π ($\neg x = \ell_{i,j_i}$ for some i). Truth value of the rest of variables not on the path π can be chosen arbitrarily. This is a valuation that satisfies φ.

[2] in general or in the halving case there may be as many as $\Omega(V^2)$ pairs, but these versions of the problem are NP-hard

On the other hand if v is a valuation that satisfies φ, the set of true literals does not contain a forbidden pair. Moreover, from each layer in G we can choose at least one vertex that corresponds to a true literal, so these vertices connect s and t in G. Thus we can find an s–t path in G that avoids the forbidden pairs. □

As Kolman and Pankrác (2008) note, the problem remains NP-hard even if every vertex appears in exactly one pair or if graphs (V, E) and (V, F) are planar (we can think of the forbidden pairs as edges).

Corollary 4.9. *(Kolman and Pankrác, 2008) The PAFP problem remains NP-hard even if both (V, E) and (V, F) are planar.*

Proof. We can add a new vertex between each layer of graph G in proof of Theorem 4.8 and substitute the $K_{3,3}$ subgraphs with stars S_7 (we do not need direct edge connecting each pair, it is sufficient that there is a path connecting them).

If we start with an instance of 3-SAT in which for each variable x_i there are at most 3 clauses containing x_i or $\overline{x_i}$ (also NP-hard, Garey and Johnson (1979)), we get planar (V, F). □

Corollary 4.10. *(Kolman and Pankrác, 2008) The PAFP problem remains NP-hard even if every vertex except for s and t appears in exactly one forbidden pair.*

We show a similar proof for the problem with useful pairs:

Theorem 4.11. *The PPUP problem is NP-complete.*

Proof. The proof is similar to the proof of Theorem 4.8. The problem is obviously in NP. We will show the hardness by reduction from 3-SAT.

Let $\varphi = \bigwedge_{1 \le i \le n} \varphi_i$, where $\varphi_i = \bigvee_{1 \le j \le 3} \ell_{i,j}$ and $\ell_{i,j} \in \{x_1, \overline{x_1}, \dots, x_m, \overline{x_m}\}$ be an instance of 3-SAT. We will construct graph $G = (V, E)$ such that φ is satisfiable if and only if there is an s–t path π in G containing (at least) n useful pairs.

G will consist of two parts: The first part will contain a vertex for each variable x_i and its negation $\overline{x_i}$ (see Fig. 4.4). Path π traversing this first part of G will correspond to a valuation v such that x_i is true if and only if π passes vertex x_i. The second part will contain a vertex for each literal $\ell_{i,j}$ as in Fig. 4.4. We add vertices s, t and useful pairs connecting each literal from the first part of G to every occurence of the very same literal in the second part of G.

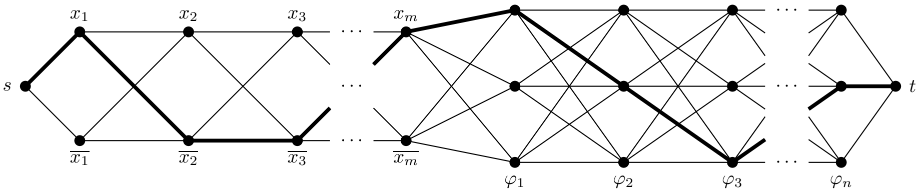


Figure 4.4: Input for the PPUP problem for formula $\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n$

Formally: $V = \{s, t\} \cup \{x_1, \overline{x_1}, \dots, x_m, \overline{x_m}\} \cup \{\ell_{i,j} \mid 1 \le i \le n, 1 \le j \le 3\}$, $E = \{(s, x_1), (s, \overline{x_1})\} \cup \{(x_i, x_{i+1}), (x_i, \overline{x_{i+1}}), (\overline{x_i}, x_{i+1}), (\overline{x_i}, \overline{x_{i+1}}) \mid 1 \le i <$

$m\} \cup \{(x_m, \ell_{1,j}), (\overline{x_m}, \ell_{1,j}) \mid 1 \leq j \leq 3\} \cup \{(\ell_{i,j}, \ell_{i+1,k}) \mid 1 \leq i < n, 1 \leq j,k \leq 3\} \cup \{(\ell_{n,j}, t) \mid 1 \leq j \leq 3\}$ and $U = \{(x_k, \ell_{i,j}) \mid \ell_{i,j} = x_k\} \cup \{(\overline{x_k}, \ell_{i,j}) \mid \ell_{i,j} = \overline{x_k}\}$.

Let n, the number of clauses, be the required number of useful pairs k. Suppose there is an s–t path π in G that passes at least k useful pairs. Then the first part of π determines a valuation v for φ. The clauses φ_i form layers of the second part of G and π passes exactly one vertex from each layer. There are n such layers and π passes a useful pair only if the literal is true in v. Therefore all the clauses must be satisfied by valuation v.

Conversly, if v is a valuation that satisfies φ, π will go through the vertices corresponding to the true literals in the first part of G and from each layer of the second part we can choose at least one vertex that is satisfied by v. Thus we pass a useful pair on every layer of the second part of G and π is an s–t path passing $k = n$ useful pairs. □

Corollary 4.12. *The problem remains NP-hard even if both (V,E) and (V,U) are planar.*

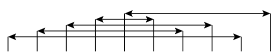
Proof. As in Corollary 4.9, this time we can do a reduction from planar 3-SAT (satisfiability of a formula in CNF with at most 3 literals in each clause such that a bipartite graph made of variables and clauses where variable x_i and clause φ_j are joined by an edge, if x_i or $\overline{x_i}$ belongs to the clause φ_j is planar). Planar 3-SAT was proved NP-hard by Lichtenstein (1982). □

Corollary 4.13. *The PPUP problem is APX-hard.*

Proof. The reduction from proof of Theorem 4.11 is actually a gap-preserving reduction to MAX-3-SAT, which is APX-complete: If k clauses of φ can be satisfied, then there is an s–t path in G that passes k useful pairs and vice versa. □

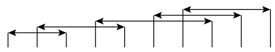
The problem with halving structure of forbidden pairs was proved NP-hard by Kolman and Pankrác (2008) for the first time. We show our own, simpler and direct proof:

Corollary 4.14. *The PAFP and PPUP problems with halving structure are NP-complete.*



Proof. Actually we have already proved this since the pairs in the proof of Theorem 4.11 have the required structure – all the first primers are before all the second primers. An analogous proof can be done for the PAFP problem. □

Theorem 4.15. *The PAFP and PPUP problems with ordered set of pairs are NP-complete.*



Proof. By reduction from 3-SAT: Let $\varphi = \bigwedge_{1 \leq i \leq n} \varphi_i$, where $\varphi_i = \bigvee_{1 \leq j \leq 3} \ell_{i,j}$ and $\ell_{i,j} \in \{x_1, \overline{x_1}, \ldots, x_m, \overline{x_m}\}$ be an instance of 3-SAT.

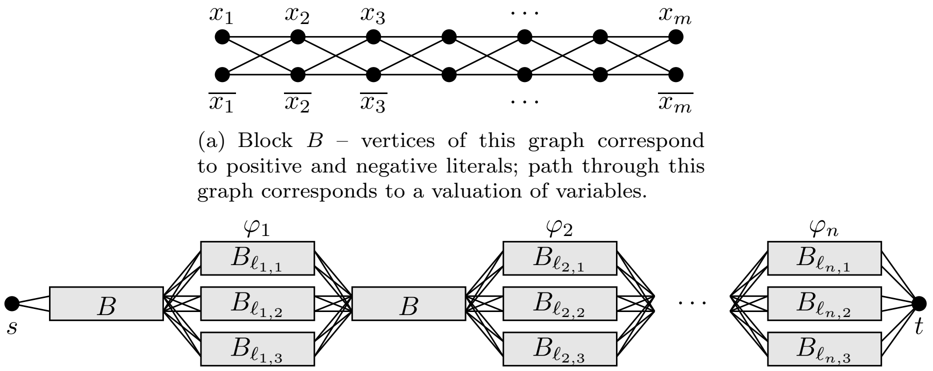
We will construct graph G from blocks: block B is a graph of $2m$ vertices corresponding to the positive and negative literals with edges from both x_i and $\overline{x_i}$ to both x_{i+1} and $\overline{x_{i+1}}$ for $1 \leq i < m$ (see Fig. 4.5(a)). We have already encountered block B in the proof of Theorem 4.11 – it was the first part of the graph. As we know, a path in B naturally corresponds to a valuation of variables.

Let ℓ be a literal and denote by B_ℓ graph $B - \overline{\ell}$ i.e., block B from which we delete vertex $\overline{\ell}$. A path in B_ℓ (since it cannot go through $\overline{\ell}$) naturally corresponds to a valuation where ℓ is true. Blocks B_ℓ will be the basic building blocks of our graph. Because of the restriction on the positions of pairs we will copy whole blocks and ensure that the valuation is always the same in each block by forbidden or useful pairs.

There will be three blocks corresponding to each clause φ_i. For example if $\varphi_i = (x \vee \overline{y} \vee z)$, there will be blocks B_x, $B_{\overline{y}}$ and B_z in graph G. An s–t path in G could go either through B_x, through $B_{\overline{y}}$ or through B_z. Note that the order of the vertices is important. Therefore we represent clause φ_i by subgraph $\mathrm{Zip}(B_x, B_{\overline{y}}, B_z)$, where by zipping 3 graphs with the same number of vertices we mean making a union $B_x \cup B_{\overline{y}} \cup B_z$ and furthermore, if the order of vertices in the three graphs is $x_1 < x_2 < \cdots < x_k$, $y_1 < y_2 < \cdots < y_k$ and $z_1 < z_2 < \cdots < z_k$, then the order of vertices in the zipped graph is $x_1 < y_1 < z_1 < x_2 < y_2 < z_2 < \cdots < x_k < y_k < z_k$.

Again, due to the constraints on position of the pairs we place block B between every two subgraphs corresponding to the consecutive clauses. The resulting graph will look like the one on Fig. 4.5(b). Formally: let us define $\mathit{first}(B) = \{x_1, \overline{x_1}\}$, $\mathit{last}(B) = \{x_m, \overline{x_m}\}$, $\mathit{first}(\mathrm{Zip}(G_1, G_2, G_3)) = \bigcup_i \mathit{first}(G_i)$ and $\mathit{last}(\mathrm{Zip}(G_1, G_2, G_3)) = \bigcup_i \mathit{last}(G_i)$; further define $G_1 \bowtie G_2$ as $G_1 \cup G_2$ with edges joining each vertex from $\mathit{last}(G_1)$ with each vertex from $\mathit{first}(G_2)$. Then we can write the resulting graph G as:

$$\begin{aligned} G \quad = \quad & s \bowtie B \bowtie \mathrm{Zip}(B_{\ell_{1,1}}, B_{\ell_{1,2}}, B_{\ell_{1,3}}) \bowtie B \bowtie \mathrm{Zip}(B_{\ell_{2,1}}, B_{\ell_{2,2}}, B_{\ell_{2,3}}) \bowtie B \bowtie \cdots \\ & \cdots \bowtie B \bowtie \mathrm{Zip}(B_{\ell_{n,1}}, B_{\ell_{n,2}}, B_{\ell_{n,3}}) \bowtie t \end{aligned}$$



(a) Block B – vertices of this graph correspond to positive and negative literals; path through this graph corresponds to a valuation of variables.

(b) Construction of G from the blocks and zipped blocks corresponding to the clauses.

Figure 4.5: Construction of the graph G for a 3-SAT formula φ.

It remains to show how to ensure the same valuation in each block along an s–t path. This is slightly different for the PPUP and PAFP problem: In the PPUP problem the vertices in a block are ordered $x_1 < \overline{x_1} < x_2 < \overline{x_2} < \cdots < x_m < \overline{x_m}$ and a useful pair connects vertices representing the same literals in every two consecutive blocks. An s–t path is required to pass $k = (2n-1)m$ useful pairs. In the PAFP problem there is an alternative ordering of vertices: $\overline{x_1} < x_1 < \overline{x_2} < x_2 < \cdots < \overline{x_m} < x_m$. Forbidden pairs connect variables and their negations, so in order to get an ordered set of pairs we have to alternate between the two orderings. □

4.3.2 Good news

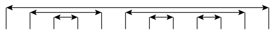
In the previous sections we identified the cases where there is no point of trying to develop an efficient algorithm. However, there are special cases for which the problem *is* efficiently solvable. All the algorithms are based on dynamic programming and can be generalized to a pseudopolynomial algorithm for the special cases of Problem 1.

We start with a simple observation. Our algorithms are based on dynamic programming on vertices of the graph. However, not all the vertices are important. We can contract the paths through vertices where no pair starts or ends into simple edges.

Lemma 4.16 (Graph contraction). *We can modify the input graph in such a way, that exactly one pair (useful or forbidden, depending on the problem) either starts or ends in each vertex – this graph has $O(P)$ vertices. This can be done in $O(P(V+E))$ time.*

Proof. Call the vertices where no pair starts or ends *free* and call *pair* vertices the rest. We modify the input graph as in Lemma 4.7. Now we just traverse the graph (either by DFS or BFS) starting from all the pair vertices, s and t, but going through free vertices only. We form a new graph, where we leave only the pair vertices and discard the free vertices. There will be an edge connecting two pair vertices if there was a path (through free vertices) in the unmodified graph. If we save also these paths, we can reconstruct a path in the unmodified graph from the solution to the new graph (in time proportional to the length of the path). □

Theorem 4.17. *The PPUP problem with well-parenthesized useful pairs can be solved in $O(P(V+E)+P^3)$ time.*



Proof. First we modify the graph so that at most one pair starts or ends in each vertex (Lemma 4.7). We solve the problem by dynamic programming. For every $0 \le i \le j \le n+1$ we calculate the value $\mathcal{P}[i,j]$ defined as the highest number of useful pairs we can take on a path from v_i to v_j. The table $\mathcal{P}$ will be calculated in "diagonal order" by increasing value of $j-i$ i.e., we will calculate the value of $\mathcal{P}$ for vertices that are gradually farther and farther apart.

Let us define an auxilliary value

$$M_{i,j} = \max\{\mathcal{P}[k,j] \mid i < k \le j,\ v_k \text{ is a neighbour of } v_i\}$$

which we can compute in $O(j-i) = O(V)$ time. Then

$$\mathcal{P}[i,j] = \begin{cases} M_{i,j} & \text{if no useful pair begins in } v_i \\ M_{i,j}+1 & \text{if } (v_i,v_j) \text{ is a useful pair} \\ \max(M_{i,j}, \mathcal{P}[i,k]+\mathcal{P}[k,j]) & \text{if } (v_i,v_k) \text{ is a useful pair for } i<k<j. \end{cases}$$

Let π be the v_i–v_j path with the highest number of useful pairs. The first case is obvious: if no useful pair begins in v_i, the second vertex of π must be one of the v_i's neighbours and we take the best path from them to v_j (this is the value $M_{i,j}$). We add one if (v_i, v_j) is itself a useful pair. If there is a useful pair starting in v_i, we may either take it or let it be. If we do not want to take it, the best possibility is $M_{i,j}$ as in the previous cases. However, if we decide to take it, we have to go through vertex v_k. Let the subpath of π from v_i to v_k be π_1 and π_2 be the rest. Thanks to the well-parenthesized structure, no pair can begin on π_1 and end on π_2. Thus the best v_i–v_k–v_j path is composed of the best path from v_i to v_k and the best path from v_k to v_j.

This way the value of $\mathcal{P}[0, n+1]$, which is the highest number of useful pairs on an s–t path, can be computed in $O(V^3)$. Using the contraction from Lemma 4.16 we get an algorithm with time complexity $O(P(V+E)+P^3)$. □

Corollary 4.18. *There is an algorithm for the general PPUP problem with parametric complexity $O(2^k P^3 + P(V+E))$, where k is the minimal number of pairs we would have to take away to get a well-parenthesized set of pairs.*

Proof. We contract the graph according to Lemma 4.16. The value of k and the choice of k pairs that we should take away can be calculated in $O(P^2)$: Let $\mathcal{T}[i,j]$ be the number of parenthesis we should take away to make a region from v_i to v_j "well-parenthesized" (there may be unmatched parentheses but no overlapping pairs). If no useful pair starts at v_i, $\mathcal{T}[i,j] = \mathcal{T}[i+1,j]$; otherwise, if $(v_i, v_k) \in U$ is a useful pair, then we either take it away and $\mathcal{T}[i,j] = \mathcal{T}[i+1,j]+1$, or we leave it there, but then we have to make the inner and outer regions well-parenthesized, so $\mathcal{T}[i,j] = \mathcal{T}[i+1,k-1] + \mathcal{T}[k+1,j]$ ($\mathcal{T}[i,j]$ is the minimum of the two). Finally $k = \mathcal{T}[0,n+1]$.

There are exactly 2^k combinations of whether we take or do not (necessarily) take the "bad" parentheses. We try all of them; we can force an s–t path to go through some vertices by simply deleting edges that jump over them. To compute the exact solution for the remaining well-parenthesized pairs we use algorithm of Theorem 4.17. Thus the whole algorithm runs in time $O(2^k P^3 + P(V+E))$. □

Similar result for the forbidden pairs was first proved by Kolman and Pankrác (2008). Their algorithm uses three rules for reducing the input graph:

1. Contraction of a vertex – if v does not appear in any forbidden pair, we can delete it and add a direct edge (u, w) for every pair of edges (u, v), (v, w).

2. Removal of an edge – if edge $e \in E \cap F$ joins two vertices that make up a forbidden pair, we can remove e from E.

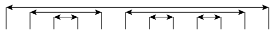
3. Removal of a forbidden pair – if $(u, v) \in F$ is a forbidden pair, but there is no path from u to v, we can remove (u, v) from F.

These three rules are alternatively applied to the graph until we end up with vertices s and t only – either joined by an edge or disconnected – in this case there is no s–t path avoiding forbidden pairs in the original graph.

A simple implementation of this approach gives an $O(V^2E)$ algorithm. Using fast matrix multiplication we can reduce this to $O(V^{3.376})$ or using a dynamic data structure for "finding paths and deleting edges in directed acyclic graphs" by Italiano (1988) we can reduce it still to $O(V^3)$.

Here we describe our own $O(V^3)$ algorithm, its advantages being simplicity, not using advanced data structures or algorithms and extensibility – for example it is easy to compute the least number of forbidden pairs on an s–t path (this is not at all obvious for the former solution). Furthermore the algorithm blends very well with the previous one for the PPUP problem, which we use in Section 4.4.

Theorem 4.19. *The PAFP problem with well-parenthesized useful pairs can be solved in $O(P(V+E)+P^3)$ time.*



Proof. Let us try similar approach as in Theorem 4.17. We modify the input graph so that no two forbidden pairs start or end in the same vertex (Lemma 4.7). We will calculate $\mathcal{A}$ in diagonal order for each $0 \le i \le j \le n+1$ Let $\mathcal{A}[i,j]$ be true if and only if there is a path from v_i to v_j that does not contain forbidden pairs. First if $(v_i, v_j) \in F$ is itself a forbidden pair, then $\mathcal{A}[i,j]$ is false by definition. If no forbidden pair starts in v_i, the second vertex on the v_i–v_j path has to be one of the v_i's neighbours, so $\mathcal{A}[i,j] = \bigvee_k \mathcal{A}[k,j]$ where $i < k \le j$ and v_k is v_i's neighbour. That is, $\mathcal{A}[i,j]$ is true if and only if there is a "safe" path from some neighbour v_k to v_j. If there is a forbidden pair starting in v_i, there still may be no forbidden pair ending in v_j and we may calculate $\mathcal{A}[i,j]$ in the same way symmetrically.

The last case, when there are two forbidden pairs, say (v_i, v_p) and (v_q, v_j) is the hardest. There is a possibility that there is a safe v_i–v_j path through some "middle" vertex v_m for $p < m < q$ (see Fig. 4.7(a)). If this is the case, we can discover it easily: $\mathcal{A}[i,j] = \bigvee_{p<m<q}(\mathcal{A}[i,m] \wedge \mathcal{A}[m,j])$. However, the path from v_i to v_j may also "jump" over this region (see Fig. 4.7(b)). Then there is an edge on the path that starts before v_p and ends after v_q. Surely we can try all such edges, but this way we get time complexity $O(V^4)$. In the rest of the proof we show how to decrease the complexity to $O(V^3)$.

Consider all the forbidden pairs which contain v_j (v_j is between the first (exclusive) and the second primer (inclusive) in the topological order) and do not contain v_i. Let us define $\mathit{last}(i,j)$ as the last beginning vertex from these forbidden pairs. If we wrote the forbidden pairs as a well-parenthesized sequence of left and right parentheses, $\mathit{last}(i,j)$ would be the last unmatched left parenthesis (strictly) between i and j (see Fig. 4.6).

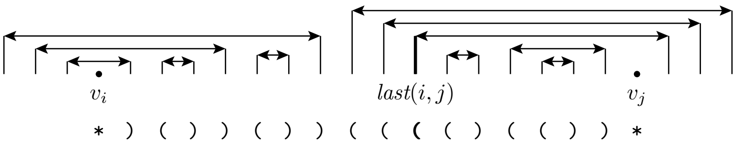
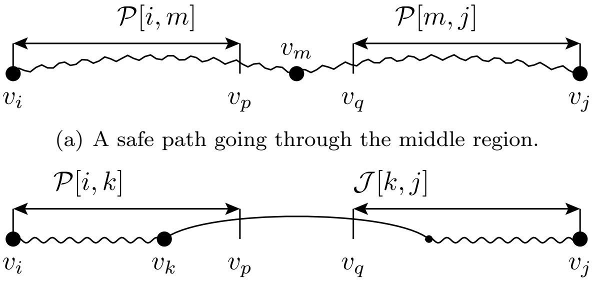


Figure 4.6: Definition of $\mathit{last}(i,j)$ – it is a starting vertex of the most nested pair containing v_j and not v_i, or equivalently, the last unmatched left parenthesis if we depict the pairs as matching parentheses.

Values $\mathit{last}(i,j)$ can be computed easily with help of one stack: we just push a forbidden pair on the stack when it begins and pop it from the stack when it ends. We start from $j = i+1$ with an empty stack with convention that popping from an empty stack does not do anything and top of an empty stack is undefined. This way $\mathit{last}(i,j)$ is always the value on the top of the stack (possibly undefined). Note that when there is a forbidden pair ending in vertex v_j, then $\mathit{last}(i,j)$ is exactly the first vertex of this pair (unless this is before v_i).

In addition to calculating values $\mathcal{A}[i,j]$ we will compute also values $\mathcal{J}[i,j]$ which we define as true if and only if there is a *safe* v_i–v_j path (i.e., without forbidden pairs) such that the first edge jumps over the $\mathit{last}(i,j)$. In other words there is a neighbour v_k of v_i such that $\mathit{last}(i,j) < k \leq j$ and there is a safe path from v_k to v_j – $\mathcal{A}[k,j]$ is true. Values $\mathcal{J}[i,j]$ are computed easily: we just examine the neighbours of v_i.



(a) A safe path going through the middle region.

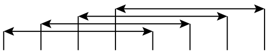
(b) A safe path jumping over the middle region; arc represents one edge, wavy lines represent paths.

Figure 4.7: Two cases that may arise when finding a safe path from v_i to v_j – either the path goes through the middle region, or it jumps over.

Using the table $\mathcal{J}$ we can compute also $\mathcal{A}$ efficiently. The first three cases remain the same; if (v_i, v_j) is a forbidden pair, $\mathcal{A}[i,j]$ is false. If no pair starts in v_i or ends in v_j, we compute $\mathcal{A}[i,j]$ by examining their neighbours. If there are forbidden pairs (v_i, v_p) and (v_q, v_j), then either the path goes through the middle region between v_p and v_q (see Fig. 4.7(a)), or there is vertex v_k before v_p such that there is a safe path from v_i to v_k and from v_k it jumps after v_q and goes safely to v_j (see Fig. 4.7(b)). Formally:

$$\mathcal{A}[i,j] = \begin{cases} \text{false} & \text{if } (v_i, v_j) \in F \text{ is a forbidden pair} \\ \bigvee_k \mathcal{A}[k,j] & i < k \leq j,\ v_k \text{ is a neighbour of } v_i \\ & \text{if no forbidden pair starts in } v_i \\ \bigvee_k \mathcal{A}[i,k] & i \leq k < j,\ v_k \text{ is a neighbour of } v_j \\ & \text{if no forbidden pair ends in } v_j \\ \bigvee_{p<m<q} (\mathcal{A}[i,m] \wedge \mathcal{A}[m,j]) & \text{if } (v_i, v_p) \text{ and } (v_q, v_j) \text{ are forbidden pairs} \\ \vee \bigvee_{i \leq k < p} (\mathcal{A}[i,k] \wedge \mathcal{J}[k,j]) & \end{cases}$$

This way $\mathcal{A}[0, n+1]$, which is true if and only if there is a safe s–t path, can be computed in $O(V^3)$, or in $O(P(V+E)+P^3)$ using the preprocessing step from Lemma 4.16. □



Theorem 4.20. *The PPUP problem with ordered halving useful pairs can be solved in $O(P(V+E)+P^3)$ time.*

Proof. We first transform the graph into the normal form from Lemma 4.16 and calculate a transitive closure, so that there is an edge between two pair vertices if there is a path connecting them in the unmodified graph. Now if there are

n useful pairs, there are pairs starting in vertices $v_1, v_2, \ldots, v_n$ and ending in $v_{n+1}, v_{n+2}, \ldots, v_{2n}$, respectively – the k-th pair is (v_k, v_{n+k}).

When we search the best s–t path, we are actually searching for a sequence of useful pairs such that

1. there is an edge in the modified graph joining the consecutive starting and the consecutive ending vertices and

2. there is an edge joining the last starting and the first ending vertex.

Our algorithm has two phases: in the first phase we find long sequences of useful pairs that satisfy the first condition, or more precisely, for each $1 \leq i \leq j \leq n$ we find the longest sequence of useful pairs satisfying condition 1 starting with the i-th and ending with the j-th pair. In the second phase we then find the longest sequence out of these satisfying condition 2.

Thus let $\mathcal{P}[i,j]$ be the longest sequence of useful pairs from the i-th to the j-th satistying 1, or in other words, the maximum number of useful pairs on a path from v_i to v_j and a path from v_{n+i} to v_{n+j}. The values $\mathcal{P}[i,j]$ can be computed by dynamic programming – we just search for the best second to last pair. If π is the longest sequence of pairs with i-th pair being the first and j-th pair the last and if k-th pair is the second to last pair of S, then S without the last pair is the longest sequence starting with the i-th pair and ending with the k-th pair. Thus $\mathcal{P}[i,j]$ is the maximum of $\mathcal{P}[i,k]+1$ through all k such that k-th pair is a possible second to last pair, or in other words,

$$\mathcal{P}[i,j] = \max_{i \leq k < j} \{\mathcal{P}[i,k] + 1 \mid (v_k, v_j) \in E \wedge (v_{n+k}, v_{n+j}) \in E\}$$

In the second phase we choose the longest of these sequences such there is an edge joining the last starting and the first ending vertex. That is, we take the maximum of $\mathcal{P}[i,j]$ through all i,j such that the starting vertex of the j-th pair is connected with the ending vertex of the i-th pair; thus the result is

$$\max_{i,j} \{\mathcal{P}[i,j] \mid (v_j, v_{n+i}) \in E\}.$$

(Here we assume that there is a path from s to every starting vertex and from every starting vertex to t – otherwise we could discard this test in the beginning.)

From another point of view: If we create a graph where vertices represent useful pairs and there is a blue edge between pairs (v_i, v_{n+i}) and (v_j, v_{n+j}) if and only if there is a path from vertex v_i to v_j and from v_{n+i} to v_{n+j} and there is a red edge between pairs (v_i, v_{n+i}) and (v_j, v_{n+j}) if and only if there is a path from v_{n+i} to v_j. This graph can be constructed in $O(P(V+E))$ and all the blue paths are exactly the sequences of pairs that satisfy condition 1. Blue paths with first and last vertex joined by a red edge also satisfy condition 2. So we are trying to find the longest blue path with first and last vertex joined by a red edge. □

Corollary 4.21. *There is an algorithm for the general PPUP problem with parametric complexity $O(2^k P^3 + P(V+E))$, where k is the minimal number of pairs we would have to take away to get an ordered halving set of pairs.*

Proof. The proof is similar to that of Corollary 4.18: We contract the graph, calculate the value of k and which pairs should be taken away. This can be calculated in $O(P^2)$: For pairs $p = (v_i, v_j)$ and $q = (v_k, v_l)$ define $p \prec q$ if and only if $i < k < j < l$; Given the sequence of useful pairs (sorted by the first vertex) we find the longest ascending subsequence (these are the "good" pairs).

Then we try all the 2^k combinations of taking and not taking the "bad" pairs using algorithm of Theorem 4.20. □

4.4 More on Multiple RT-PCR Tests

In the previous section we have thoroughly studied the problem of finding an s–t path with as many useful pairs as possible. It was shown, that there is no point in considering general, halving or ordered RT-PCR tests. However, we have shown that the PPUP problem is efficiently solvable for the other special cases such as well-parenthesized pairs, ordered halving pairs, nested or disjoint pairs. In this section we generalize these results. The results are summarized in Table 4.3.

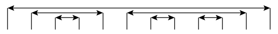
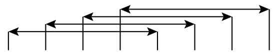
Problem	Complexity	Reference	Example
well-parenthesized	$O(V^3 \Delta M)$	Thm. 4.22	
	$O(T \cdot M(V+E) + T^3 M^2)$	Cor. 4.23	
ordered halving	$O(T \cdot M(V+E) + T^3 \Delta^3 M^2)$	Thm. 4.20	
disjoint	$O(M(V+E))$	Cor. 4.4	

Table 4.3: Complexity of the Problem 1 with no penalty ($P = 0$) for special cases of RT-PCR positions; V and E are the number of vertices and edges, T is the number RT-PCR tests, M is the maximal measured length and Δ is the maximal number of lengths that get any bonus (e.g. $\Delta = M - m + 1$ for one measured length $\langle m, M \rangle$).

It turns out that the algorithms work also for RT-PCR test results with multiple measured lengths and even for a more general bonus function $b_{i,j}$, where $b_{i,j}(l) \geq 0$ for all lengths l. Our constant bonus B with alternative splicing can be modeled as a function b where $b(l) = B$ if length l explains some measured length and $b(l) = 0$ otherwise. However, we may also use fancier functions where the bonus is lower on the boundaries of the interval $\langle m, M \rangle$ and higher in the middle. Let Δ_b be the number of lengths that get any bonus i.e., $\Delta_b = \#\{l \mid b(l) > 0\}$ (this is just $M - m + 1$ for an interval $\langle m, M \rangle$).

Theorem 4.22. *A special case of Problem 1 with well-parenthesized RT-PCR tests and penalty $P = 0$ can be solved in $O(V^3 \Delta M)$, where Δ is the maximum of Δ_b among all RT-PCR test bonus functions and M is the maximum length among all the measured lengths.*

Proof. We generalize our dynamic programming solution from Theorem 4.17 in Section 4.3.2. Let $\mathcal{H}[i, j, d]$ be the maximal score of a v_i–v_j path of length d and let

$$M_{i,j,d} = \max_k \{S(i,k) + \mathcal{H}[k, j, d - \ell(v_i)] \mid (v_i, v_k) \in E, k \leq j\}.$$

Then if there is no first primer in vertex v_i, $\mathcal{H}[i,j,d]$ is simply $M_{i,j,d}$ – we examine all the potential successors of v_i on the path to v_j and find one for which the score is maximal. If v_i and v_j are primers of one RT-PCR test, then the best path has also score $M_{i,j,d}$, however, bonus B is added, if length d explains some length of the test. The last case is when there is an RT-PCR test with primers say v_i and v_k. Then either we do not attempt to explain the lengths of this result, or the v_i–v_j path goes through v_k and there is a d' which explains this result. The highest possible score is then $\mathcal{H}[i,k,d'] + \mathcal{H}[k,j,d-d']$. Or formally:

$$\mathcal{H}[i,j,d] = \begin{cases} M_{i,j,d} & \text{if no first primer begins in } v_i \\ & \text{(or } d \text{ does not explain any length)} \\ M_{i,j,d} + B & \text{if } v_i, v_j \text{ are primers of an RT-PCR test} \\ & \text{and } d \text{ explains the result} \\ \max(M_{i,j,d}, & \text{if } v_i, v_k \text{ are primers of an RT-PCR test} \\ \mathcal{H}[i,k,d'] + \mathcal{H}[k,j,d-d']) & \text{and } d' \text{ explains the result; } i < k < j. \end{cases}$$

□

We can also contract the input graph as we did in the previous section. The following algorithm is more efficient than the first one when $T \ll |V|$ and $M \sim V$ (or more precisely if $M < (|V|/T)^3$).

Corollary 4.23. *A special case of Problem 1 with well-parenthesized RT-PCR tests and no penalty ($P = 0$) can be solved in $O(T \cdot M(V + E) + T^3M^2)$, where T is the number of RT-PCR tests.*

Proof. We run Algorithm 1 from Section 4.1 from the first primer of every RT-PCR test. From the acquired information we can construct the contracted graph, such that vertices are the primers and there is an edge of certain length between two vertices if and only if there is a path of that length between the primers. Its score is the highest score from all paths of the given length. (Note that we get a multidigraph – between every two vertices there are many (but at most $M + 1$) arcs of different lengths.) This can be done in $O(T \cdot M(V + E))$.

The dynamic programming is then run on the modified graph with a minor change that

$$M_{i,j,d} = \max_{k,d'}\{S(e) + \mathcal{H}[k,j,d-d'] \mid e = (v_i, v_k) \in E, \ell(e) = d', k \leq j\}.$$

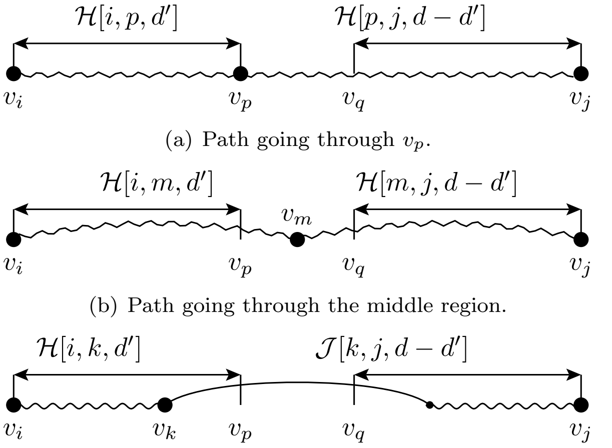
Since now we examine all the neighbours and all possible lengths, the dynamic programming runs in time $O(T^3M^2)$. □

Moreover, the special case with well-parenthesized RT-PCR tests can be solved even if we consider the penalties for inconsistency. The idea is to combine the algorithms for useful and forbidden pairs from Section 4.3.2. (Note that this is not very hard, since both algorithms use dynamic programming method through all pairs of vertices; it would be hardly possible to use the algorithm of Kolman and Pankrác (2008) for the forbidden pairs problem.)

Theorem 4.24. *A special case of Problem 1 with well-parenthesized RT-PCR tests can be solved in $O(T \cdot M(V + E) + T^3M^2)$, where T is the number of RT-PCR tests.*

Proof. We combine the ideas of dynamic programming for the PPUP problem (Theorem 4.17) and PAFP problem (Theorem 4.19) on a contracted graph (Corollary 4.23). Apart from computing the table $\mathcal{H}$ ($\mathcal{H}[i, j, d]$ is the maximal score of a v_i–v_j path of length d), we use table $\mathcal{J}$: $\mathcal{J}[i, j, d]$ is the maximal score of a v_i–v_j path of length d such that its first edge jumps over the $\mathit{last}(i, j)$.

Table $\mathcal{J}$ can be computed from $\mathcal{H}$ by examining the out-neighbours of a given vertex. When computing $\mathcal{H}[i, j, d]$, there are several cases, we describe the "hardest", when there is an RT-PCR test both starting in v_i (say v_p is the second primer) and ending in v_j (say v_q is the first primer). Then either we go through v_p or v_q or neither of them. If we pass vertex v_p, the best way to go from v_i to v_p is $\mathcal{H}[i, p, d']$ for some length d' and the best way to go from v_p to v_j is $\mathcal{H}[p, j, d - d']$ (see Fig. 4.8(a)). Analogically if we pass vertex v_q, the best way is $\mathcal{H}[i, q, d'] + \mathcal{H}[q, j, d - d']$ for some length d'. On the other hand, if we go neither through v_p nor through v_q, then either we go through the middle region (see Fig. 4.8(b)) or we do not – then we have to jump over it (see Fig. 4.8(c)).



(a) Path going through v_p.

(b) Path going through the middle region.

(c) Path jumping over the middle region; arc represents one edge, wavy lines represent paths.

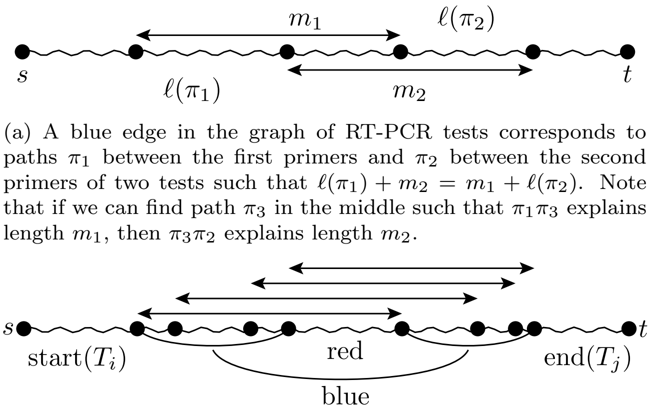
Figure 4.8: Three cases we consider when finding the best path from v_i to v_j of length d – either we explain the RT-PCR test (v_i, v_p), or the path goes through the middle region, or it jumps over.

We calculate $\mathcal{H}$ and $\mathcal{J}$ for all pairs of primers $O(T^2)$ and all lengths $O(M)$ and filling in each $\mathcal{H}[i, j, d]$ can be done in $O(TM)$ time (going through at most all out-neighbours and all lengths d'). Thus a special case of Problem 1 with well-parenthesized RT-PCR tests can be solved in $O(T \cdot M(V + E) + T^3 M^2)$. □

Theorem 4.25. *A special case of Problem 1 with ordered halving RT-PCR tests and no penalty ($P = 0$) can be solved in $O(T \cdot M(V + E) + T^3 \Delta^3 M^2)$, where T is the number of RT-PCR tests.*

Proof. First let us assume that all the RT-PCR tests yield just one product and we measure only one length. Further, let us assume that the measured lengths are precise (we do not have length intervals). We construct a graph of RT-PCR tests as in Theorem 4.20: let $T_1, \ldots, T_r$ be the vertices. Two RT-PCR tests with measured lengths m_1 and m_2 are connected by a blue edge, if there is path

π_1 connecting the first primers and path π_2 connecting the second primers such that $\ell(\pi_1)+m_2 = m_1+\ell(\pi_2)$ (see Fig. 4.9(a)). The rationale behind this is that *if* we find path π_3 such that $\pi_1\pi_3$ (π_1 and π_3 concatenated) explains the first test, then $\pi_3\pi_2$ explains the second one. It goes also the other way around: Certainly any transcript that explains both tests must go through all four primers and the lengths of the corresponding subpaths must satisfy the given equation. Define length $\ell(e)$ of this blue edge as $\ell(\pi_1)$ and score $S(e) = S(\pi_1) + S(\pi_3) + B$. Only the highest score edges will be stored for each length and pair of vertices.



(a) A blue edge in the graph of RT-PCR tests corresponds to paths π_1 between the first primers and π_2 between the second primers of two tests such that $\ell(\pi_1) + m_2 = m_1 + \ell(\pi_2)$. Note that if we can find path π_3 in the middle such that $\pi_1\pi_3$ explains length m_1, then $\pi_3\pi_2$ explains length m_2.

(b) Every s–t path in the input graph can be divided into 5 parts: the first part goes from s to the first primer of the first RT-PCR test on the path (say T_i) and the last part goes from the second primer of the last RT-PCR test (say T_j) to t; the highest scores of these paths are start(T_i) and end(T_j); the 2nd and 4th parts correspond to a path of blue edges; the middle part corresponds to a red edge.

Figure 4.9: Construction of the graph of RT-PCR tests with red and blue edges.

We also add the red edges: for each (highest score) path connecting a starting and an ending primer (possibly from different tests) we have a red edge connecting the two tests. Length and score of the edge is the length and score of the corresponding path. Finally, for each RT-PCR test T_i we compute the highest score of a path from s to the first primer and the highest score of a path from the second primer to t (let us denote these scores as start(T_i) and end(T_i)).

Note that every s–t path can be divided into 5 parts as shown on Fig. 4.9(b). Let T_i and T_j be the first and the last explained RT-PCR test on this path. Then the highest score of the first and the last part is start(T_i) and end(T_i), respectively. The best choice of the middle part corresponds to a red edge from T_i to T_j (of given length). The 2nd and 4th parts correspond to a path of blue edges.

The algorithm will consist of several steps:

1. First we contract the input graph as we did in Theorem 4.23. This can be done in $O(T \cdot M(V + E))$ time by running the dynamic programming algorithm from every primer (and s).

2. We construct the graph of RT-PCR tests with red and blue edges and we calculate the start and end values. The red edges are exactly the edges of

the contracted graph that connect a first primer with a second primer (of possibly different tests). The start and end values are exactly the highest scores of an edge from s to some first primer or from second primer to t, respectively. The blue edges can be found in $O(T^2M)$ time by considering every pair of RT-PCR tests and every length (of π_1).

3. For every pair of RT-PCR tests T_i, T_j and every length $d \leq M$ we calculate the highest score of a blue path from T_i to T_j of length d – this can be done by the familiar method of dynamic programming in time $O(T^3M^2)$:

$$\mathcal{H}[i,j,d] = \max_{i \leq k < j, d'} \{\mathcal{H}[i,k,d'] + S(e) \mid e = (v_k, v_j) \in E, \ell(e) = d - d'\}.$$

   This corresponds to the "first phase" of the algorithm from Theorem 4.20.

4. Recall that the length of a blue path in the graph of RT-PCR tests is the length of the second part of the corresponding s–t path in the input graph (see Fig. 4.9(b)). If we can find the middle part that explains such that we explain the first RT-PCR test (the first vertex of the blue path), we explain all of them. Thus we keep only the blue T_i–T_j paths such that there is a red edge from T_i to T_j such that the length of the blue path plus the length of the red edge explains T_i. Finally we consider also the start and end values and find the maximum. This corresponds to the "second phase" of the algorithm from Theorem 4.20 and can be done in $O(T^2M)$.

Thus we can solve the special case of Problem 1 with ordered halving RT-PCR tests, no penalty, no alternative splicing and precise lengths in time $O(T \cdot M(V + E) + T^3M^2)$. This algorithm can be generalized by having a unique vertex for every pair (RT-PCR test, length explaining this test). Thus if there is a measured length $\langle m, M \rangle$ in a result of an RT-PCR test, we will have $M - m + 1$ vertices for this test (insted of just one vertex). Generally, considering alternative splicing (more measured lengths) we will have overall $T\Delta$ vertices instead of just T. We can solve the special case of Problem 1 with ordered halving RT-PCR tests and no penalty in time $O(T \cdot M(V+E) + T^3\Delta^3M^2)$. □

Chapter 5

Conclusion

We have defined and studied a new problem in bioinformatics. This problem can be formalized as finding the best path in an exon graph with given lengths for certain pairs of vertices. This problem is provably hard: it is weakly NP-hard for a single RT-PCR test and for multiple tests with alternative splicing it is strongly NP-hard even for a simpler version that we called the length inference problem.

Then we study the problem of finding a path without forbidden pairs and a path passing useful pairs. The former problem has already been studied, however we improve on several results:

- proof of NP-hardness for ordered pairs (this has not been studied before)
- simpler and direct proof of NP-hardness for halving pairs
- simple and efficient algorithm for the problem with well-parenthesized pairs (that does not use advanced data structures)

The latter problem has not been studied before and we feel than we have satisfactorily solved it for each subclass of pair positions. We have thus studied the borderline between NP-hardness and efficient solvability.

Finally we returned back to our general problem with scores and lengths. We generalized the algorithms for finding useful pairs for well-parenthesized and ordered halving pairs and showed that in these special cases our problem can be solved in pseudopolynomial time.

There is still work to be done and there are still unanswered problems. In the near future we will carry out experiments that will show how efficient is this approach in discovering the genes in practice. There are open questions including the following:

- How fast can be the nested and ordered halving case of forbidden pairs problem solved? (It is known that these problems are polynomially solvable.)
- Is it possible to approximate the PPUP problem at all? (We have shown that it is APX-hard.)

Bibliography

Agrawal, R. and Stormo, G. D. (2006). Using mrnas lengths to accurately predict the alternatively spliced gene products in *aenorhabditis elegans*. *Bioinformatics*, 22(10):1239–1244.

Brazma, A., Parkinson, H., Schlitt, T., and Shojatalab, M. (2001). *A quick introduction to elements of biology – cells, molecules, genes, functional genomics, microarrays*. `http://www.ebi.ac.uk/microarray/biology_intro.html`. Draft, [Online; accessed 8 January 2008].

Cooper, G. M. and Hausman, R. E. (2004). *The cell : a molecular approach*. ASM Press, Washington, D.C., 3rd edition.

Gabow, H. N., Maheswari, S. N., and Osterweil, L. J. (1976). On two problems in the generation of program test paths. *IEEE Trans. Software Eng.*, 2(3):227–231.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gerstein, M. B., Bruce, C., Rozowsky, J. S., Zheng, D., Du, J., Korbel, J. O., Emanuelsson, O., Zhang, Z. D., Weissman, S., and Snyder, M. (2007). What is a gene, post-encode? history and updated definition. *Genome Res*, 17(6):669–681.

Guigo, R., Flicek, P., Abril, J. F., Reymond, A., Lagarde, J., Denoeud, F., Antonarakis, S., Ashburner, M., Bajic, V. B., Birney, E., Castelo, R., Eyras, E., Ucla, C., Gingeras, T. R., Harrow, J., Hubbard, T., Lewis, S. E., and Reese, M. G. (2006). EGASP: the human ENCODE Genome Annotation Assessment Project. *Genome Biol*, 7 Suppl 1:1–31.

Ibarra, O. H. and Kim, C. E. (1975). Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468.

Italiano, G. F. (1988). Finding paths and deleting edges in directed acyclic graphs. *Inf. Process. Lett.*, 28(1):5–11.

Kolman, P. and Pankrác, O. (2008). On the complexity of paths avoiding forbidden pairs.

Krause, K. W., Smith, R. W., and Goodwin, M. A. (1973). Optional software test planning through automated network analysis. *Proceedings 1973 IEEE Symposium on Computer Software Reliability*, pages 18–22.

Lichtenstein, D. (1982). Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343.

Sakharkar, M. K., Chow, V. T. K., and Kangueane, P. (2004). Distributions of exons and introns in the human genome. *In Silico Biology*, 4.

Srimani, P. K. and Sinha, B. P. (1982). Impossible pair constrained test path generation in a program. *Inf. Sci.*, 28(2):87–103.

Strachan, T. and Read, A. P. (1999). *Human Molecular Genetics*. Bios Scientific Publishers Ltd, Oxford, UK, 2nd edition.

Abstrakt

Formulujeme nový problém z oblasti bioinformatiky; ten sa dá formulovať ako hľadanie s–t cesty s čo najvyšším skóre v grafe. Pre niektoré dvojice vrcholov máme dané dĺžky. Ak cesta prechádza dvojicou vrcholov a jej časť medzi týmito vrcholmi má danú dĺžku, dostaneme bonus (inak malus). Dokazujeme, že niektoré varianty problému sú NP-ťažké a pre niektoré triedy nachádzame efektívne algoritmy.

Už pre jednu dĺžku je problém (slabo) NP-ťažký, avšak dá sa riešiť v pseudopolynomiálnom čase; tiež navrhujeme aproximatívne riešenie. Špeciálne sa venujeme problému hľadania s–t cesty v grafe, ktorá neobsahuje žiadnu "zakázanú" dvojicu; resp. takú, ktorá obsahuje čo najviac "dobrých" dvojíc. Toto sú špeciálne prípady nášho všeobecného problému a tvoria akési jeho "ťažké jadro". Zložitosť problému do značnej miery závisí od vzájomných pozícií týchto dvojíc; tieto môžeme veľmi prirodzene rozdeliť do 7 tried, pre ktoré sa snažíme problém riešiť. Dokazujeme, že pre niektoré triedy je problém stále (silne) NP-ťažký a pre niektoré navrhujeme efektívne algoritmy (takto hľadáme "hranicu" medzi NP-ťažkosťou a efektívnou riešiteľnosťou). Niektoré z týchto efektívnych algoritmov sa potom dajú rozšíriť na špeciálny prípad pôvodného problému.