Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics
Department of Applied Informatics

Alignments on Sequences with Internal Structure

Dissertation

Martina Višňovská
author

Bronislava Brejová
supervisor

9.2.1 Computer Science

Bratislava, 2013

32865922

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Mgr. Martina Višňovská
Study programme:	Computer Science (Single degree study, Ph.D. III. deg., full time form)
Field of Study:	9.2.1. Computer Science, Informatics
Type of Thesis:	Dissertation thesis
Language of Thesis:	English

Title: Alignments on Sequences with Internal Structure

Aim: The goal of the thesis is to explore two computational problems arising in processing databases of DNA or protein sequences containing many similar sequences. The first problem is to provide more accurate estimates of statistical significance of similarities found in such databases. The second problem is to segment input sequences to shorter atomic regions classify these atomic regions by sequence similarity.

Tutor:	Mgr. Bronislava Brejová, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. PhDr. Ján Rybár, PhD.
Assigned:	27.10.2010
Approved:	27.10.2010

prof. RNDr. Branislav Rovan, PhD.
Guarantor of Study Programme

....
Student

....
Tutor

32865922

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Mgr. Martina Višňovská
Študijný program:	informatika (Jednoodborové štúdium, doktorandské III. st., denná forma)
Študijný odbor:	9.2.1. informatika
Typ záverečnej práce:	dizertačná
Jazyk záverečnej práce:	anglický

Názov: Zarovnania sekvencií s internou štruktúrou

Cieľ: Cieľom práce je riešiť dva výpočtové problémy súvisiace so spracovaním databáz DNA alebo proteínových sekvencií, ktoré obsahujú veľké množstvo podobných sekvencií. Prvý problém je poskytnúť presnejšie odhady štatistickej významnosti podobností nájdených v takýchto databázach. Druhý problém je segmentovať vstupné sekvencie na kratšie atomické úseky a klasifikovať tieto úseky podľa sekvenčnej podobnosti.

Školiteľ:	Mgr. Bronislava Brejová, PhD.
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	doc. PhDr. Ján Rybár, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 27.10.2010

Dátum schválenia: 27.10.2010 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

....

študent školiteľ

Acknowledgement

I am very grateful to my supervisor Broňa Brejová for her guidance through research and writing. Tomáš Vinař's encouragement and the willingness to look at the bright side of the things is greatly appreciated. Thank you both for being here. It was a pleasure to work with all members of the bioinformatics group and the time spent here was very enjoyable.

Abstract

Search for sequence similarity in genomic databases is one of the essential problems in bioinformatics. Genomic sequences evolve by local changes affecting one or several adjacent symbols, as well as by large-scale rearrangements and duplications. In this thesis we address two different problems, one connected to the local changes and the other to the large-scale events. Both problems deal with genomic databases with a rich internal structure consisting of repeating sequences.

First, considering only the local changes, we discuss the problem of distinguishing random matches from biologically relevant similarities. Customary approach to this task is to compute statistical P-value of each found match between a query and the searched database. Biological databases often contain redundant identical or very similar sequences. This fact is not taken into account in P-value estimation, resulting in pessimistic estimates. We propose to use a lower effective database size instead of its real size and to estimate the effective size of a database by its compressed size. We evaluate our approach on real and simulated databases.

Next, we concentrate on large-scale duplications and rearrangements, which lead to mosaic sequences with various degree of similarity between regions within a single genome or in genomes of related organisms. Our goal is to segment DNA to regions and to assign such regions to classes so that regions within a single class are similar and there is low or no similarity between regions of different classes. We provide a formal definition of the segmentation problem, prove its NP-hardness, and give two practical heuristic algorithms. We evaluated the algorithms on real and simulated data. Segments found by our algorithm can be used as markers in a wide range of evolutionary studies.

Keywords: homology search, statistical significance, genomic data compression, sequence segmentation, NP-completeness, dynamic programming, heuristic algorithms

Abstrakt

Hľadanie podobností medzi sekvenciami v genomických databázach je jedným z ústredných problémov bioinformatiky. Genomické sekvencie evolvujú malými zmenami ovplyvňujúcimi iba jednu prípadne niekoľko za sebou idúcich báz, rovnako ako aj preusporiadaniami a duplikáciami veľkých blokov v rámci jednej sekvencie. V tejto práci sa zaoberáme dvoma problémami, pričom prvý súvisí s lokálnymi zmenami v sekvenciách a druhý s blokovými preusporiadaniami a duplikáciami.

V prvej časti sa venujeme problému rozoznávania biologicky významných sekvenčných podobností od náhodných zhôd. Zvyčajným postupom pri riešení tohto problému je výpočet štatistickej P-hodnoty pre každú nájdenú podobnosť medzi dotazovanou sekvenciou a prehľadávanou databázou. Biologické databázy často obsahujú niekoľko kópií identických alebo veľmi podobných sekvencií. Tento fakto sa neberie do úvahy pri odhadovaní P-hodnoty, čo spôsobuje pesimistické odhady. Navrhujeme v odhade použiť namiesto reálnej veľkosti databázy menšiu efektívnu veľkosť, ktorú určíme ako veľkosť databázy po kompresii. Náš prístup testujeme na reálnych ako aj simulovaných databázach.

Pri riešení druhého problému sa sústredíme na blokové preusporiadania a duplikácie. Tieto evolučné udalosti vytvárajú mozaikové sekvencie s rôznym stupňom podobnosti medzi regiónmi v rámci jedného genómu ako aj medzi regiónmi v genómoch evolučne blízkych organizmov. Našim cieľom je segmentovať DNA na regióny a rozdeliť tieto regióny do tried tak, že regióny patriace do jednej triedy sa navzájom podobajú, ale podobnosť medzi regiónmi patriacimi do rôznych tried je výrazne nižšia. V tejto práci formálne zadefinujeme problém segmentácie, dokážeme, že je NP-ťažký a popíšeme dva praktické heuristické algoritmy. Úspešnosť našich algoritmov vyhodnotíme na reálnych aj simulovaných dátach. Segmenty nájdené takýmto spôsobom môžu slúžiť ako vstupné dáta pre štúdium evolúcie.

Kľúčové slová: hľadanie homológov, štatistická významnosť, kompresia genomických dát, segmentácia sekvencií, NP-úplnosť, dynamické programovanie, heuristické algoritmy

Contents

Introduction

Search for similarities between biological sequences is one of the essential problems in bioinformatics. Importance of the search grows with increasing amount of available genomic sequences as sequence similarity possibly indicates functional similarity or homology (shored evolutionary origin) of the similar sequences. Results of such similarity search combined with experimental bio-molecular methods allow us to study possible sequence functions in a more efficient way than what can be done using purely bio-molecular approaches. In addition, with identified homologous sequences evolutionary processes at the region can be studied in more detail.

This thesis deals with two problems connected to similarity search in genomic sequences. We focus on sequences which do not follow traditional sequence models assuming random distribution of bases within a sequence. Instead, we expect that the sequence at hand is internally structured, containing several almost identical regions. We study possibility of using this information to improve methods for solving the problems of our interest. First, we address the problem of estimating statistical significance of similarity search for a database consisting of many similar sequences. Second, we are given a genomic sequence, and our task is to uncover it's internal structure. In particular, we identify non-overlapping segments of the sequence and to partition these segments into classes so that segments within each class are similar to each other, and there are no significant similarities between segments from different classes.

For the purpose of this thesis, we represent a DNA sequence as a finite string over the four letter alphabet $\{\mathrm{A}, \mathrm{T}, \mathrm{C}, \mathrm{G}\}$ and call the letters bases or nucleotides. In bioinformatics, sequence similarity search is typically formalized as search for high-scoring sequence alignments. A sequence alignment is an arrangement of the sequences obtained by adding spaces, which we call gaps, inbetween bases of any of the aligned sequences, so that we obtain a rectangular block consisting of bases and gaps, where each row represents one sequence. Corresponding bases of the input sequences are then aligned in the same column. A scoring function is usually defined, which assigns a score to each column and reflects similarity among the column bases. The alignment score, which is the sum of the scores of all alignment columns, reflects the length of the aligned sequences and their similarity. Among all possible alignments of the considered sequences, the highest scoring alignments are typically of interest, and can be computed for two sequences by dynamic programming as we review in Section 1.3. We often wish to find high-scoring alignments between continuous subsequence of a newly sequenced query DNA sequence and subsequences in a database containing already known sequences.

However, many alignments found in this way are just random matches without biological meaning. To distinguish such random matches from the biologically relevant similarities, statistical significance (P-value) of the alignment is estimated. The matches with small enough P-value are then considered to be the relevant similarities. In this context, P-value is the probability that an alignment with a given score or higher would occur by chance in a comparison of a random query and a random database. This P-value is a function of the database size, since a high-scoring similarity is more likely to exist by chance in a larger database. Traditional techniques used to evaluate the significance of search results (Karlin and Altschul, 1990) often build upon theoretical models of databases which differ in important way from the database at hand.

We concentrate on providing better estimates for databases containing many almost identical sequences. Such databases are commonplace due to redundancy of sequence data production and evolutionary relationship of the sequenced regions. To compensate for differences between the real database and the model, we propose to calculate effective database size (smaller than the real size) and use this effective size instead the real size as a parameter in the traditional formulas for P-value estimation. We will study a possibility of using a compressed size of a database to estimate its effective size in context of DNA databases. An appropriate compression algorithm will effectively store only one copy of each repeated string, resulting in a file whose size roughly corresponds to the amount of unique sequences in the database. We test this approach on both simulated and real data.

The second problem which we will address in this thesis is sequence segmentation. In comparative genomics, identification of homologous segments within the sequences of interest is the first step for studying rearrangement and duplication histories (i.e. Kováč et al., 2011; Holloway et al., 2012). The segments are typically constructed from annotated genes in the sequences. However, the histories reconstructed from segments determined according their already known functionality ignore relations among segments of non-coding sequences. These reconstructions are also sensitive to errors in annotation, and may even introduce artifacts due to incomplete gene copies. Using segmentation, which is not based solely on protein coding genes, it may be possible to study recent evolutionary events in more detail.

We provide formal definition of the segmentation problem, prove its NP-hardness, and give two heuristic algorithm for the problem. We evaluated the algorithms on real and simulated data. Segments found by our algorithm can be used as markers in a wide range of evolutionary studies.

The thesis structure is following. In Chapter 1 we introduce the topic of sequence homology search, and review traditional approaches to sequence similarity search and estimation of statistical significance. Chapter 2 deals with the approach to P-value estimation based on database effective size (Višňovská et al., 2010). Chapter 3 is devoted to the segmentation problem (Višňovská et al., 2013).

Chapter 1

Homology Search

In this chapter, we provide basic biological background related to the problems, which we discuss in the following two chapters. We also discuss traditional bioinformatic algorithms for homology search and customary methods to estimate statistical significance of a particular search result.

1.1 Sequences of DNA and Their Evolution

Deoxyribonucleic acid (DNA) consists of two long DNA strands which run in opposite directions (we say they are anti-parallel) and form a shape of double helix as can be seen in Figure 1.1. We denote $5'$ terminus the beginning of each DNA sequence and $3'$ terminus its end. A DNA sequence consists of adenine (A), cytosine (C), guanine (G) and thymine (T). The two DNA strands are bonded so that A and T are always paired together and C and G are always paired together. The main function of DNA is to store genetic information. Segments of DNA which carry information necessary for building RNA and proteins are called genes. For our purpose, we will represent DNA sequences as strings over the four letter alphabet.

Since 1968 when first DNA was sequenced, scientists are improving techniques of DNA sequencing, and the amount of sequenced DNA grows enormously. History and review of sequencing techniques can be found in (Hutchison, 2007). Results of DNA sequencing projects are typically made available to public through various sequence databases. One of the frequently used nucleotide databases, GenBank (Benson et al., 2008), contains nowadays more than 160 million sequences with total

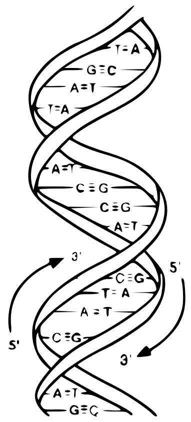


Figure 1.1: A pair of complementary DNA strands from (Holmes and Jobling, 1996).

length almost 150 billion nucleotide base pairs[1], and its growth is shown in Table 1.1.

Evolutionary changes of DNA sequences. Over time DNA sequence can undergo some changes, which we call mutations. We consider the following types of mutations:

- *Single-base substitution.* A base at a particular position of the DNA is exchanged for a different nucleotide.
- *Deletion.* A continuous portion of the DNA is deleted.
- *Insertion.* Adding a random sequence at a particular position of the DNA.
- *Rearrangement.* During this event, a continuous portion of DNA moves from its position to some other position.
- *Inversion.* It is similar to rearrangement, except that the cutout piece of the DNA is inserted at the same position as it was previously, but in a reverse direction relative to the rest of the sequence.
- *Duplication.* In this event a copy of a continuous portion of DNA is inserted at another position.

[1]GenBank Release 194, February 2013:
http://www.ncbi.nlm.nih.gov/genbank/statistics

Date		Base Pairs	Entries
Dec	1982	680 338	606
Nov	1984	3 689 752	4 393
Nov	1986	9 615 371	9 978
Dec	1988	24 690 876	21 248
Dec	1990	51 306 092	41 057
Dec	1992	120 242 234	97 084
Dec	1994	230 485 928	237 775
Dec	1996	730 552 938	1 114 581
Oct	1998	2 008 761 784	2 837 897
Dec	2000	11 101 066 288	10 106 023
Dec	2002	28 507 990 166	22 318 883
Dec	2004	44 575 745 176	40 604 319
Dec	2006	69 019 290 705	64 893 747
Dec	2008	99 116 431 942	98 868 465
Dec	2010	122 082 812 719	129 902 276
Dec	2012	148 390 863 904	161 140 325

Table 1.1: Size of GenBank sequence database since 1982 to 2012.

- *Speciation.* This event occurs when one species splits into two distinct species. Usually, some environmental conditions cause that the population of one species splits into two groups. Initially, their DNA is the same, but over time, each of the subpopulations undergoes independent mutations, which in the end cause a differentiation of the two species.

Some regions within a genome tend to mutate faster than the others. Mutations in functional regions of DNA sequences often lead to a nonviable form of life or a form that will be destroyed through natural selection. Therefore functional regions tend to be more similar to their counterparts in other species than nearby non-functional regions. For example, it was shown that the alignment of whole mouse and human genomes covers 40 % of human genome and an identity within the alignment is 70 %. In comparison, 98 % of human exons[2] are aligned to mouse and identity within alignments is 85 % (Waterston et al., 2002). Therefore we expect that functional regions stay similar, and we can detect them on the basis of their similarity.

[2]Exon is a piece of a DNA sequence that translates to a protein or its part.

Consider genomic sequences S and T belonging to two different contemporary species and the genomic sequence U of their most recent common ancestor. Consider also a particular region u in U and the corresponding sequences s and t in S and T, respectively, such that the regions s and t evolved from u. We then say that s, t, and u are homologous to each other. More precisely, two regions are homologous if they have a common ancestor. Homologous sequences often have have similar functions but there are examples of homologous sequences with different functions as well as functionally similar sequences which are not homologous.

In fact, we do not know how an evolution really progressed because we can analyze only sequences of current species and we are usually interested in homology among DNA sequences which are very distanced in time. We would be able to confirm that two sequences S and T are homologous only if we could explore their most recent common ancestor U and all sequences on the lineages between S and U and between T and U. Since we do not know sequences of all extinct forms, this confirmation is not possible, and we have to decide homology of S and T on the basis of sequence similarity as it is an observable event and homology is not.

We have shown that sequence similarity is a tool for detecting sequence homology and function similarity and now it is a legitimate question why this is useful. Wet lab experiments are expensive in time and money and most of them are tuned to answer a very specific question about the analyzed sequences. Even though it is necessary to evaluate the results of computational methods by wet lab experiments, sequence similarity detection can help to focus such experiments, because we can estimate a function of an unknown sequence region on the basis of its similarity with sequences which have known function. In a consequence, we can propose an experiment which accepts or rejects our hypothesis. We can also use sequence similarity among sequences to infer homology and use putative homology to create phylogenetic trees. It is important to be careful in an interpretation of results we get from such a comparison and strictly distinguish homology, functional similarity and sequence similarity. Although the three concepts partially overlap, they also differ significantly.

				$\mathrm{s}(S,T)$
sequence S	A	T	C	
sequence T	A	T	A	
$w(s_i,t_i)$	$+1$	$+1$	-1	**1**

Figure 1.2: Similarity score of sequences S ="ATC" and T ="ATA". Under each pair of bases is the pair score and the number in last column is sum of pair scores.

1.2 Sequence Alignments

In this section, we formalize biological intuition to the point that allows us to attack sequence similarity search as a computational problem. In particular, we introduce a concept of pairwise alignment, an arrangement of two genomic sequences that leads to the highest similarity score for the two sequences.

For simplicity, we will first consider a simplified model of evolution which allows only single-base substitution. Later we will add more mutation types, in particular insertions and deletions. If only substitutions are allowed, all sequences have the same size. To measure similarity between two DNA sequences, we define sequence similarity score. Let $S = s_1s_2\dots s_n$ and $T = t_1t_2\dots t_n$ are sequences of length n. We assign *base pair similarity score* $w(s_i,t_i)$ to every pair of corresponding bases s_i and t_i according to a scoring function and set *similarity score* $\mathrm{s}(S,T)$ to a sum of its base pair scores:

$$\mathrm{s}(S,T) = \sum_{i=1}^{n} w(s_i,t_i).$$

In the case of DNA sequences, scoring schemes are usually simple, for example Altschul et al. (1990) use score $+5$ for match and score -4 for mismatch. More complex schemes are needed for proteins which consist of amino acids of 20 different types, which show different level of similarities to each other. We will use for illustration in our examples a simple scoring function that assigns score $+1$ for match and score -1 for mismatch. Consider two DNA sequences S ="ATC" and T ="ATA". Base pair scores are $+1$, $+1$, -1, respectively and similarity score $\mathrm{s}(S,T)$ equals to 1 as shown in Figure 1.2.

Alignment with gaps. A situation is more complicated when the sequences differ in length. This can happen when we allow other mutations: insertions and deletions.

									s(S,T)
seq. S	G	C	T	A	T	G	T	G	
seq. T	C	T	A	G	–	–	–	–	
$w(s_i, t_i)$	−1	−1	−1	−1	−1	−1	−1	−1	**−8**

									s(S,T)
seq. S	G	C	T	A	T	G	T	G	
seq. T	–	C	T	A	–	G	–	–	
$w(s_i, t_i)$	−1	+1	+1	+1	−1	+1	−1	−1	**0**

									s(S,T)
seq. S	G	C	T	A	T	G	T	G	
seq. T	–	C	T	–	–	–	A	G	
$w(s_i, t_i)$	−1	+1	+1	−1	−1	−1	−1	+1	**-2**

									s(S,T)
seq. S	G	C	T	A	T	G	T	G	
seq. T	–	C	T	A	G	–	–	–	
$w(s_i, t_i)$	−1	+1	+1	+1	−1	−1	−1	−1	**-2**

Figure 1.3: Four alignments of sequences S ="GCTATGAGC" and T ="CTAG" and their similarity scores. Gaps in sequence T are shown as "−". We assume that insertions in sequence S or deletions in sequence T occurred. The score for a gap is -1.

When we compare two sequences S and T, we capture an insertion which occurs within sequence S (without loss of generality) by adding so-called gaps (empty positions) into sequence T. In the complementary situation when deletion occurs within sequence T, we again add gaps into sequence T. For now, we do not consider duplications or rearrangements. If we would represent all potential mutations and create a complex model, many nowadays known and used algorithms become infeasible.

In Figure 1.3 we can see four examples of alignments between two sequences with different lengths (9 and 4 bases). As before the score of an alignment is the sum of similarity scores for pairs of corresponding nucleotides. To score missing bases (gaps in the alignment), we use a gap penalty; in our scoring system gap penalty equals -1. We do not know which bases corresponds to each other and we want to choose the best alternative. The goal of sequence alignment programs is to find the highest-scoring alignment of two input sequences.

In practice we are interested in two types of alignments. First of them is a *global alignment*. It is an alignment of whole sequences S and T (e.g., genomes of two species or more typically two evolutionary related DNA sequences coding similar proteins). This method can be useful if S and T are similar across their entire length. An illustration of such an alignment is in Figure 1.4. Black parts are the same in both sequences and we want to align them. Because of that we have to shift subsequences of S and T which do not match and create gaps on some positions in S or T.

A highest-scoring global alignment captures the minimum of mutation operations

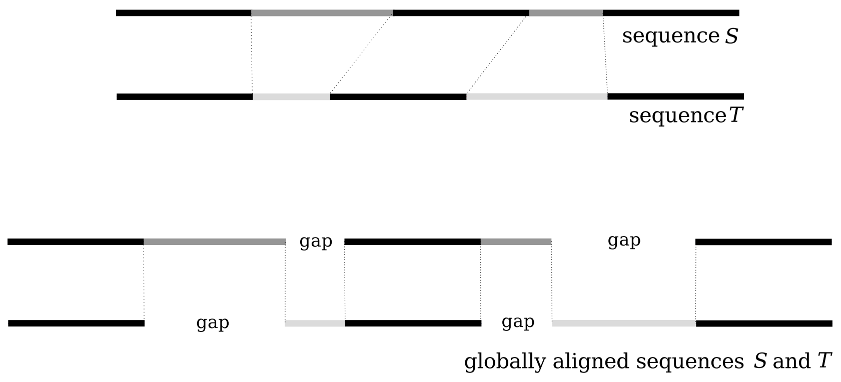


Figure 1.4: Illustration of global alignment.

seq. S	A A *C T A G T* T T T T **A C G T A T** C C
seq. T	C C **A C C G T C T** G G G *C T A C T* A A A

							score
subsequence of S	*C*	*T*	*A*	–	*G*	*T*	
subsequence of T	*C*	*T*	*A*	*C*	–	*T*	
$w(s_i, t_j)$	+1	+1	+1	−1	−1	+1	**2**

								score
subsequence of S	**A**	**C**	–	**G**	**T**	**A**	**T**	
subsequence of T	**A**	**C**	**C**	**G**	**T**	**C**	**T**	
$w(s_i, t_j)$	+1	+1	−1	+1	+1	−1	+1	**3**

Figure 1.5: Sequences S and T uppermost, local alignments between **bold** and *italic* subsequences with the corresponding similarity scores below.

which are necessary to transform one sequence to the other. We suppose that this is the most probable way of evolution, therefore we chose the highest-scoring global alignment as the optimal one.

In this work we will concentrate more on the second type of alignments, a *local alignment*. We showed on the example with the alignment of exon parts of the mouse and human genomes that functional regions stay conserved. If we would try to align whole genomes overall similarity score would be low because of parts between exons (these parts are called introns) where the human and mouse genomes are not similar. This happens despite of high similarity between exons. To find these shorter similar subsequences, we have to compare subsequences of given long

Species	Size in Gb
Human	3.2
Chimpanzee	3.1
Mouse	2.5
Rice	0.42
Yeast	0.012

Table 1.2: Genome sizes of selected species.

sequences and search for high-scoring pairs. The alignment of such shorter pieces is named the local alignment. When we study local alignments of two long genomic sequences we are usually interested not only in one optimal (the highest-scoring) local alignment but also in other high-scoring alignments which we can find in other parts of compared sequences. We illustrate this situation by two sequences S and T in Figure 1.5 which contain two high-scoring local alignments.

In many situations we need to search for local alignments between a query sequence and a database. For example we can try to explore a new DNA sequence and estimate functionality of its parts. As sequence databases grow, it is getting more difficult to efficiently find information we are looking for. Older methods let down in efficiency and therefore new more efficient ones are researched.

In the case of short sequences like those in Figure 1.3 we can try all possibilities and chose the optimal alignment (global or local), but real sequences can be several Megabases long (e.g. see Table 1.2). It is infeasible to compare all possibilities and chose the best one. Therefore, efficient algorithms for finding optimal global alignment were constructed. In following sections 1.3 and 1.4 we describe two standard techniques of search for local alignments. Similar algorithms can be used for global alignment.

1.3 Exact Similarity Search

In this section we describe the Smith–Waterman algorithm for finding the optimal local alignment of two sequences (Smith and Waterman, 1981; Gotoh, 1982). We also show how to extend this algorithm in order to find k best alignments. Finally, we discuss why this algorithm is often infeasible in practice.

This algorithm uses the technique of dynamic programming (Cormen et al., 1989, ch.16). The goal of the algorithm is to compute the highest scoring alignment between two sequences $S = s_1 s_2 \dots s_n$ and $T = t_1 t_2 \dots t_m$. Let $w(s_i, t_j)$ be the score of aligning base s_i of sequence S to base t_j of sequence T and let g be the gap penalty (see Section 1.2 for description of the scoring scheme).

Instead of directly computing the optimal local alignment, the dynamic programming algorithm will compute optimal local alignments of all prefixes of sequences S and T. More precisely, let $c(i, j)$ be the similarity score of the highest scoring local alignment between prefixes $s_1 \dots s_i$ and $t_1 \dots t_j$ containing both bases s_i and t_j (not necessarily aligned to each other), or 0 if such an alignment would have a negative score.

The alignment whose score is in the cell $c(i, j)$ of the table should have in its last column either nucleotide pair s_i, t_j or nucleotide s_i and a gap in sequence T (because t_j was aligned before) or gap in sequence S and nucleotide t_j. If we discard the last column of the alignment, the rest of the alignment must be optimal for corresponding prefixes of sequences S and T. Therefore we can calculate the highest similarity score $c(i, j)$ with a help of cells storing the highest scores for shorter sequences.

To calculate the value of $c(i, j)$, the algorithm chooses the best from the following possibilities:

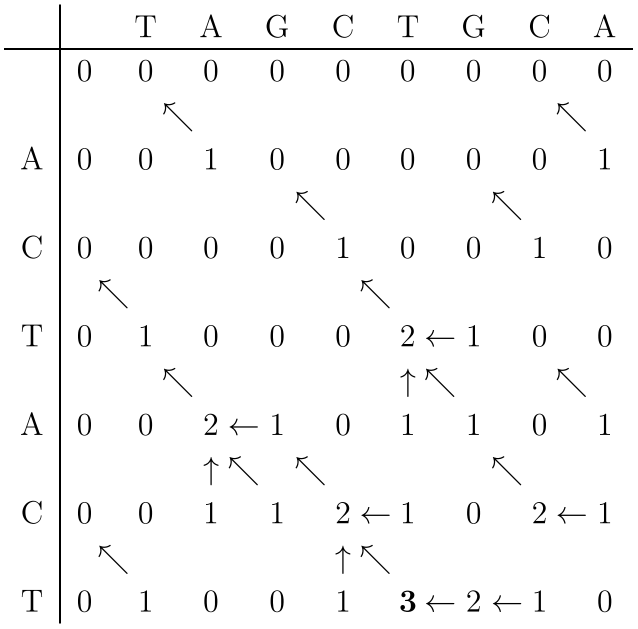
- Nucleotides s_i and t_j can be appended to the alignment which ends in $c(i - 1, j - 1)$. Then the value of $c(i, j)$ will be $c(i - 1, j - 1) + w(s_i, t_j)$, where $w(s_i, t_j)$ may be positive or negative, depending on whether s_i and t_j match or not.

- Nucleotide s_i and a gap in sequence T can be appended to the alignment which ends in $c(i - 1, j)$. Then the value of $c(i, j)$ will be $c(i - 1, j) - g$ (gap penalty).

- Similarly, a gap in the sequence S and t_j can be appended to the alignment which ends in $c(i, j - 1)$. Then a value of $c(i, j)$ will be $c(i, j - 1) - g$ (gap penalty).

- Finally, we also consider an empty alignment with score 0. This may be a start of a local alignment starting at this position.

Formally,



		T	A	G	C	T	G	C	A
	0	0	0	0	0	0	0	0	0
A	0	0	1	0	0	0	0	0	1
C	0	0	0	0	1	0	0	1	0
T	0	1	0	0	0	2	← 1	0	0
A	0	0	2	← 1	0	1	1	0	1
C	0	0	1	1	2	← 1	0	2	← 1
T	0	1	0	0	1	**3**	← 2	← 1	0

Figure 1.6: Smith–Waterman algorithm table for sequences S ="ACTACT" and T ="TAGCTGCA".

$$c(i,j) = \max \begin{cases} c(i-1,j-1) + w(s_i,t_j), \\ c(i-1,j) - g, \\ c(i,j-1) - g, \\ 0. \end{cases} \tag{1.1}$$

Using this equation, the algorithm fills a table in a row-wise order. As a special case, values of $c(i,0)$ and $c(0,j)$ are set to zero for all i and j. Figure 1.6 shows an example of the table $c(i,j)$ for the sequences S ="ACTACT" and T ="TAGCTGCA". The example illustrates the Smith-Waterman algorithm for a simple scoring scheme that assigns the score $+1$ to a match, -1 for mismatch and -1 for a gap.

The highest value of $c(i,j)$ in the whole table marks the end of the optimal local alignment between the sequences S and T. So that the algorithm can reconstruct the optimal local alignment, we have to store not only the score for every cell $c(i,j)$, but also a pointer to the cell from which $c(i,j)$ has been computed. In the example in Figure 1.6, the highest number in the table is $c(6,5) = 3$ (in bold). The optimal local alignment can be reconstructed by following the pointers mentioned above and stopping when score of zero is reached. In our example, this results in the local alignment shown in Figure 1.7.

Additional local alignments with high scores can be found easily from the same

seq. S	T	A	–	C	T
seq. T	T	A	G	C	T

Figure 1.7: Optimal local alignment of given sequences S and T.

table by locating the second, third, fourth, etc. highest score in the table. We are not interested in shorter sub-alignments of already found alignments but only in non-overlapping lower-score alignments. Details of the algorithm can be found in (Waterman and Eggert, 1987).

Since the algorithm requires time proportional to the product of m and n (it has to fill out whole table of this size), the algorithm is infeasible in task of sequence database search. We remark that sequences several Megabases long are sequenced nowadays (see Table 1.2). Therefore we are forced to give up on accuracy of search results in exchange for feasible runtime. In the next section we introduce a well-known heuristic algorithm for local alignment problem.

1.4 Heuristic Search for Similarities

In this section we introduce one of the well known heuristics algorithms called BLAST (Altschul et al., 1990) and discuss its properties and drawbacks. We also review different approaches that achieve better results.

As we said before, a class of biologically relevant questions about sequences homology and function can be attacked with a help of huge sequence databases. Therefore we explain BLAST algorithm in the context of search for similarity between a query sequence S and a target sequence T from a database. Search through the database then contains comparisons of S and every T in the database.

Part One: Search for Seeds

In the first part of the algorithm we search for positions in S and T that may correspond to each other within a high-scoring local alignment. These positions are called seeds. It is faster to decide whether two sequences exactly match than compute their similarity score. Since the speed is crucial in the task of local alignment search, BLAST finds seeds of fixed length k (usually $k = 11$) and it requires complete match between regions in sequences S and T on k consecutive positions. The algorithm

AA	$\rightarrow$	4	19	20
AC	$\rightarrow$	11	14	
AG	$\rightarrow$	8		
AT	$\rightarrow$	5	21	
CA	$\rightarrow$	10		
CC				
CG	$\rightarrow$	15		
CT	$\rightarrow$	2	12	
GA	$\rightarrow$	7		
GC	$\rightarrow$	1	9	
GG	$\rightarrow$	16		
GT	$\rightarrow$	17		
TA	$\rightarrow$	3	13	18
TC	$\rightarrow$	22		
TG	$\rightarrow$	6		
TT				

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
query S	G	C	T	A	A	T	G	A	G	C	A	C	T	A	C	G	G	T	A	A	A	T	C

Figure 1.8: The lookup table created by BLAST to identify hits.

searches for seeds in two steps. First it builds a data structure for faster lookup in the query sequence S and then goes through the target sequence T and searches for seeds between sequences S and T.

The mentioned data structure created from query S is a lookup table where we store all appearances of k bases long consecutive DNA subsequences of the query S. The lookup table is composed of all k-letter words and every word points to positions in query S where the word begins. For example, when $k = 2$ the lookup table stores 4^2 words as shown in Figure 1.8. Occurrences of a word "TA" in query S are shaded in the figure as well as the row in the lookup table where the occurrences are stored.

We use the lookup table to search for hits (matching seeds) in the target sequence T. We go through T and for each continuous k-letter word we use the lookup table to find all its occurrences in S. Each occurrence will be a hit. This process continues according to the algorithm of Wilbur and Lipman (1983). Output of BLAST first part is a list of pairs (i, j) where hit starts on the positions s_i and t_j in query S and target sequence T, respectively.

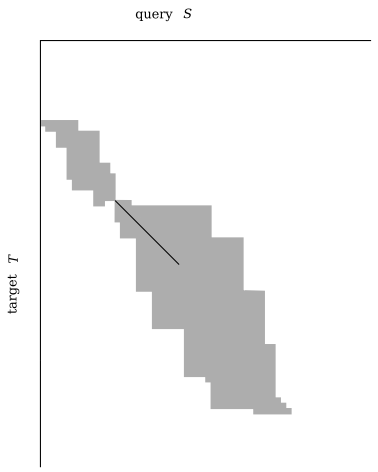


Figure 1.9: Drop-off parameter: The shaded positions are those where a score is above a drop-off parameter and the white are below it. The black line represents a seed.

Part Two: Hits Extension

From the first part of the algorithm we have a list of indexes on which hit (matching seed) starts in S and T. The goal of the second part is to extend hits as much as it gets and those that reach a high enough score will be declared as local alignments.

The algorithm uses a dynamic programming for hit extension from the position (i, j) to both sides. Local alignments with long low-scoring regions are eliminated by a heuristic approach.

The algorithm stops when the actual score decreases below the best score found so far for alignments by a given value. This value is called drop-off parameter and it influences an amount of values $c(i, j)$ which we have to calculate during dynamic programming. In Figure 1.9 we can see an illustration how the heuristic can saves runtime of the algorithm. With this heuristic approach we can lose some alignments because we stop extension of seed too early, but a probability of such a mistake is low and time saving high (Altschul et al., 1990). BLAST uses additional techniques to improve memory and CPU usage of the search.

Properties of BLAST

As we have seen, the BLAST algorithm uses several heuristic rules to increase efficiency of local alignment search. As a result, there might be differences between the solution we want to find (all high-scoring local alignments) and the solution which BLAST finds. On the one hand there are local alignments which BLAST never finds because they do not contain a seed (short high-enough-scoring region). On the other hand, some hits which BLAST finds are not contained in any high-scoring local alignment. For reasoning about quality of BLAST we formalize these two facts.

We call false negatives those homologous alignments which do not contain any hits. BLAST can not find these alignments. False positives are hits which we examine during the extension phase of BLAST but the score of the resulting alignment is too low and the alignment is discarded. The number of different false negatives and false positives influence performance of BLAST in different ways. If the number of false negatives is relatively high compared to the number of found local alignments, the algorithm is not sensitive enough because it lost a big portion of truly homologous alignments. To detect this situation we use a ratio of false negatives to all homologous alignments. False positives do not influence the output but the runtime of BLAST because the algorithm has to check all hits and try to extend them to a high-scoring alignment if it is possible. Therefore we are interested in how often such a spurious hit occurs by chance.

Both of these numbers are influenced by the parameter k, the size of searched hits. On the one hand, if k is small, we find many homologous alignments but it is also easier to find shorter hits simply by chance. It means that for small k the probability of false positives increases. A problem is that runtime of the algorithm is determined mostly by false positives and therefore excessively increases with a growth of false positives count. On the other hand, when k is big the algorithm is much faster, but it also misses more of homologous alignments.

Suppose that sequences S and T are random DNA sequences where every base occurs with equal probability of 1/4. Then, any pair of nucleotides matches with probability 1/4. A hit of length k occurs at each position with probability 4^{-k}. Therefore the expected number of false positives is $O(mn4^{-k})$. Under this simple random model false negatives rate can also be estimated, although it is more complicated (see Brown (2006)).

Weaker seeding requirements

A problem of BLAST is a high ratio of false negative alignments. A way how many approaches handle with this problem is by relaxing a criterion laying on seeds because a high false negatives ratio means that big portion of homologous alignments do not contain k matching consecutive nucleotides. We review this approaches in following text.

The first of this approaches is introduced by algorithm BLAT in (Kent et al., 2002) where mismatches within a seed are allowed. In addition to the parameter k which defines a length of the seed there is a new parameter which specifies the maximal number of mismatches within a seed. If we permit some mismatches within the seed of the same length as before, the probability of false positives grows. Therefore we increase the length of seeds which we search for (usually the length of the seed is set to 15 and 2 mismatches are allowed). The result of this approach is a more sensitive algorithm (with a lower ratio of false negatives) but without an increase of false positives probability.

A different seed requirement modification is defined in (Ma et al., 2002) and used by program PatternHunter. This approach specifies positions within a seed at which mismatches can occur. These seeds are named spaced seeds. We represent every k-letter spaced seed by a string of length k which consists of zeros and ones. A number one in a string at position i means that nucleotides at position i within a hit must match and a number zero at the j-th position means that we do not care whether nucleotides at the j-th position of hit match or mismatch. A spaced seed which is represented by a string of length k which contains only 1's is the same as a seed used in BLAST. Therefore we can see spaced seeds as a generalization of BLAST seeds but there is no a way how to represent BLAT seeds.

The last approach which we review is from Brejová et al. (2005) and provides a general view on all formerly described seeds. These last seeds are called vector seeds and their definition follows.

Let A is an ungapped local alignment of sequences S and T and its length is k. Then a scoring vector V of the alignment A is k-tuple which consist of similarity scores for aligned nucleotide pairs. A vector seed Q is a pair (v, t) where v is k-tuple of ones and zeros which specifies meaningful positions of k-letter seed by ones and unimportant positions by zero. A hit is detected when alignment A satisfies a condition $V \cdot v \leq t$. It means that dot product of vectors V and Q exceeds a threshold t.

In the case of DNA sequences and scoring system with $+1$ for match and -1 for mismatch, the vector seed $((1,1,1,1,1),3)$ requires four matches from five consecutive positions and therefore corresponds to BLAT seed of length 5 with one mismatch. Seeds which are used in BLAST corresponds to vector v which consist of k ones and threshold t equals k. Finally, vector seeds of form $((1,1,0,1,1),4)$ corresponds to spaced seed of length 5 with possible mismatches on the third position. These examples we show that it is easy to construct a corresponding vector seed to any of the formerly described seeds.

The more general seeding requirement is used in a search algorithm, the better are algorithms sensitivity and specificity. The formalization of vector seeds brought an efficient algorithm that estimates sensitivity and specificity.

1.5 Statistical Significance

In the two previous sections we have shown how to search for highest-scoring local alignment of query S and a target sequence from a database. When we have such an alignment, the important task is to decide whether the reached similarity score is statistically significant. If this is the case, we will say that the aligned sequences are related. Otherwise, such a high score can appear often by chance and therefore we say that the aligned sequences are unrelated.

The statistical significance of a similarity score is typically given in the form of P-value. In this context, P-value is the probability that such or higher score appears simply by chance between a random query and a random database. Alternatively we can use E-value, which is the expected number of such or higher score occurs in a random database only by chance. We assume that the lower is P-value or E-value of a particular alignment score the higher is probability that the score detects related sequences because occurrence of the score or higher in the random database is highly improbable. To specify this probability, we need a probabilistic model of DNA sequences and in the following text, we show two sequence models usually considered in P-value estimations. We also provide the commonly used formulas for calculations of P-value and E-value.

Probabilistic Model of Biological Sequences

In a probabilistic model of sequences, sequence $S = s_1 s_2 \dots s_n$ is an intersection of n events of the following form: the character on the i-th position of S is exactly s_i for $i = 1, 2, \dots, n$. Let X_i is a random variable for the character on the i-th position. In general, the probability that character s_i occurs on i-th position can be affected by letters on other positions. By multiple application of the multiplicative law of probability ($P(A, B) = P(B|A)P(A)$) on the probability of sequence S which we denote $P(S)$, we get:

$$\begin{aligned} P(S) &= P(X_n = s_n, X_{n-1} = s_{n-1}, \dots, X_1 = s_1) \qquad (1.2) \\ &= P(X_n = s_n | X_{n-1} = s_{n-1}, \dots X_1 = s_1) \times \\ &\quad P(X_{n-1} = s_{n-1} | X_{n-2} = s_{n-2}, \dots X_1 = s_1) \times \\ &\qquad\qquad \vdots \\ &\quad P(X_2 = s_2 | X_1 = s_1) \times \\ &\quad P(X_1 = s_1). \end{aligned}$$

Most often considered models of probabilistic sequence are two simplifications of this general approach; the first is a model with independent and identically distributed (IID) bases, and the second is Markov chain. In the IID model, each nucleotide of the considered sequence is supposed to be drawn from the same background probabilistic distribution independently on any other position of the sequence. The probability $P(S)$ from (1.2) simplifies to $\prod_{i=1}^{n} P(X_i = s_i)$. In the more complex case of Markov chains, local dependence of variable X_i on several previous variables is allowed. Chains where a variable X_i depends on m previous variables is called Markov chain of order m and probability of sequence S generated by a Markov chain of order m is:

$$\begin{aligned} P(S) \quad = \quad & P(X_n = s_n | X_{n-1} = s_{n-1}, \dots, X_{n-m+1} = s_{n-m+1}) \times \dots \\ & P(X_m = s_m | X_{m-1} = s_{m-1}, \dots, X_1 = s_1) \times \dots \\ & P(X_1 = s_1). \end{aligned}$$

Significance of Single-sequence Runs

The simplest scenario is a single-sequence case, in which a significance of high-scoring segment is the asked question. For a single sequence $T = t_1t_2 \dots t_n$ and a scoring function that assigns a score to each symbol of T, we are interested in high-scoring regions within T. For example, one can use this approach to detect long runs of a particular letter l using scoring function that assigns $+1$ to the letter l and $-\infty$ to any other letter.

This scenario for IID sequences is considered in two papers (Karlin et al., 1990; Dembo and Karlin, 1991b). Assume that expected score for a letter is negative and there is a chance obtaining a positive score. Let $S(n)$ be a random variable representing a score of the highest-scoring segment of T and let w be a threshold for the scores we are interested in. In the first paper, the authors have shown, that a probability of a segment with a score exceeding w is:

$$P\left(S(n) - \frac{\ln n}{\lambda^*} > w\right) \approx 1 - \exp(K^* e^{-\lambda^* w}), \tag{1.3}$$

where λ^* and K^* are some particular computable constants dependent on the scoring function and the IID sequence model assumption. From this follows, that the asymptotic distribution of the number of segments obtaining a score at least $w + \ln n/\lambda^*$ is Poisson with parameter $K^* e^{-\lambda^* w}$. In the second paper, strong limit theorems for a length of the highest-scoring segment are provided. Similar results were shown in (Karlin and Dembo, 1992; Dembo and Karlin, 1991a) for the case when Markov chain is the underlying model of sequence T.

Significance of Local Alignment

Generalization of the previous scenario leaded to formalization of the P-value calculations for an ungapped local alignment search. Consider a query sequence $Q = q_1q_2 \dots q_n$ and a database sequence $D = d_1d_2 \dots d_m$, both IID with nucleotide probabilities $p_q = \{p^q_A, p^q_T, p^q_C, p^q_G\}$ and $p_d = \{p^d_A, p^d_T, p^d_C, p^d_G\}$, respectively. Let w be a score threshold, and we want to know a probability of Q having the best match with score at least w in database D. The pair of nucleotides a in the query and b in the database occur with probability $p^q_a p^d_b$ and we have a scoring scheme that assigns a score s_{ab} to such a pair. Assume that the expected pair score $\sum p^q_a p^d_b s_{ab}$ is negative and there is a chance obtaining a positive score.

Consider the two following restrictions: the probability distributions p_q and p_d are relatively similar to each other, and sequence lengths m and n grow at roughly equal rates. Then the random variable $S(mn)$ (the maximal local alignment score) has the following approximating distribution (Dembo et al., 1994):

$$P(S(mn) - \frac{\ln mn}{\lambda^*} > w) \approx 1 - \exp(K^* e^{-\lambda^* w}), \tag{1.4}$$

where λ^* is the unique positive solution of the equation $\sum_{a,b} p_a^q p_b^d \exp(\lambda s_{ab}) = 1$ and a parameter K^* is some positive constant. If the restrictions are not fulfilled, the formula 1.4 overestimates the probability of high-scoring local alignments and that leads to conservative significance estimates. For large w, it holds that:

$$P\left(S(mn) - \frac{\ln mn}{\lambda^*} > w\right) \leq K^* e^{-\lambda^* w}.$$

Although an exact mathematical apparatus is not built for the case of gapped alignments, in several papers it was empirically shown (e.g. Collins et al., 1988; Waterman and Vingron, 1994) that highest scores of gapped alignments approximately follow Gumbel distribution. The parameters K_g and λ_g (where g stands for gapped) have to be calculated empirically.

Estimating *E*-value

The formulated results about the maximal score $S(n)$ distribution imply that the asymptotic distribution of the number of non-overlapping segments with score at least $x = \ln(n)/\lambda^* + w$ is Poisson with parameter $K^* n e^{\lambda^* x}$ in the case of single sequence of length n. In the case of two sequences the parameter of Poisson distribution is $K^* nm e^{-\lambda^* x}$, for n and m being lengths of the two sequences and $x = \ln(mn)/\lambda^* + w$. The parameters $K^* n e^{-\theta^* x}$ and $K^* nm e^{-\theta^* x}$ of Poisson distributions are also expected values of these distributions. The expected value is called E-value.

In the following chapter we discuss methods to improve estimates of statistical significance, and propose our own approach to this task.

Chapter 2

Effective DNA Database Size

In this chapter, we study the problem of estimating statistical significance of a given sequence alignment. In Section 1.5 we have reviewed the traditional model of statistical significance. Recall that P-value of an alignment is is the probability that score of a value such or a higher score appears simply by chance between a random query and a random database. If the model used to obtain the estimate of P-value does not capture important properties of the real database, we may obtain unnecessarily pessimistic P-values and reject many meaningful alignments.

We explain drawbacks of the traditional model, and propose to use a lower effective database size instead of its real size. The further text is organized in six sections. Our motivation and work related to this topic are described in Section 2.1. In Section 2.2 we propose to estimate the effective size of a database by its compressed size. In Section 2.3 we explain our assumptions about homology search and about artificial databases, which we will use in the following experiments, and explain details of an algorithm that we have implemented for the exact P-value calculations. Each of the following two sections 2.4 and 2.5 deals with P-value calculations for a particular type of artificial databases, and we combine the obtained information about the artificial databases to estimate P-values for the real genomic databases in Section 2.6.

2.1 Motivation and Related Work

Databases of biological sequences such as GenBank (Benson et al., 2008), contain large amounts of internal redundancy. Exact duplicates are usually removed, but there is an evidence of many almost identical sequences. Nearly identical sequences occur in the sequence databases due to redundancy of sequence data production or evolutionary relationship. The redundancy which scientists bring to the databases can be caused for instance by partial overlapping subsequences of the same long sequence sequenced by different groups. Errors that appear during sequencing may prevent detection of duplicates. However, the sequences can be very similar to each other also due to their evolutionary relationship. For illustration, assume that short aligned regions in genomes of human, chimpanzee, mouse, dog, frog and fish are as follows:

human	`AATAC`
chimpanzee	`AATAG`
mouse	`AAAAG`
dog	`AAAAA`
frog	`ACCAC`
fish	`ACCAG`

We suppose relationship among species as is shown in the phylogenetic tree in Figure 2.1. Using the known sequences of the current species we can estimate the sequences that were present in the ancestral species in the same region of the genome. As we can see, some nucleotides can stay unchanged in all species or at least in large portions of the tree. Therefore, present-day DNA sequences are clustered around the ancestral sequence.

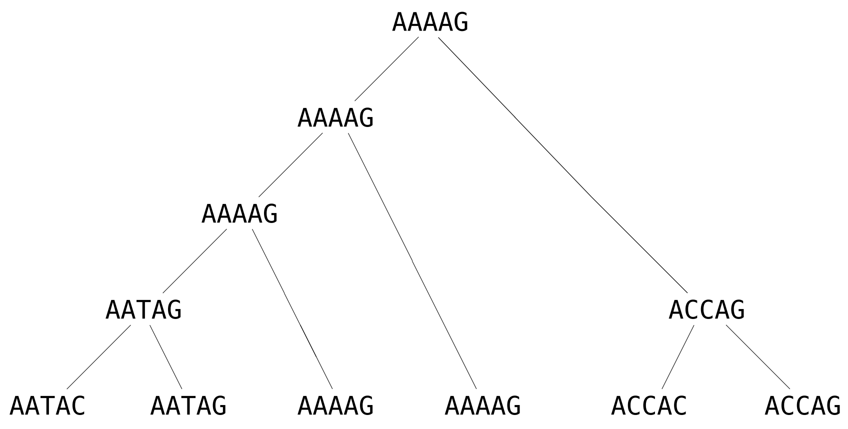


Figure 2.1: Corresponding segments of DNA sequence from different species (leaves of the tree) and estimated corresponding sequences of common ancestors (internal nodes).

Redundancy in the sequence database can be the cause of various biases in the statistical analysis of the sequence data. If much more sequences similar to sequence S occur in a database than sequences similar to some other sequence T, the sequences similar to S are over-represented and given higher weight in a statistical analysis that assume that the sequences are independent. Therefore, databases which contain only a representative set of sequences are smaller, search in them is faster than search in the origin databases and sequences fit better to the IID probability model which is used as a background for P-value estimation.

These representative data sets are usually created in the following way. First, sequences with similarity above the user specified threshold are grouped together. Then, one representative sequence is chosen from each group and copied to the new database. We now shortly explain two algorithms used for this purpose.

UniRef. The UniRef databases (Suzek et al., 2007) of representative protein sequences are created from UniProtKB which is a repository of all known protein sequences. The UniRef100 database combines identical sequences and sequence fragments from the same organism and stores them in one database entry with information about all merged sequences and links to UniProtKB.

Databases UniRef90 and UniRef50 are built from UniRef100 by clustering the sequences at the 90 % and 50 % sequence identity levels. For building the UniRef90 and UniRef50 databases, sequences are sorted in order of decreasing lengths. The first sequence is chosen as the representative sequence C_1 of the first cluster. Every other sequence S is compared to previously chosen representative sequences C_1, $C_2, \ldots, C_n$. If the similarity of S with one of the representatives C_i, is above the sequence identity level threshold, sequence S is added to the cluster represented by C_i. Otherwise, sequence S becomes a new representative sequence, C_{n+1}. The algorithm for building UniRef90 and UniRef50 is called Cd-hit (Li and Godzik, 2006).

UniqueProt. The UniqueProt (Mika and Rost, 2003) is a tool for selecting an unbiased representative subset from a given set of sequences. First, all versus all sequences are compared. Authors define their own similarity measure among aligned proteins based on their alignment length and percentage of identical amino acids. Sequences with similarity above a given threshold are clustered together. Finally, a list of representative sequences is created in the following way. Sequences which are alone in a cluster are added to the list and from each bigger cluster one representative sequence is chosen.

Related work. Consider calculation of E-value from Section 1.5. Usually, instead of actual sequence lengths, shorter effective lengths are used for the following reason. If a maximal scoring segment (in the case of single sequence) starts near the end of sequence it can run out of the sequence before it reaches a score which defines this segment as maximal scoring. Therefore an effective length $n = n_o - l$ is used, where n_o is the original length and l is the typical length of a maximal scoring. Similarly in the case of local alignment, effective lengths of both aligned sequences are used.

To estimate length l, the following approaches were proposed: Altschul et al. (1996) suggested estimation of l by $\ln(K_g nm)/H$, where H is the relative entropy of the used scoring system, m and n are sequence lengths, and K_g is a parameter calculated for gapped alignments. Later, Altschul et al. (2001) showed approximately linear dependence of length l on the score x of the alignment, $l \approx \alpha x + \beta$ and proposed how to estimate constants α and β from simulations.

E-value estimation is better if the underlying sequence random model match to the real sequence database and worse if the model and the database do not match well. One example when the two things do not match is a database containing high fraction of low-complexity regions. A low-complexity region contains some bases in very high frequencies and the other bases in very low frequencies, creating a bias towards the first group. If we want to use the method from Section 1.5 we somehow want to transform each real sequence from the database so that the resulting database is more similar to the random sequence model.

One of the commonly used tools for decreasing the bias towards some amino acids is low-complexity filter. Using this tool a biological sequence is divided into short regions. We say that a region is of a high complexity when approximately the same number of different bases occur in the region. Otherwise, we say that the region is of a low complexity. For each region a single value which measures complexity of the region is calculated. The higher is the value, the higher complexity. If the complexity measure decreases under the specified threshold, then we mask the letters in the small part of sequence with a special symbol which means that any of the bases could be on the position in the sequence. The low-complexity regions of DNA sequences and proteins slightly differs, and therefore the algorithms for their filtering (Morgulis et al., 2006; Wootton and Federhen, 1993) differ as well.

Schäffer et al. (2001) proposed another method to improve accuracy of E-value estimation. The method is based on scaling parameter λ^* according to estimated differences in base occurrence between the random model database and the real one (which optionally can be filtered by the low-complexity filter).

These examples illustrate that even simple preprocessing of the input data, or adjustment of the underlying sequence model parameters can improve quality of significance estimates.

Problem visualization. Consider sequence database D, query Q, and an illustration in Figure 2.2. Suppose that the best similarity score of Q and D is s. We can visualize individual sequences in D as points in the sequence space and the neighborhoods containing all sequences with sequence similarity score s or higher as the shaded areas around the points. In this sequence space, the query Q is located at the boundary of one of these neighborhoods. If the neighborhoods cover a large fraction of the sequence space, the P-value is high, because a randomly generated query will have a high probability of falling into one of the neighborhoods and having the similarity score greater than s.

The illustration on the right shows a database with clusters of similar sequences that have overlapping neighborhoods which cover a much smaller fraction of the sequence space than the neighborhoods of a random database on the left. Hence, the P-value estimate based on the assumption that database sequences are uniformly distributed in the sequence space will necessarily overestimate the P-value. This can potentially leads to a rejection of high scoring matches as the matches which likely occurred by chance.

Effective database size. One of the main parameters in P-value computation is the database size n. Instead of the real database size, we propose to use an *effective database size* n' smaller than n such that it will account for redundancies in the database. For example, if we take a random database of size n and double its content by adding second exact copy of each sequence, we expect the effective database size still be n, even though its real size is $2n$.

Usage of effective sizes instead of the real ones is not unusual in theoretical modeling of biological phenomena; we have already seen one example of its usage to compensate for possibly broken high-scoring alignments at the ends of database

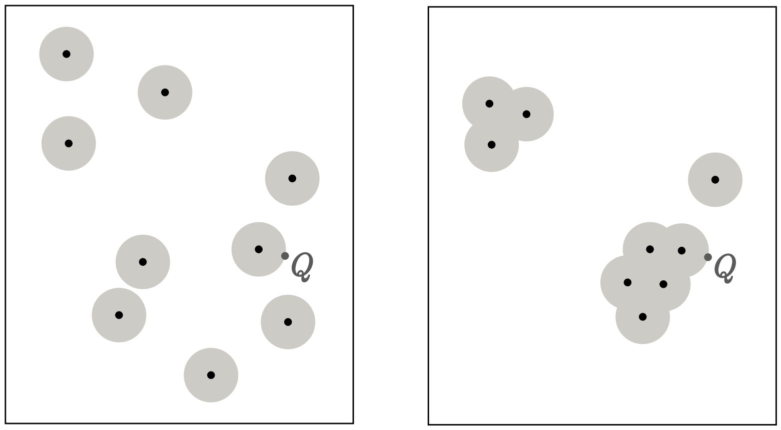


Figure 2.2: Illustration of database space. The points represent sequences in database D and the shaded areas show their neighborhoods with scores at least as good as the similarity score of query Q. Left: Database with randomly distributed sequences. Right: Database with clustered sequences.

sequences. Other example are statistical models in population genetics, where population size is used as a parameter, but instead of the actual number of individuals, effective population size is typically used. Its use compensates for various effects that are not considered by the model, such as population size changing over time (Hartl and Clark, 2006).

Several software tools providing P-value estimates for homology search results, for example BLAST (Altschul et al., 1990), have an option for setting an arbitrary value as the database size. This makes effective database size easily implementable in the existing tools, if its usefulness would be proven.

2.2 Kolmogorov Complexity and Genomic Sequence Compression

The effective database size should be an estimate of the amount of unique sequence in the database D, taking into account substrings that may be present in D in many exact or approximate copies. One way of describing the information content of a database is its *Kolmogorov complexity* (Li and Vitányi, 2008).

Kolmogorov complexity $K(D)$ of a sequence D is the bit length of the shortest program P for a fixed universal Turing machine that outputs sequence D. Kolmogorov complexity can be understood as a lower bound (up to a constant additive term) of compression achievable by any general-purpose algorithm. A string of length n sampled uniformly at random from a fixed alphabet is on average almost incompressible (Li and Vitányi, 2008, Section 2.8.1). In particular, a string over a four-letter alphabet requires on average approximately $2n$ bits (with up to an $O(\log n)$ additive term). Thus if we believe that databases with the same Kolmogorov complexity will behave similarly, we should use $n' = K(D)/2$ as an estimate of the effective database size of database D.

In this context, Kolmogorov complexity seems to be an ideal estimator to use. It accounts for possible major differences between the real database and a randomly generated one. In particular, using Kolmogorov complexity would compensate for: 1) differences in frequencies of individual nucleotides (database containing only a long stretch of As should not have a large effective size), and 2) redundant sequence content (sequences that have only few differences can be described very efficiently in small space). Moreover, the concept of Kolmogorov complexity has been successfully used in similar contexts before, for example to compute distance between genomes (Li et al., 2001) (see also Giancarlo et al. (2009); Nalbantoglu et al. (2010) for overview).

The exact Kolmogorov complexity of database D is not computable. Yet in practice, we can use various compression algorithms instead of computing the Kolmogorov complexity. For a fixed compression algorithm, the compressed size $c(D)$ of database D is an upper bound on the Kolmogorov complexity $K(D)$ (up to a constant term related to the size of the decompression algorithm). We can use value $n' = c(D)/2$ as an estimate of the effective database size. The upper bounds on P-values, usually called *conservative estimates*, are generally desirable in cases where exact P-values cannot be computed. Since the P-values increase monotonically with the database size, using an upper bound on the effective database size should not by itself lead to non-conservative bounds.

As the number of available genomic sequences grows very fast, techniques for DNA sequence compression are vividly studied, and several algorithms for this task are available nowadays (e.g. Chen et al., 1999, 2002; Korodi and Tabus, 2005; Behzadi and Le Fessant, 2005; Wang and Zhang, 2011; Deorowicz and Grabowski, 2011; Pinho et al., 2012). The file compression algorithms usually identify symbols or small groups of symbols that occur in the file more frequently and encode them

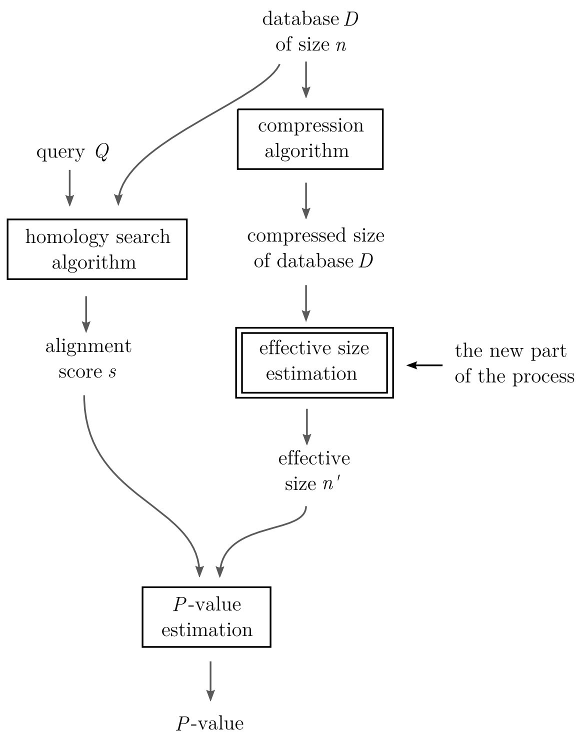


Figure 2.3: Generic schema using existing tools.

by shorter codewords. Therefore, the size of the compressed database depends on the entropy of the source that generated the database. In addition to this, some of the compression algorithms identify the exact or the approximate repeats, store only one occurrence of the repeat, and encode the approximate repeats as sequences of differences from the stored repeat.

In our experiments, we try to separate these two features of the real genomic databases creating two different types of the artificially generated databases. The first types are databases consisting of many almost perfect copies of a random sequence to mimic the issue with the repeated segments. We discuss these experiments in Section 2.4. Second, in Section 2.5, we generate databases of various entropy levels. For this type of databases we also provide formulas for approximate calculation of P-values. Finally, we will also consider real DNA sequences, where both entropy and repeats play a role. For all database types, we compare P-value estimates based on the effective database size and the estimates which consider the real database size.

Motivated by the discussion in the previous text, we explore the use of compression software for estimating the effective database size. The methodology is very simple. First, we compress the database and measure the size of the resulting file in bytes. Then, we multiply this size by four to account for the fact that in a uniformly random database, we need two bits to encode each nucleotide. In this way, we obtain an estimate of the effective database size which can be used in any formula or algorithm for estimating P-values on uniformly distributed databases. In a scheme in Figure 2.3 the traditional way of P-value estimation is represented by the left path through a diagram, and the new approach follows the right path.

Unfortunately, Kolmogorov complexity and its compression estimates do not necessarily lead to conservative bounds in all instances. As an extreme case, base-4 expansion of many fundamental constants, such as π, can be generated by a constant-size program. First n bits generated by such a program can be used as a sequence database of size n, replacing digits $0, \ldots, 3$ with nucleotides. Kolmogorov complexity of a such database is $O(\log n)$ (we need a constant number of bits to represent the program, and $\log n$ bits to represent the real size of the database), yet for all practical purposes, this database behaves as a random database of size n (Bailey and Crandall, 2002).

Thus using Kolmogorov complexity and compression-based estimates of effective database size may lead to non-conservative estimates of P-values in homology search. However, we will show that in practical applications this problem can be addressed by a simple adjustment.

2.3 Exact *P*-value Computation in a Simple Scenario

In order to evaluate our compression-based estimates, we will consider a simplified homology scenario, where P-value can be computed exactly and then compared with our estimate. This scenario is motivated by a real application. With the next generation sequencing technologies available, a sequencing machine generates many short sequences called *reads*. If we have the reference genome of a particular species at hand, the classical approach is to map the obtained reads to the reference. Most of the reads map to the reference sequence exactly, but sometimes we can see a few mismatches. These mismatches may be caused either by sequencing errors, or (more interestingly) by differences between the reference and the currently sequenced individual. Assigning a P-value to each match of a read within database D, we can filter out the reads of low quality and analyze only matches, which according to their P-value are not likely to occur by chance.

Motivated by this scenario, we create a simplified model, where database D is a single string of length n, the query Q is a string of length m. We mimic the homology search by a search for a substring of D with length m, which has the smallest Hamming distance from Q. In contrast to the full homology search problem, insertions and deletions are forbidden in D or Q and we search for the best match of the whole query Q, as each read has to map completely to the reference.

We will say that the minimum Hamming distance between Q and some substring of D with length m is the *distance of Q and D*. Then for a particular Hamming distance h, we calculate the P-value as the number of all possible m-tuples with distance at most h from D divided by 4^m (the number of all possible m-tuples). For the P-value calculation, we keep the database fixed, and the query is chosen randomly from a uniform distribution of all m-tuples over the nucleotide alphabet. In contrast, most of the classical approaches consider both query and database as random.

The main advantage of this model is that we can compute P-values exactly in a reasonable time, compare them to the P-value estimates and this way evaluate the accuracy of our concept.

Database models for separation of two phenomena effecting compression. The file compression algorithms usually identify symbols or small groups of symbols that occur in the file more frequently and encode them by shorter codewords. Therefore, the size of the compressed database depends on the entropy of the source that generated the database. In addition to this, some of the compression algorithms identify the exact or the approximate repeats, store only one occurrence of the repeat, and encode the approximate repeats as sequences of differences from the stored repeat.

In our experiments, we try to separate these two features of the real genomic databases creating two different types of the artificially generated databases. The first type are databases generated with various entropy levels, we discuss them in Section 2.5. For this type of databases we also provide formulas for approximate calculation of P-values. Second, in Section 2.4, we generate databases consisting of many almost perfect copies of a random sequence to mimic the issue with the repeated segments. Finally, we will also consider real DNA sequences, where both entropy and repeats play a role. For all database types, we compare P-value estimates based on the effective database size and the estimates which consider the real database size.

Algorithm to Compute Exact *P*-values

We have implemented an algorithm that for a given query length m and database D computes simultaneously P-values of all prefixes of D and for all distances $h = 0, 1, \ldots, h_{\max}$. In experiments we use $m = 15$ and $h_{\max} = 3$.

The algorithm maintains two arrays M and H while sequentially processing each base of the database D. Array M of size 4^m stores the distance to the already processed prefix of D for each possible m-tuple. If the distance is more than $h_{\max}$, the algorithm uses a special ∞ value for the m-tuple, as we do not need to distinguish among the distances bigger than $h_{\max}$. The second array H stores for each distance $h \leq h_{\max}$ the number of m-tuples, for which their distance from the already processed prefix of D equal to h.

While processing a new nucleotide of D, the last m-tuple of the current prefix of D is the only one which should be taken into account in updating arrays M and H. Let us denote new m-tuple as T. We enumerate all m-tuples at a distance of at most $h_{\max}$ from T and for each such m-tuple S, the arrays M and H are updated as follows; Suppose that Hamming distance between T and S is k. If an actual value of $M[S]$ is greater than k, we assign $M[S] = k$. We also increase value $H[k]$ by 1. Consider now query Q which has distance h from database D. From the values stored in array H, we can easily calculate P-value of Q as a sum $\sum_{k=0}^{h} H[k]$ divided by 4^m, the number of all distinct m-tuples. This algorithm is feasible only for small values of m, as it requires $\theta(4^m)$ memory.

We will use this algorithm for P-value calculation in three different ways:

- First, we calculate exact P-value for an available database D, which contains either the simulated data or the real genomic sequences. Having this *real*

P-value at hand, we are able to compare quality of estimates made by different approaches.

- Second, we estimate the effective size of database D from its compressed size. We then use the algorithm on a random database of a size equal to the effective size of D. This way we obtain the P-value *predicted* by the proposed approach based on the effective database size.

- Finally, we use the algorithm to calculate P-value of a random database with the same size as D. This *simple* P-value corresponds to the commonly used estimate of P-value, when the real size of D is considered by the estimation.

The random database in the second and third approach contains independent uniformly generated bases. In real applications, predicted P-values would be computed by estimates based on extreme value theory or other fast methods rather than the exact algorithm.

2.4 Redundant Databases

We consider generated databases that are concatenation of many mutually similar sequences of the same length k. In the experiments, we use $k = 10^4$. As the first step while creating such a database D, we sample a sequence $S = s_1 s_2 \dots s_k$ from uniform nucleotide frequencies, independently at each position. This sequence will be the center of a cluster of similar sequences stored in D. We obtain each sequence concatenated in D from S by randomly mutating nucleotides on several positions of S such that the nucleotide on position j is the same as s_j with the probability of 90%, and with the probability of 10% it mutates to a nucleotide randomly selected from the other three. This way, we get a *clustered database* of sequences that differ on average on 10% positions from the center of the cluster.

We have tested the P-value estimations on clustered databases of various sizes and results in Table 2.4 are averaged over 5 different databases. For each clustered database D we compute the real P-value, the simple estimation of the P-value, which uses the effective size equal to the real size of D, and two estimates based on two different lossless compression programs GenCompress (Chen et al., 1999) and bzip2.

	h=0				h=2			
Database size	10^4	10^5	10^6	10^7	10^4	10^5	10^6	10^7
Real	$9.3 \cdot 10^{-6}$	$8.1 \cdot 10^{-5}$	$6.5 \cdot 10^{-4}$	$4.3 \cdot 10^{-3}$	$9.2 \cdot 10^{-3}$	$6.8 \cdot 10^{-2}$	$3.4 \cdot 10^{-1}$	$8.8 \cdot 10^{-1}$
GenCompress	$9.3 \cdot 10^{-6}$	$7.7 \cdot 10^{-5}$	$6.8 \cdot 10^{-4}$	$6.5 \cdot 10^{-3}$	$9.2 \cdot 10^{-3}$	$7.3 \cdot 10^{-2}$	$4.9 \cdot 10^{-1}$	1.0
bzip2	$1.0 \cdot 10^{-5}$	$9.2 \cdot 10^{-5}$	$8.1 \cdot 10^{-4}$	$8.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-2}$	$8.7 \cdot 10^{-2}$	$5.5 \cdot 10^{-1}$	1.0
Simple	$9.3 \cdot 10^{-6}$	$9.3 \cdot 10^{-5}$	$9.3 \cdot 10^{-4}$	$9.3 \cdot 10^{-3}$	$9.2 \cdot 10^{-3}$	$8.8 \cdot 10^{-2}$	$6.0 \cdot 10^{-1}$	1.0

Table 2.1: P-values for the artificial clustered database.

Bzip2 is based on Burrows-Wheeler transform (Burrows and Wheeler, 1994) designed for textual data. If the compressed text contains several exact repeats of a particular substring, the transformation reorganizes letters of the text so that the corresponding letters of the repeated sequences are moved together, creating blocks of the same letter. This transformed text can be encoded by other compression techniques, such as run-length encoding.

GenCompress (Chen et al., 1999) is an algorithm developed specifically for compressing DNA sequences. It passes through the sequence only once, simultaneously identifying approximate repeats and encoding them by a sequence of edit operations. As GenCompress proceeds along the database, it seeks for the best prefix of the non-processed part, which can be encoded as an approximate repeat of some subsequence from the previously processed part of the database.

In this experiment, the estimates based on data compression are mostly conservative and better than the estimates obtained by the simple method. We can also see that the GenCompress estimates usually outperform the estimates made by bzip2.

2.5 Databases of Various Entropy Levels

The four nucleotides do not occur in genomes with the same frequencies. A commonly used measure of DNA composition is GC-content: the percentage of Cs and Gs in the given sequence. As GC-content varies in different genomes and also among segments of the same genome, we explore the influence of GC-content of a given database on the P-value prediction. Note that for a discrete random variable X with n outcomes $x_1, x_2, \ldots, x_n$, the Shannon entropy $H(X)$ is defined as

$$H(X) = -\sum_{i=1}^{n} p(x_i) \lg p(x_i)$$

where $p(x_i)$ is a probability of x_i. Therefore a random database with GC-content g and independently generated bases has entropy $-g\lg(g/2) - (1-g)\lg((1-g)/2)$, which we denote by $H(g)$. Therefore the database can be encoded by approximately $H(g)n$ bits, for example by arithmetic encoding (Rissanen and Langdon, 1979). Following our general approach, we will use the uniform nucleotide frequencies and effective database size $E(n, g) = H(g)n/2$ to estimate the P-values for the actual database of size n and GC-content g. We evaluate this approach on simulated data but also analytically by deriving formulas for P-value approximation.

2.5.1 Approximation of *P*-values

Consider random database D of length n and GC-content g. We will derive formulas for computing P-value of obtaining Hamming distance at most h for a random query of length m. Note that here both query and database are random. While database is generated from a given GC-content g, query is chosen uniformly from all 4^m possibilities.

Let S be a continuous subsequence of database D having length m, and Q be a random query. Suppose that there is base G or C on some position of S. The probability of the base being aligned to the same base in query Q is $g/2$ and a probability that the aligned bases differ is $1 - g/2$. For the position of S containing base A or T, the probability that the base aligns to the same base in Q is $(1-g)/2$ and a probability that the bases differ is $(1+g)/2$. Let Q be a query containing z Gs and Cs. Let X be the event that query Q has an occurrence with distance exactly k at a particular position in D. Then we can express probability of X (over random database D with GC-content g) as:

$$P(X|k, Q, g) = \sum_{i=0}^{k} \binom{z}{i} \left(1 - \frac{g}{2}\right)^{i} \left(\frac{g}{2}\right)^{z-i} \binom{m-z}{k-i} \left(\frac{1+g}{2}\right)^{k-i} \left(\frac{1-g}{2}\right)^{m-z-(k-i)} \tag{2.1}$$

$$= \left(\frac{1-g}{2}\right)^{m} \left(\frac{1+g}{1-g}\right)^{k} \left(\frac{g}{1-g}\right)^{z} \sum_{i=0}^{k} \binom{z}{i} \binom{m-z}{k-i} \left[\frac{(2-g)(1-g)}{g(1+g)}\right]^{i}. \tag{2.2}$$

In each of the products in expression 2.1, we distinguish between i mismatches on G/C positions represented by the first three factors and mismatches on $k-i$ A/T positions in Q represented by the second three factors. For $g = 0.5$, this probability expectably simplifies to

$$P(X|k, Q, g = 0.5) = \binom{m}{k} \left(\frac{3}{4}\right)^k \left(\frac{1}{4}\right)^{m-k}$$

using Vandermonde's convolution (Graham et al., 1989).

Let Y be the event that query Q has an occurrence with distance at most h at the particular position in D. The probability $P(Y|h, Q, g)$ as well as the expected value $E(Y|h, Q, g)$ equals to a sum of probabilities $P(X|k, Q, g)$ for $k \in 0, 1, \ldots, h$ as $Y \sim \mathit{Bernoulli}(P(Y|h, Q, g))$. If event Y happens at the particular position of D, we say that the position is h-close to Q.

Let Z be the number of h-close positions in D for Q. The expected value of Z can be computed by linearity of expectation (for simplicity we ignore the edge effect at positions $n - m + 2, \ldots, n$, which is negligible for large n):

$$E(Z|h, Q, g, n) = \sum_{i=1}^{n} E(Y|h, Q, g) = nP(Y|h, Q, g). \tag{2.3}$$

We will approximate the distribution of variable Z by the Poisson distribution with mean $\lambda = E(Z|h, Q, g, n)$. This approximation disregards dependencies between occurrences at adjacent positions and also assumes that n is large and λ relatively small. If Z is from the Poisson distribution, the probability of at least one occurrence of a given query Q is $P(Z > 0|h, Q, g, n) = 1 - e^{-\lambda}$. To obtain the final P-value, we have to consider $P(Z > 0|h, Q, g)$ for different groups of queries with the same number Gs and Cs, of which Q_z is one representative:

$$P(Z > 0|h, g, n) = 2^{-m} \sum_{z=0}^{m} \binom{m}{z} P(Z > 0|h, Q_z, g, n) \tag{2.4}$$

We compute two different P-values calculated according to the expression 2.4: The first of them is $P_{real}(h, g, n)$ calculates P-value for the GC-content g and the real database size n. The second one is $P_{est}(h, g, n)$, the compression estimate which uses GC-content 50 % and effective database size $E(n, g)$ instead of n. Formally,

$$P_{est}(h, g, n) = P_{real}(h, 0.5, E(n, g)).$$

Approximating function $1 - e^{-\lambda}$ by λ, which is reasonable for small values of λ considering the Taylor series (Brannan, 2006), we obtain approximations $P'_{real}(h, g, n)$ and $P'_{est}(h, g, n)$ of the P-values:

$$\begin{aligned} P'_{real}(h,g,n) &= n2^{-m}\sum_{z=0}^{m}\binom{m}{z}P(Y|h,Q_z,g) \\ &= P'_{real}(h-1,g,n) + n2^{-m}\sum_{z=0}^{m}\binom{m}{z}P(X|h,Q_z,g) \qquad (2.5) \\ P'_{real}(0,g,n) &= n2^{-m}\sum_{z=0}^{m}\binom{m}{z}\left(\frac{g}{2}\right)^{z}\left(\frac{1-g}{2}\right)^{m-z} = n2^{-2m} \qquad (2.6) \end{aligned}$$

$$\begin{aligned} P'_{est}(h,g,n) &= P'_{est}(h-1,g,n) + \frac{nH(g)}{2}2^{-m}\sum_{z=0}^{m}\binom{m}{z}P(X|h,Q_z,g=0.5) \qquad (2.7) \\ &= P'_{est}(h-1,g,n) + \frac{nH(g)}{2}2^{-m}\binom{m}{h}\left(\frac{3}{4}\right)^{h}\left(\frac{1}{4}\right)^{m-h}\sum_{z=0}^{m}\binom{m}{z} \\ &= P'_{est}(h-1,g,n) + \frac{nH(g)}{2}\binom{m}{h}3^{h}2^{-2m} \qquad (2.8) \\ P'_{est}(0,g,n) &= n2^{-2m}\cdot\frac{H(g)}{2} \qquad (2.9) \end{aligned}$$

Comparing expressions 2.6 and 2.9 we can see that the estimate $P'_{est}(0, g, n)$ is lower than $P'_{real}(0, g, n)$ by a factor $H(g)/2$. However, this estimate is valid only for low P-values.

Now we look at the exact formulas for the P-values P_{real} and P_{est}, not using the Taylor series approximation:

$$\begin{aligned} P_{real}(h,g,n) = P(Z>0|h,g) &= 1 - 2^{-m}\sum_{z=0}^{m}\binom{m}{z}\exp\left(-nP(Y|h,Q_z,g)\right), \\ P_{est}(h,g,n) = P(Z>0|h,g=0.5) &= 1 - 2^{-m}\sum_{z=0}^{m}\binom{m}{z}\exp\left(-\frac{nH(g)}{2}P(Y|h,Q_z,g=0.5)\right). \end{aligned}$$

For $h = 0$ we obtain expressions:

$$
\begin{aligned}
P_{real}(0,g,n) &= 1-\sum_{z=0}^{m} 2^{-m}\exp\left(-n2^{-m}g^{z}(1-g)^{m-z}\right), \qquad (2.10)\\
P_{est}(0,g,n) &= 1-\sum_{z=0}^{m} 2^{-m}\exp\left(-n2^{-2m}\frac{H(g)}{2}\right) =\\
&= 1-(m+1)2^{-m}\exp\left(-n2^{-2m}\frac{H(g)}{2}\right). \qquad (2.11)
\end{aligned}
$$

For sufficiently large n, the estimates $P_{est}(0,g,n)$ become consistently greater than $P_{real}(0,g,n)$ and therefore conservative. Let us assume that

$$n > \frac{m2^{-m}\ln(2)}{H(g)2^{-m-1}-(1-g)^{m}} = \frac{1.386}{H(g)}m4^{m}+o(m4^{m}). \qquad (2.12)$$

This implies that the whole sum over z in the expression 2.11 is less than the first element of the sum in the expression 2.10 and therefore $P_{real}(0,g,n) \leq P_{est}(0,g,n)$ for any GC-content g. Note, that for n fulfilling inequality 2.12, the approximations 2.6 and 2.9 are not appropriate. In addition, for such big values of n, P-value of the considered query is very close to one and the query becomes uninteresting from the biological point of view. Nonetheless, we have shown that there is a lower bound for database size ensuring that $P_{est}(0,g,n)$ is the conservative estimate of $P_{real}(0,g,n)$. Perhaps a tighter bound on n can be obtained by considering additional elements of the sum in 2.10 or a higher value h.

2.5.2 Simulations

In this section, we test our general idea of using effective database size instead of the real size for P-values estimation, on simulated databases of various sizes and GC-content levels using the algorithm from Section 2.3.

Data shown in Table 2.2 are averages over values for five different randomly generated databases. The P-values in different rows are obtained by distinct methods for two different GC-content values 75% and 90%. For each GC-content, in the first line is the real P-value calculated on database D. In the second line is the estimate of P-value predicted by our compression method using effective database size of D computed as $E(n,g) = H(g)n/2$ and then applying the algorithm on a database generated uniformly. The next two lines contain P'_{real} and P'_{est} calculated according to formulas 2.5 and 2.7, respectively.

	h=0				h=2			
Database size	10^4	10^5	10^6	10^7	10^4	10^5	10^6	10^7
Real (GC 75%)	$9.3 \cdot 10^{-6}$	$9.3 \cdot 10^{-5}$	$9.2 \cdot 10^{-4}$	$8.4 \cdot 10^{-3}$	$8.8 \cdot 10^{-3}$	$7.0 \cdot 10^{-2}$	$3.2 \cdot 10^{-1}$	$7.2 \cdot 10^{-1}$
Predicted (GC 75%)	$8.4 \cdot 10^{-6}$	$8.5 \cdot 10^{-5}$	$8.4 \cdot 10^{-4}$	$8.4 \cdot 10^{-3}$	$8.3 \cdot 10^{-3}$	$8.1 \cdot 10^{-2}$	$5.7 \cdot 10^{-1}$	1.0
P'_{real} (GC 75%)	$9.3 \cdot 10^{-6}$	$9.3 \cdot 10^{-5}$	$9.3 \cdot 10^{-4}$	$9.3 \cdot 10^{-3}$	$4.0 \cdot 10^{-2}$	$4.0 \cdot 10^{-1}$	1.0	1.0
P'_{est} (GC 75%)	$8.4 \cdot 10^{-6}$	$8.4 \cdot 10^{-5}$	$8.4 \cdot 10^{-4}$	$8.4 \cdot 10^{-3}$	$8.3 \cdot 10^{-3}$	$8.3 \cdot 10^{-2}$	$8.3 \cdot 10^{-1}$	1.0
Real (GC 90%)	$9.2 \cdot 10^{-6}$	$8.7 \cdot 10^{-5}$	$6.7 \cdot 10^{-4}$	$3.9 \cdot 10^{-3}$	$6.5 \cdot 10^{-3}$	$3.3 \cdot 10^{-2}$	$1.1 \cdot 10^{-1}$	$2.6 \cdot 10^{-1}$
Predicted (GC 90%)	$6.5 \cdot 10^{-6}$	$6.8 \cdot 10^{-5}$	$6.8 \cdot 10^{-4}$	$6.8 \cdot 10^{-3}$	$6.4 \cdot 10^{-3}$	$6.5 \cdot 10^{-2}$	$4.9 \cdot 10^{-1}$	1.0
P'_{real} (GC 90%)	$9.3 \cdot 10^{-6}$	$9.3 \cdot 10^{-5}$	$9.3 \cdot 10^{-4}$	$9.3 \cdot 10^{-3}$	$1.1 \cdot 10^{-1}$	1.0	1.0	1.0
P'_{est} (GC 90%)	$6.8 \cdot 10^{-6}$	$6.8 \cdot 10^{-5}$	$6.8 \cdot 10^{-4}$	$6.8 \cdot 10^{-3}$	$6.8 \cdot 10^{-3}$	$6.8 \cdot 10^{-2}$	$6.8 \cdot 10^{-1}$	1.0
Simple	$9.3 \cdot 10^{-6}$	$9.3 \cdot 10^{-5}$	$9.3 \cdot 10^{-4}$	$9.3 \cdot 10^{-3}$	$9.2 \cdot 10^{-3}$	$8.8 \cdot 10^{-2}$	$6.0 \cdot 10^{-1}$	1.0
P'_{simple}	$9.3 \cdot 10^{-6}$	$9.3 \cdot 10^{-5}$	$9.3 \cdot 10^{-4}$	$9.3 \cdot 10^{-3}$	$1.2 \cdot 10^{-2}$	$1.2 \cdot 10^{-1}$	1.0	1.0

Table 2.2: P-values for random databases of various lengths n, GC content 75% or 90%, query length $m = 15$, and the query distance $h = 0$ or $h = 2$.

The predictions in the last two lines of the table do not depend on GC-content. The first of them shows the simple P-value estimate that we obtain by calculating P-value for a database with the uniform nucleotide frequencies and with the same size as D. In the last line of the table is the P-value estimate P'_{simple} obtained by formula 2.5 using $g = 0.5$ and the real size of D. The different columns represent various database sizes for two different h values, the distance between database D and the considered query for which P-value is predicted.

For small P-values the real and simple estimates are quite similar, which is expected since in a short database very few m-tuples occur multiple times, even if the composition of the database is skewed. As a result, the compression method gives non-conservative estimates, because it uses a much smaller database size. For small P-values also P'_{real} and P'_{est} approximate the P-values obtained by our algorithm.

For larger P-values, the compression estimates become conservative, and are closer to the true P-value than the simple estimates. In Table 2.2 we can see that compression estimates are better than the simple for databases larger than 10^7 and 10^5 nucleotides for $h = 0$ and $h = 2$, respectively. This is because a database with high GC-content is less likely to contain m-tuples with low GC-content, and thus a larger database size is required to achieve the same P-value. For larger P-values, a difference between approximations P'_{real} and P'_{est} and the P-values obtained by the algorithm gets bigger as the approximations are less accurate.

In this experiment we can see that for databases with skewed GC-content, we can get non-conservative estimates using either simulated databases or the derived formulas. We will return to this topic later, while discussing the real data.

2.6 Real Data

In this section we test our compression method on the real genomic data from human, chimpanzee, and rhesus macaque. We use a piece of genomic sequence on human chromosome 22 and the corresponding sequences from the other two species. We have excluded larger blocks that did not have their counterparts neither in chimpanzee, nor in macaque. The genomic sequences and genome alignments which identifies the corresponding parts of the sequences among the primate sequences were obtained from UCSC genome browser (Kent et al., 2002).

We created a database from the genomic data such that a short block of the human sequence is followed by the corresponding blocks from the other genomes, then another human block follows, etc. This organization of sequences in the database causes that similar sequence blocks occur close to each other, which improves compression by algorithms such as bzip2, which always consider only a block of the whole file.

The real P-values as well as their estimates obtained by the previously described methods are shown in Table 2.6 for databases of various sizes. As we can see, GenCompress often underestimates the real P-value, while performance of bzip2 estimates is only slightly better than the simple estimates. In section 2.5, we have shown that skewed GC-content should not automatically imply lower effective database size since this would lead to underestimation of small P-values. However, compression algorithms use both sequence redundancy and lower sequence entropy in case of locally high or low GC-contents to obtain good compression. This can be the reason, why the P-value estimates based on the size of GenCompress results are non-conservative.

We have attempted to further correct for the underestimation caused by entropy in the following way: We have computed database entropy H' and the effective database size estimated by GenCompress was then multiplied by the correction factor of $2/H'$ that corresponds to ratio $P'_{real}(0,g,n)/P'_{est}(0,g,n)$. By the correction

	h=0				h=2			
Database size	$1.6 \cdot 10^4$	$2.1 \cdot 10^5$	$1.5 \cdot 10^6$	$1.1 \cdot 10^7$	$1.6 \cdot 10^4$	$2.1 \cdot 10^5$	$1.5 \cdot 10^6$	$1.1 \cdot 10^7$
Real	$1.1 \cdot 10^{-5}$	$1.4 \cdot 10^{-4}$	$9.4 \cdot 10^{-4}$	$6.0 \cdot 10^{-3}$	$1.0 \cdot 10^{-2}$	$1.1 \cdot 10^{-1}$	$4.6 \cdot 10^{-1}$	$9.1 \cdot 10^{-1}$
GenCompress	$9.3 \cdot 10^{-6}$	$1.2 \cdot 10^{-4}$	$8.5 \cdot 10^{-4}$	$6.0 \cdot 10^{-3}$	$9.2 \cdot 10^{-3}$	$1.1 \cdot 10^{-1}$	$5.7 \cdot 10^{-1}$	1.0
GC corrected	$1.1 \cdot 10^{-5}$	$1.4 \cdot 10^{-4}$	$9.6 \cdot 10^{-4}$	$6.8 \cdot 10^{-3}$	$1.1 \cdot 10^{-2}$	$1.3 \cdot 10^{-1}$	$6.1 \cdot 10^{-1}$	1.0
bzip2	$1.5 \cdot 10^{-5}$	$1.9 \cdot 10^{-4}$	$1.3 \cdot 10^{-3}$	$9.4 \cdot 10^{-3}$	$1.5 \cdot 10^{-2}$	$1.7 \cdot 10^{-1}$	$7.2 \cdot 10^{-1}$	1.0
Simple	$1.5 \cdot 10^{-5}$	$1.9 \cdot 10^{-4}$	$1.4 \cdot 10^{-3}$	$1.1 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$1.8 \cdot 10^{-1}$	$7.4 \cdot 10^{-1}$	1.0

Table 2.3: P-values for genomic data from human, chimpanzee, and rhesus.

we eliminate the fact that skewed entropy of a database leads to underestimation by effective size of the database.

To estimate the value H' for this experiment, we have computed entropy separately for non-overlapping windows of size 1000 bases to capture different properties of individual genomic regions. We have estimated a Markov chain of second order from each window of the sequence and computed an entropy of this Markov chain, that is, entropy where the probability of each nucleotide is conditioned on the two previous nucleotides to capture local dependencies in DNA sequences. The average entropy H' of the whole database was then computed as an average of entropy values from all windows.

The correction leads to surprisingly good P-value estimates. The estimates were conservative in our experiments (Table 2.6, line GC corrected) what indicates relevance of this approach.

Chapter 3

Segmentation of DNA Sequences

In this chapter, we study the problem of sequence segmentation. We assume that large-scale evolutionary events such as duplications and rearrangements have an influence on genomic sequences as well as small-scale substitutions, insertions, and deletions. We are given a genomic sequence, and our task is to identify non-overlapping segments of the sequence and to partition these segments into classes so that segments within each class are similar to each other, and there are no significant similarities between segments from different classes. Initially, we search for significant local alignments within the sequence. A pair of regions aligned to each other share high level of similarity. It is possible that one of the aligned regions have been created by a duplication event from the other region. Boundaries of an alignment indicate decline in similarity towards ends of the aligned regions. This happens, for example, nearby the position where a copy of some duplicated region have been inserted. Therefore, we will use alignment boundaries to identify boundaries of segments.

Segments found by our algorithm can be used as markers in a wide range of evolutionary studies. This way, one can abstract from particular bases in the markers, and consider only the order of the markers. This idea was introduced by Nadeau and Taylor (1984) to study rearrangements between genomic sequences of mouse and human, and at that time the segments at hand consisted of several millions base pairs. Smaller events occurring within these long segments were ignored in the analysis.

Nowadays, methods for segmentation of shorter and complexly structured genomic regions are also needed. For example, study of genomic regions where mas-

$$\begin{aligned} X&: \quad ABCDEFG \\ Y&: \quad ABCDEB'C'D'FG \\ Z&: \quad ABCDEB'D'FC'G \end{aligned}$$

Figure 3.1: Sequence Z underwent two segmental events, first duplication of segment BCD and later rearrangement of C' during its evolution from sequence X. Considering X as the reference ancestral sequence, X consists of one true atom $ABCDEFG$, Y consists of true atoms A, BCD, $B'C'D'$, E, and FG, with BCD and $B'C'D'$ belonging to the same class. In sequence Z each symbol is a separate true atom with atoms denoted by the same letter (either normal or primed) belonging to the same class.

sive duplication processes have taken place (Zhang, 2003) possibly identifies the evolutionary events which happened in the regions and explains functionality of the segments in the contemporary species. These complex regions usually contain relatively short homologous segments (hundreds to thousands of base pairs), which we call *atoms* in the following text.

If we would know the exact history of evolutionary events which happened in a region, we would be able to identify *true atoms* and separate them into classes such that each class originate from one segment of the shared ancestral sequence. For example, consider contemporary sequence Z and its ancestors X and Y in Figure 3.1. During evolution, two segmental events happened: First, segment BCD of X have been duplicated and its copy $B'C'D'$ have been inserted between E and F to create sequence Y. Later, segment C' have been rearranged and moved inbetween segments F and G in sequence Z. In this situation, considering X as the reference ancestral sequence sequence, Z consists of ten true atoms with atoms denoted by the same letter (either normal or primed) belonging to the same class. The *true atomization* is not known for real sequences but we assume its existence. Here we want to use the information about sequence similarity to formulate the optimization problem, which solution allows us to predict the true atomization.

Problem description. The input for our problem is a group of evolutionary related DNA sequences and a collection of high-scoring local alignments among homologous regions of the sequences and also between parts of the same sequence.

We want to tile the input sequences by segments of two different kinds: atoms and waste regions. Each atom has length at least L (where L is a given minimal length parameter); a waste region may be of any size. Each base of the input sequences belong to exactly one atom or waste region. Atoms are the segments, which can be used as markers for subsequent analysis. Waste regions are the segments, which we cannot reliably include in any atom due to imperfections in the input alignments. Therefore we want to have high coverage of the input sequence by atoms and low number of atoms to avoid unnecessary segmentation. In addition, we divide atoms into classes so that every two atoms in the same atomic class share high sequence similarity across their entire length, and two atoms of different classes do not appear to be similar at the chosen sequence similarity threshold.

Related work. In comparative genomics, identification of homologous segments within the sequences of interest is the first step for studying rearrangement (Blanchette et al., 1999; Bourque et al., 2004; Kováč et al., 2011) and duplication (Jaitly et al., 2002; Zhang et al., 2003; Vinar et al., 2010; Holloway et al., 2012) histories. The segments are typically constructed from annotated genes in the sequences, and we can either use individual genes directly or combine them to longer regions called synteny blocks (Choi et al., 2007; Hachiya et al., 2009). Lengths of such blocks usually exceed 10^5 base pairs. Occasionally, synteny blocks are build without gene annotation, only according to sequence similarity (Pevzner and Tesler, 2003). Synteny blocks may contain short segmental rearrangements and duplications within them. Our atoms can be seen as shorter sized synteny blocks, and more detailed segmentation allows us to study smaller-scaled evolutionary events that happened in a given region.

With the availability of the next generation sequencing methods, the number of the accessible genomes increases quickly (almost 2500 species genomes are finished and published[1]). Therefore, sequences of closely related species can be analyzed. In such genomes, it is often possible to detect homology not only in genes which translate to proteins, but also in many non-coding sequences. Using segmentation, which is not based solely on protein coding genes, it may be possible to study recent evolutionary events in more detail.

An example of highly structured genomic regions, which are interesting also from the biological point of view, are gene clusters. Gene cluster is a region where one gene or some portion of a gene was copied repeatedly to the position nearby. Not all the duplicated segments are translated to proteins, and the cause of cre-

[1]`http://www.genomesonline.org`

ation such gene clusters as well as potential regulation functions of the duplicated segments are not yet fully understood. One particular example is UGT1A (UDP-glucuronosyltransferase) gene cluster consisting of at least 13 duplicated copies of one short exon or its parts (Bellemare et al., 2010). We will see performance of our algorithms on this cluster at the end of this chapter.

There is some similarity between finding an atomization and finding multiple sequence alignment of the input sequences. In multiple sequence alignment, the goal is to identify columns of homologous positions, usually one position or a gap from each input sequence in each column. Most of the software tools for multiple sequence alignment require input sequences without duplications (Blanchette et al., 2004), or both duplications and rearrangements (Ogurtsov et al., 2002; Höhl et al., 2002; Brudno et al., 2003a; Bray and Pachter, 2004; Blanchette et al., 2004), and therefore are not appropriate for solving the atomization problem. There are several extensions of the previous tools, which deal also with rearrangement within the input sequences: SLAGAN, Mulan, and progressiveMauve by Brudno et al. (2003b), Ovcharenko et al. (2005), Darling et al. (2010), respectively. Algorithm SLAGAN also identifies occurrence of the segmental duplications.

The goal of these algorithms is to identify synteny blocks within the compared sequences and create multiple alignments of the corresponding blocks. Therefore, the algorithms deal with segmental insertions and deletions[2] (and also duplications in the case of Mulan and progressiveMauve) in a different way than we consider to be fruitful for further analyses of duplication histories. In particular, segmental indels are often inside bigger synteny blocks. Hence, we prefer to split such synteny blocks into several consecutive atoms with the possible insertions separated from the other parts of the synteny block. Another issue is, that Mulan and progressiveMauve create a phylogenetic tree of the input data according to the gene content of the input sequences. The algorithms use the tree during heuristic construction of synteny blocks and potential errors in gene annotations influence output of both algorithms.

In this chapter, we follow on work by Brejová et al. (2011) who informally introduce the atomization problem and give an iterative algorithm. However, they do not provide any formal definition of the problem and the resulting segmentation is not guaranteed to satisfy any clear conditions other than that each atom has length at

[2]called indels together

least W. In this work, we provide a formal definition of the problem. This allows us to study the problem more systematically from both theoretical and practical aspects. In particular, in Section 3.1, we formally define sequence segmentation as an optimization problem, and in Section 3.2, we show that under certain circumstances, the optimal and the true atomizations are identical. We prove that the optimization problem is NP-hard in Section 3.3. Two heuristic algorithms, which address the segmentation problem, are introduced in Section 3.4. These algorithms, similarly to the algorithm by Brejová et al. (2011), start with an initial set of atoms and iteratively remap the segments, change atomic boundaries, and split the atoms, until the final atomization is obtained. The most significant difference between our new algorithms from Section 3.4 and the previous approach is that we introduce waste regions of non-zero lengths at the locations where ends of local alignments occur at several nearby positions. In contrast, the algorithm by Brejová et al. (2011) uses only breakpoints of length zero separating adjacent atoms. In difficult regions this leads to atoms that are artifacts of the process, whereas our approach can mark those regions as waste. In Section 3.5 we compare the accuracy of our two algorithms with the previous algorithm on both simulated and real sequences.

3.1 Problem Definition

The goal of this section is to define the sequence segmentation problem as a formal computational problem that can be further analyzed and studied by traditional methods of computer science. We first define notation used in this work and introduce the concept of atomization as formalized sequence segmentation. Then we formulate the search for the true segmentation as the search for the optimal atomization given a well-chosen cost function.

3.1.1 Notation

As an input data for atomization we have DNA sequence $S = s_1 s_2 \dots s_n$ and a set α containing significant local alignments between pairs of homologous regions within the sequence. When we have more than one sequence as an input, we concatenate the input sequences $S_1, \dots, S_m$ and create one long sequence $S = S_1 \cdot S_2 \dots S_m$. The alignments will be changed accordingly, so that they refer to the correct coordinates

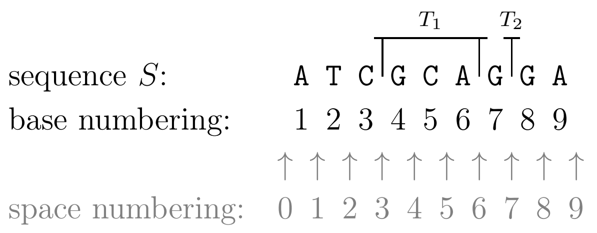


Figure 3.2: Sequence S and its subsequences T_1 and T_2 from Example.

```
sequence T1:  A T C - - G C A G G A
sequence T2:  A T C A A G T A G - A
```

Figure 3.3: An alignment of T_1 and T_2.

within sequence S.

A subsequence of S is a continuous block of bases $s_k s_{k+1} \ldots s_{k+l}$. As we will later need to refer to subsequences of length 0 (empty subsequences at specific positions within a genomic sequence), we define subsequences in a way that will allow this.

Definition 1. *We label the space between two consecutive bases s_i and s_{i+1} of sequence S by number i. The space to the left of base s_1 gets label zero and the space to the right of base s_n gets label n. Then* $\mathrm{sub}(i, j)$ *is the* subsequence *of S which starts in the space labeled by i and ends in the space labeled by j, where $0 \leq i \leq j \leq n$.*

Example 1. *Let us consider sequence $S =$"*`ATCGCAGGA`*". The numbering of bases of sequence S and the numbering of spaces between the bases is shown in Figure 3.2. Now we can refer to different subsequences of S, for example $T_1 = \mathrm{sub}(3, 6) =$"*`GCA`*" and $T_2 = \mathrm{sub}(7, 7) =$"". Subsequence T_2 is a space between two consecutive "*`G`*"s of sequence S.*

Let us recall that a pairwise sequence alignment is a way of arranging two biological sequences by adding some spaces (indels) between bases of one or the other sequence (e.g., Figure 3.3). To build an atomization of input data, we use information about corresponding base pairs in aligned subsequences of S, but we do not need knowledge about particular bases. With this in mind, we formulate a more abstract alignment definition for the purpose of this chapter. We store an alignment as a relation between positions of two sequences.

Definition 2. *Let* $\mathrm{sub}(i, i+k)$ *and* $\mathrm{sub}(j, j+l)$ *be subsequences of* S*, which are aligned by pairwise sequence alignment* a*. Let* $r \subseteq \{i+1, \ldots, i+k\} \times \{j+1, \ldots, j+l\}$ *be a binary relation that capture pairs of base indices for bases aligned to each other in* a*: pair of indices* (p, q) *is in relation* r*, if and only if base* s_p *is aligned to base* s_q *in pairwise sequence alignment* a*. Then we denote the correspondence between the base indices by* $\mathrm{align}(i, i+k, j, j+l, r)$ *and call it* an alignment.

We often refer to an alignment $\mathrm{align}(i, i+k, j, j+l, r)$ to emphasize a relationship of the parts of sequence S instead of discussing exact definition of relation r. In that case, we simplify notation by omitting r and use $\mathrm{align}(i, i+k, j, j+l)$ instead.

Definition 3. *Let* $T_1 = \mathrm{sub}(i, i+k)$ *and* $T_2 = \mathrm{sub}(j, j+l)$ *be subsequences of* S*. Suppose that* T_1 *starts earlier than* T_2 *or at the same position* ($i \leq j$)*. If* $i+k > j$*, we say that* T_1 *and* T_2 overlap *and their overlap is* $\mathrm{sub}(j, \min(i+k, j+l))$*. If* $i+k \geq j+l$*, we say that* T_1 covers T_2*.*

Let $U_1 = \mathrm{sub}(i, i+k)$*,* $U_2 = \mathrm{sub}(j, j+l)$*, and* U_3 *be subsequences of* S*. Suppose that alignment* $\mathrm{align}(i, i+k, j, j+l)$ *between* U_1 *and* U_2 *exists. If* U_1 *or* U_2 *overlaps* U_3*, we say that the* alignment overlaps U_3*. If* U_1 *or* U_2 *covers* U_3*, we say that the* alignment covers U_3*.*

Definition 4. *Let* $\mathrm{align}(i, i+k, j, j+l, r)$ *be an alignment. Let* p *and* q *are base indices such that* $i < p \leq i+k$*,* $j < q \leq j+l$*. We say that* s_p maps *to base* s_q *through alignment* $\mathrm{align}(i, i+k, j, j+l, r)$ *if a pair* (p, q) *belongs to relation* r*.*

Definition 5. *Let* $T_1 = \mathrm{sub}(i, i+k)$*,* $T_2 = \mathrm{sub}(j, j+l)$ *and* t *be an alignment of* T_1 *and* T_2*. Suppose* $U = \mathrm{sub}(p, p+m)$*, such that* T_1 *covers* U*.* Mapping of sequence U *through alignment* t *is the shortest subsequence of* T_2 *that contains mapping of every base from* U *(for which such mapping exists). We denote the mapping by* $\mathrm{map}(t, U)$*.*

3.1.2 Optimal Atomization

In this section, we introduce a formal definition of an atomization, the central concept of the work. Atomization describes segmentation of biological sequences as a computational problem that we will analyze further in this chapter. We define atoms as building blocks of any atomization, and afterwards we put restrictions on a set of atoms, which form an atomization.

Definition 6. *We say that $A = \text{sub}(i, i+k)$ is an* atom*, if both following conditions hold:*

1) *The length of A is at least L (where L is a given minimal length parameter).*
2) *Each alignment that overlaps with A also covers A.*

Suppose that sequence S contains several homologous copies of a region d. In an atomization, we want to identify every occurrence of d either as a separate atom or as a part of a bigger atom, which consists of several homologous segments ordered in the same way around every occurrence of d. However, we wish to avoid the situation when one copy of d forms an atom by itself, and another copy is only a small part of a longer atom. We avoid this problem by the following definition.

Definition 7. *We say that a set of atoms $\mathcal{A}$ is an* atomization of input sequence *S, if both following conditions hold:*

1) *No two atoms from set $\mathcal{A}$ overlap.*
2) *Each alignment from α that covers some atom $A \in \mathcal{A}$, maps the atom A to a region of S that overlaps with exactly one another atom.*

Definition 6 implies that no atom contains boundaries of any alignment. According to Definition 7, an atom A of some atomization $\mathcal{A}$ can be either aligned to a smaller part within another atom, or to a region r, which contains not only an atom B, but also some surrounding bases. The second case is possible if the bases in r surrounding B are not covered by any other atom of $\mathcal{A}$. We call waste regions the parts of sequence S that are not covered by atoms.

Definition 8. *Let $A_1, A_2, \ldots, A_p$ be the atoms of an atomization $\mathcal{A}$ ordered by their position in S. The* waste region *between atoms A_k and A_{k+1} is the subsequence $sub(i, j)$, where i is the space to the right from the last base of atom A_k and j is the space to the left from the first base of atom A_{k+1}. We also call a* waste region *the subsequence $sub(0, i)$, where i is the space to the left from the first base of atom A_1 and the subsequence $sub(i, \text{length}(S))$, where i is the space to the right from the last base of atom A_p.*

Waste regions allows us to specify an atomization in an alternative way. Depending on a situation, we refer to an atomization either by its atoms, or by the waste regions between the atoms. Alignments, which cover the same region of S usually do not have boundaries at identical positions, but the boundaries are often spread around the boundary of the true atom. The reason is a difficult detection of homology boundaries in local alignment search. As the true boundary has an uncertain position, we have chosen the conservative approach and consider the region containing several alignment boundaries close to each other to be a waste region.

To partition atoms to classes, we will represent alignments in the form of a graph. The *alignment graph* of atomization $\mathcal{A}$ is the graph in which each vertex corresponds to an atom of $\mathcal{A}$ and two vertices A_1 and A_2 are connected by an edge if some alignment maps A_1 to a region overlapping A_2 and vice versa. Then each atomic class of $\mathcal{A}$ is represented by a connected component of the alignment graph.

There are many valid atomizations of sequence S. To choose the best atomization, we introduce a cost function that scores the various atomizations such that the lower score means the better atomization. As we prefer to have a high coverage of input sequence S by atoms, we expect a cost function to penalize an atomization for bases that are located in waste regions. We also prefer an atomization with lower number of atoms to avoid unnecessary fragmentation; therefore we expect the cost function to penalize an atomization for the number of atoms. For simplicity, our cost function does not take atom distribution to classes into account.

Definition 9. *Let $\mathcal{A}$ be an atomization of sequence S. Let*

$$u = \mathrm{length}(S) - \sum_{A \in \mathcal{A}} \mathrm{length}(A)$$

be the number of bases in waste regions. We define the cost of atomization $\mathcal{A}$ as:

$$\mathrm{c}_{(a,w)}(\mathcal{A}) = |\mathcal{A}| \cdot a + u \cdot w,$$

where $a, w > 0$ are parameters determining the cost of each atom and the cost of each base in a waste region, respectively.

We can now approach the segmentation problem as an optimization problem of searching for the minimum cost atomization of S according to some cost function $\mathrm{c}_{(a,w)}(\cdot)$.

3.1.3 Influence of Cost Function

In the following text, we address influence of the chosen cost function on the optimal atomization. We also set the parameters of the cost function, which we will use in the final formulation of the atomization problem.

We will illustrate the influence of cost function on the following example. Let L be the minimum length of any atom. Let sequence S be a concatenation of $n+2$ shorter sequences $S_1, S_2, \dots, S_{n+2}$, where length$(S_1) = 2L$, length$(S_2) = L$, and length$(S_3) =$ length$(S_4) = \cdots =$ length$(S_{n+2}) = 3L$. Suppose that sequences $S_3, S_4, \dots S_{n+2}$ are identical to each other, sequence S_1 is identical to the last $2L$ bases of the long sequences, and sequence S_2 is identical to the first L bases of the long sequences. Suppose that there is an alignment between S_1 and the last $2L$ bases of sequence S_3, an alignment between sequence S_2 and the first L bases of sequence S_3 and also an alignment for each pair of sequences S_i, S_j, where $3 \le i < j \le n+2$, that maps a base on position p in sequence S_i to the base on position p in sequence S_j.

Sequence set $\{S_1, S_2, \dots, S_{n+2}\}$ is not an atomization of S. In particular, S_3 overlaps but is not covered by alignments between S_1 and S_3 and between S_2 and S_3. Let us consider two atomizations of S (Figure 3.4):

1) Sequences S_1 and S_2 remain as atoms. Every other sequence S_i, where $3 \le i \le n+2$, splits into two atoms, the first containing bases from 1 to L and the second consisting of bases from $L+1$ to $2L$. We call this atomization $\mathcal{A}$.
2) Two waste regions are created, the first one containing whole sequence S_2 and the second one bases from 1 to L of sequence S_3. Every sequence from $\{S_1, S_4, S_5, \dots, S_{n+2}\}$ becomes an atom, and the last $2L$ bases of S_3 also form an atom. We call this atomization $\mathcal{B}$.

Intuitively, atomization $\mathcal{A}$ better corresponds to homology between atoms, although the input is unusual in the sense that it does not contain alignments between S_1 or S_2 and S_i for $i > 3$. Nonetheless, we are interested in cost functions $\mathrm{c}_{(a,w)}$ that prefer atomization $\mathcal{A}$ to atomization $\mathcal{B}$. We set $w = 1$ and look for values of parameter a, for which $\mathrm{c}_{(a,1)}(\mathcal{A}) < \mathrm{c}_{(a,1)}(\mathcal{B})$.

As an extreme case, we could consider value $a = 0$. Any atomization that contains only waste regions of zero length then costs zero, and in particular $\mathrm{c}_{(0,1)}(\mathcal{A}) = 0$. However, other less desirable atomizations have the same cost. Consider for example atomization $\mathcal{C}$ of sequence S, that splits every sequence S_i, where $3 \le i \le n+2$ to three atoms of length L, splits sequence S_1 to two atoms of length L, and sequence S_2 becomes an atom of length L bases. We want the cost of atomization $\mathcal{C}$ to be higher than the cost of atomization $\mathcal{A}$, as $\mathcal{C}$ contains unnecessarily many atoms, but

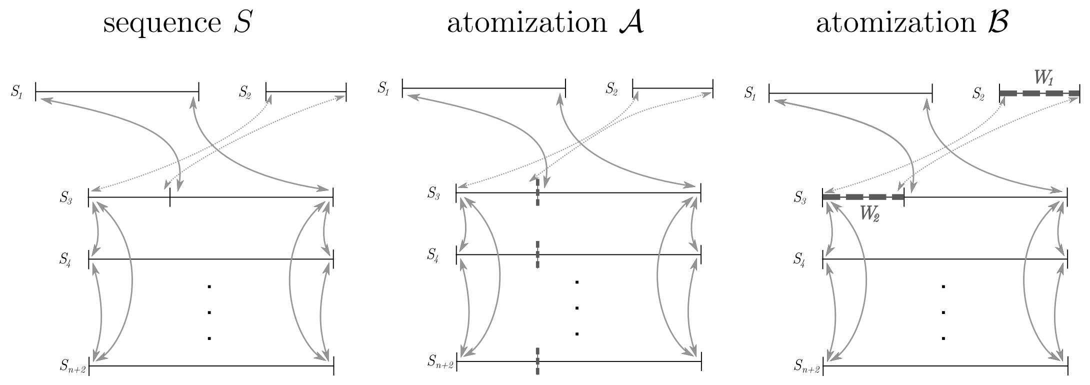


Figure 3.4: Sequence S and its two possible atomizations. a) Sequence $S = S_1.S_2 \dots S_{n+2}$ split to the short sequences, alignments between the sequences and their parts shown by gray arrows. b) In atomization $\mathcal{A}$, every sequence S_i, where $3 \leq i \leq n+2$, is split to two atoms and one waste region of length zero. The waste regions between atoms are shown by dashed lines. c) Atomization $\mathcal{B}$, waste regions W_1 and W_2 are created and every sequence S_i, except S_2 contains one atom.

cost function $\mathrm{c}_{(0,1)}(\cdot)$ prices both atomizations equally. Therefore, the parameter choice $a = 0, w = 1$ is not appropriate.

In general, we have that $\mathrm{c}_{(a,1)}(\mathcal{A}) = 2a \cdot (n+1)$ and $\mathrm{c}_{(a,1)}(\mathcal{B}) = a \cdot (n+1) + 2L$, and therefore atomization $\mathcal{A}$ costs less than $\mathcal{B}$ when $a < 2L/(n+1)$. Ideally, we would choose the value of a, which is positive, but sufficiently small regardless of values of n and L. This requirement is fulfilled, for example, by value $a = 1/\mathrm{length}(S)$. In our example, this means, $a = 1/[(n+1) \cdot 3L]$. More generally, this value guarantees lower cost for atomizations that similarly to $\mathcal{A}$ distribute information about shorter atoms to longer sequences which partially align to the shorter atoms.

From now on, we therefore use cost function $\mathrm{c}_{(\epsilon,1)}$, where $\epsilon = 1/\mathrm{length}(S)$, to measure the quality of an atomization. Thus we arrive at the final optimization version of the atomization problem:

Atomization problem, optimization version:	
Instance:	Sequence S, set of alignments α among subsequences of S, minimum atom length L.
Cost function:	$\mathrm{c}_{(\epsilon,1)}$, where $\epsilon = 1/\mathrm{length}(S)$.
Problem:	Determine atomization $\mathcal{O}$ such that cost $\mathrm{c}_{(\epsilon,1)}(\mathcal{O})$ is minimal among costs of all feasible atomizations of the instance.

3.2 Correctness of the Optimal Atomization

In this section we prove that under certain circumstances the optimal atomization is identical to the true atomization. This is in general difficult, because the true atomization is defined only with respect to a particular evolutionary history, and a given sequence can arise by various histories. Here, we define a simple model of atomization without wasted bases which consist of a sequence, set of alignments and an atomization satisfying several constraints. Given these constraints, it is reasonable to assume that the atomization $\mathcal{T}$ is the true atomization.

We say that an atomization $\mathcal{A}$ is an *atomization without wasted bases* if the following conditions are fulfilled for each waste region W (*o1*) and each atomic class β (*cl1* – *cl3*) of atomization $\mathcal{A}$:

o1: Waste region W has zero length.

cl1: Each alignment a from α which covers atom $B \in \beta$ aligns atom B only with another atom of class β. Ends of the two atoms are properly aligned to each other – gaps are not allowed at the first and the last position of any aligned atom.

cl2: Atomic class β contains at least one atom B which has an alignment boundary at its left end. We call such atoms *starting atoms*. The alignment could be an alignment covering B and starting at its first base or it could be covering atom left of B and ending just before start of B. Similarly, atomic class β contains at least one atom which has an alignment boundary at its right end. We call these atoms *ending atoms*.

cl3: Atoms of β form a connected component in the alignment graph of $\mathcal{A}$ and no other atom from a different class belongs to the component.

We expect that the realistic true atomization fulfill also other requirements, for example, that only a pair of sequences which are similar to each other along their whole lengths can be aligned by any alignment. However, in the proof of the following theorem, the conditions defined for atomization without wasted bases are sufficient.

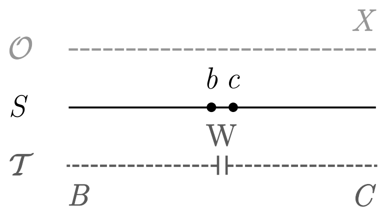


Figure 3.5: Localization of atoms B, C, and waste region W belonging to $\mathcal{T}$ and atom X of $\mathcal{O}$ on the same region of the input sequence S.

Theorem 3.2.1. *Let (S, α) be input atomization data, and $\mathcal{T}$ be the true atomization of the input data satisfying the conditions of an atomization without wasted bases. Then $\mathcal{T}$ and the optimal atomization $\mathcal{O}$ for input (S, α) are identical at the level of atoms and also at the level of atomic classes.*

Proof. The atomization $\mathcal{T}$ contains no bases within its waste regions. According to the cost function $c_{(\epsilon,1)}(\cdot)$, every atomization with some bases in waste regions costs more than $\mathcal{T}$. Therefore $\mathcal{O}$ contains no bases in its waste regions.

We organize this proof into two parts: We show first, that every waste region of $\mathcal{T}$ has its counterpart (a waste region at the same position) in $\mathcal{O}$. This implies that $\mathrm{c}_{(\epsilon,1)}(\mathcal{T}) = \mathrm{c}_{(\epsilon,1)}(\mathcal{O})$, and $\mathcal{T}$ and $\mathcal{O}$ are equal at the atomic level. In the second part we discuss atomic classes and prove that classes of $\mathcal{O}$ are identical to classes of $\mathcal{T}$.

1st part: We prove by contradiction that each waste region of $\mathcal{T}$ has its counterpart in $\mathcal{O}$. This implies that atoms of the two atomizations are identical.

Assume to the contrary that $\mathcal{T}$ contains waste region W which does not have a counterpart waste region in $\mathcal{O}$. Waste region W separates two consecutive atoms B and C of $\mathcal{T}$. We denote b to be the last base of B and c to be the first base of C. As W is not a waste region in $\mathcal{O}$, bases b and c are consecutive bases of the same atom X in this atomization. The situation can be seen in Figure 3.5. Atom B belongs to a class in $\mathcal{T}$ which we denote β. This class fulfills assumptions *cl1* to *cl3*, as $\mathcal{T}$ is the atomization without waste bases.

No input alignment has its left or right boundary between b and c, because otherwise each atomization of S would contain a waste region covering the space between bases b and c. Therefore, B is not an ending atom of class β according to the assumption *cl3*. However, β contains at least one ending atom, we arbitrarily

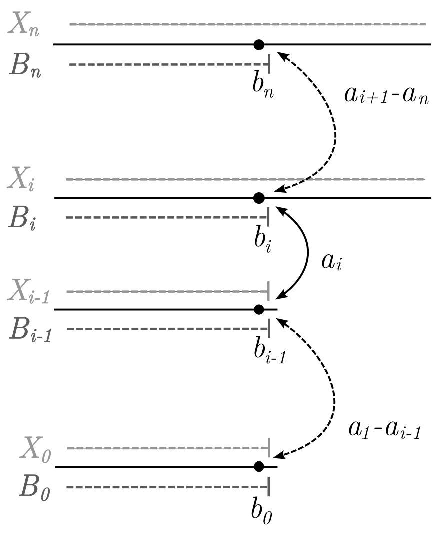


Figure 3.6: Alignment sequence $a_1, a_2, \ldots, a_n$ mapping the last bases of atoms from β. Atoms $X_0, X_1, \ldots, X_n$ of $\mathcal{O}$ at these regions are also drawn.

choose one of them and denote it by B_0. We also denote the last base of B_0 by b_0. Each atom of β can be mapped by a sequence of input alignments to any other atom of the class (*cl3*), and each of the alignments maps the first and the last base of one atom to the first and the last base of another atom in β (*cl1*). Therefore, there is the sequence of alignments a_1, a_2,. . . ,a_n from α, which maps base b_0 to base b. As b_0 is the last base of B_0, which is an ending atom of β, at least one alignment exists, which ends or starts right after b_0. Therefore, atomization $\mathcal{O}$ also contains a waste region W' of length zero located right after b_0.

Hence, base b_0 is the last base of an atom $X_0 \in \mathcal{O}$. As sequence of alignments a_1, a_2,. . . ,a_n maps base b_0 to base b, it also has to align optimal atoms X_0 and X to each other through a sequence of atoms X_1,. . . ,X_{n-1} on the way. Alignment a_i maps atoms X_{i-1} and X_i to each other, given that $X_n = X$. We denote $b_i = map(a_i, b_{i-1})$, where $b_n = b$. Each pair of bases b_i, b_{i+1} maps to each other. The situation can be seen in Figure 3.6. The alignment sequence a_1,. . . ,a_n aligns two atoms X_0 and X_n to each other so that the last base of X_0 aligns to an internal base of X_n. Hence, there has to be an alignment a_i in the alignment sequence, which aligns two consecutive atoms X_{i-1} and X_i so that the last base of X_{i-1} (b_{i-1}) aligns to an internal base of X_i (b_i).

The alignment a_i has to contain at least one aligned base pair following the pair

b_{i-1}, b_i, as a_i covers X_i and each alignment should end with a pair of properly aligned bases (*cl1*). As the true atomization $\mathcal{T}$ contains only zero length waste regions (*o1*) and the first and the last base of each atom in $\mathcal{T}$ should align properly (*cl1*), the bases to the right from b_i and b_{i-1} (let us call them c_i and c_{i-1}, respectively) have to align to each other in alignment a_i. Note that atomization $\mathcal{O}$ also contains only zero length waste regions. Therefore some atom of $\mathcal{O}$ has to start on base c_{i-1}. We denote the atom by Y. Consider now mapping of atom X_i through alignment a_i. It will overlap both Y and X_{i-1}, which contradicts Definition 7. This leads us to the conclusion that for each waste region W from $\mathcal{T}$ a waste region W' should exist in $\mathcal{O}$ covering the same location as W. As the cost of $\mathcal{O}$ does not exceed cost of $\mathcal{T}$, $\mathcal{O}$ does not contain any additional waste regions not included in $\mathcal{T}$.

2nd part: We have shown that waste regions of $\mathcal{T}$ and $\mathcal{O}$ are identical. This implies that atoms of the two atomizations are also identical. Here we prove that atomic classes of the two atomizations are identical as well. Note that atomic classes of $\mathcal{T}$ can be any partition of atoms of $\mathcal{T}$ to classes satisfying conditions of atomization without wasted bases. In contrast, classes of $\mathcal{O}$ are constructed as connected components of the alignment graph (see Section 3.1.2).

We distinguish two cases: singleton classes and classes containing more than one atom. In the first case, no alignment exists that would cover an atom of a singleton class. For that reason, an atom belonging to a singleton class of $\mathcal{T}$ lasts in the singleton classes also in $\mathcal{O}$. In the second case, consider an arbitrary atomic class β from the true atomization with more than one atom. We choose an atom B of class β and denote β_{opt} the atomic class of $\mathcal{O}$ containing B.

Now we prove that if β and β_{opt} contains the same atom B then the classes are identical. First, β_{opt} cannot contain an atom that does not belong to β. According to *cl1* no the input alignment can align an atom from β to an atom that does not belong to β. For this reason, optimal class β_{opt} may contain only atoms of β and no atoms from a different class of $\mathcal{T}$. Second, each atom from β also belongs also to β_{opt}. According to *cl3* for each atom $C \in \beta$, $C \neq B$ a sequence of alignments exists that maps C to B possibly through several intermediate atoms. Hence, C has to belong to the same class of $\mathcal{O}$ as B. We have shown that each atom of β belongs to β_{opt} and also that no one other atom can belong to this class from $\mathcal{O}$.

Therefore, the optimal and the original atomization are identical at the level of atoms and also at the level of atomic classes. □

3.3 Computational Complexity of the Atomization Problem

In this section, we will prove NP-hardness of the atomization problem. We will consider a decision version of the atomization problem (APD), in which we are given a threshold B, and we ask if there is an atomization with cost at most B. It is easy to see that APD is in NP. For an instance of APD, we can nondeterministically guess a set of atoms and calculate its cost as well as verify the requirements of Definition 7 in polynomial time. We will prove NP-hardness of this problem by a reduction from the classical one-in-three 3SAT problem (Garey and Johnson, 1979):

Atomization problem, decision version (APD):	
Instance:	Sequence S, set of alignments α among subsequences of S, minimum atom length L, and threshold B.
Problem:	Is there atomization $\mathcal{A}$ with $c_{(\epsilon,1)}(\mathcal{A}) \leq B$?

One-in-three 3SAT problem:	
Instance:	Set U of variables, collection C of clauses over U such that each clause $c \in C$ has $\lvert c \rvert = 3$.
Problem:	Is there a truth assignment for U such that each clause from C contains exactly one true literal?

First, we show a method for constructing an instance of APD for any instance of one-in-three 3SAT problem and then we prove that the atomization of cost at most B exists for the APD instance if, and only if, the corresponding instance of one-in-three 3SAT is satisfiable.

3.3.1 Construction of APD Instance for One-in-three 3SAT Formula

We start the proof by showing a method for constructing an instance of APD for any instance of one-in-three 3SAT problem. Consider a one-in-three 3SAT instance

(U, C); the goal is to create the corresponding APD instance (S, α, B). During construction of the APD instance, we create many short sequences and alignments between these sequences or their parts. To get APD instance sequence, we concatenate all the short sequences to obtain one input sequence S and adapt the alignments so that they refer to the subsequences of the concatenated sequence that correspond to the short sequences, which were aligned to each other. The adapted alignments form alignment set α. Length threshold L can be an arbitrary constant greater than two.

Gadget description. The basic building block of the constructed APD instance is a gadget consisting of $k+3$ sequences $S_0, S_1, \ldots, S_{k+2}$, all of the same length m. We call sequence S_0 the *interface*, as it is the sole part of the gadget, which will be aligned to sequences outside this gadget; sequences $S_1, \ldots, S_{k+2}$ align only with other sequences within the gadget. In particular, for every i and j such that $0 \leq i < j \leq k+2$, we create alignment $a_{i,j}$. This alignment maps the first and the last base of S_i to the first and the last base of S_j, respectively. It also maps the r-th base of sequence S_i to the $(r+1)$-st base of sequence S_j, for $2 \leq r \leq m-2$. The second base of sequence S_i and the $(m-1)$-st base of sequence S_j remain unaligned in this alignment.

The goal of such a gadget is to ensure that the interface sequence S_0 contains exactly one atom; otherwise the cost of the segmentation will be higher than B. This follows from the following lemma if we set $k = \lceil B \rceil + 1$ (the value of B will be determined later).

Lemma 3.3.1. *If the number of atoms in sequence S_0 is different from one, the gadget $S_0, S_1, \ldots, S_{k+2}$ contains at least k bases in waste regions.*

Proof. If sequence S_0 would not contain any atom, then all sequences of the gadget would have to be covered only by waste regions as well, which rises the number of wasted bases above k.

Let us assume now that S_0 contains at least two different atoms, but the number of bases in waste regions is at most $k-1$. This implies that at least three sequences S_i, S_j, and S_l for $1 \leq i < j < l \leq k+2$ do not contain any wasted bases. However, each of these sequences aligns to S_0, and since S_0 contains at least two atoms, each of these sequences needs to be split into at least two atoms due to Definition 7. As the sequences S_i, S_j, and S_l do not contain any wasted bases, their atoms have to be separated by waste regions of length zero. We denote such a waste region of S_i by W_i. Let us assume that W_i splits sequence S_i between bases x and $x+1$ into two atoms. Bases at positions $x+1$ and $x+2$ in S_j align to the bases adjacent to waste W_i through alignment $a_{i,j}$, and thus they cannot occur in the same atom in S_j. This means that there has to be a waste region between bases $x+1$ and $x+2$ in S_j, we denote it W_j. By the same reasoning, sequence S_l has to contain waste region W_l between bases $x+1$ and $x+2$ due to alignment $a_{i,l}$ and waste region W_l' between bases $x+2$ and $x+3$ due to alignment $a_{j,l}$. In this way, we get two waste regions W_l and W_l' within distance 1. The region between W_l and W_l' cannot be an atom, as each atom needs to have a length at least $L > 2$. However, if the base $x+2$ of S_l belongs to a waste region, we get a contradiction. Therefore, the set $S_1, S_2, \dots, S_{k+2}$ contains at least k sequences containing at least one wasted base each, and the total number of wasted bases in the gadget is at least k. □

APD instance construction. For each variable $y \in U$ we create two sequences X_y and $X_{\neg y}$, each of length L. We also create a gadget $\mathcal{U}_y$ for each variable y, with each sequence of the gadget having length $2L$. Finally, for each clause $c_i \in C$ we create a gadget $\mathcal{C}_i$ containing sequences of length $3L$. In addition to the alignments within gadgets, we add also alignments between sequences X_y and parts of interfaces of gadgets $\mathcal{U}_y$ and $\mathcal{C}_i$. We say that an alignment a is *gapless*, if it aligns regions T_1 and T_2 of the same length and maps the r-th base of T_1 to the r-th base of T_2. In particular, we will align sequence X_y by a gapless alignment to the first half of the interface sequence of $\mathcal{U}_y$ and we will align sequence $X_{\neg y}$ to the second half of this interface. For a clause $c_i = l_1 \vee l_2 \vee l_3$ we align X_{l_1} to the first third of the interface for $\mathcal{C}_i$ and similarly align X_{l_2} to the second third and X_{l_3} to the last third of the interface. Finally, we set $B = 2L(|U| + |C|) + 1/2$.

Clearly the construction of the APD instance for a given instance (U, C) of one-in-three 3SAT problem is polynomial in terms of $|U| + |C|$.

Example 2. *Consider boolean formula* $(a \vee \neg b \vee c) \wedge (a \vee \neg c \vee d)$. *An instance of atomization problem consists of* $6(k+3)+8$ *sequences, where* $k = \lceil B \rceil + 1$ *and* $B = 12L + 1/2$. *Eight of the sequences have length L and each of them represents particular literal. Another* $2(k+3)$ *sequences have length* $3L$ *and they are grouped*

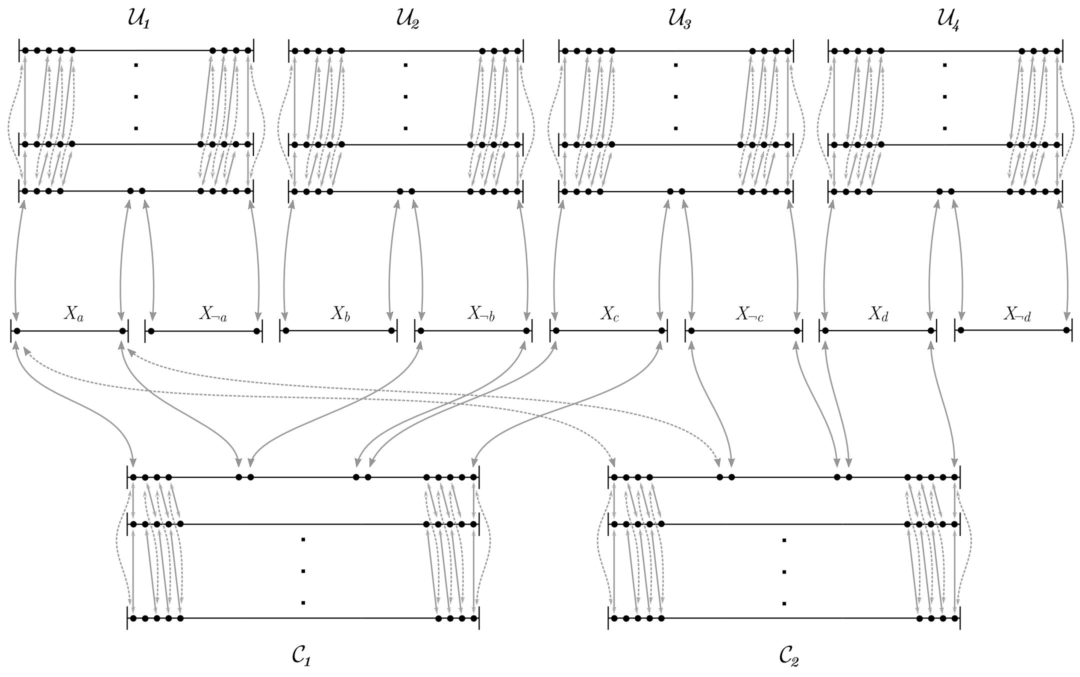


Figure 3.7: The instance of APD for 3SAT boolean formula $(a \vee \neg b \vee c) \wedge (a \vee \neg c \vee d)$.

to two gadgets of $k+3$ sequences, each representing one clause of the formula. This atomization instance also contains $k+3$ sequences of length $2L$ for each of the variables in the formula. The instance of atomization problem is shown in Figure 3.7.

3.3.2 NP-hardness of Atomization Problem

Theorem 3.3.2. *One-in-three 3SAT instance (U, C) is satisfiable if and only if the corresponding instance of the segmentation problem has a solution with cost at most B.*

Proof. We will prove the two implications of the stated theorem separately.

$\Rightarrow$: Let us first assume that instance (U, C) is satisfied by a truth assignment $t : U \to \{\mathrm{T}, \mathrm{F}\}$. We will create a corresponding atomization $\mathcal{A}$ with cost $\mathrm{c}_{(\epsilon,1)}(\mathcal{A}) \leq B$.

Consider variable y and the sequences X_y and $X_{\neg y}$ representing its positive and negative literals. Atomization $\mathcal{A}$ will contain one of these sequences as an atom and the other one as a waste region. One of the sequences X_y and $X_{\neg y}$ corresponds to the literal satisfied in the truth alignment t. We add this sequence as an atom to $\mathcal{A}$ and the other sequence will be completely covered by a waste region.

Each sequence in each gadget $\mathcal{U}_y$ and $\mathcal{C}_i$ will contain one atom. Each non-interface sequence will be completely covered by an atom. Interface sequences will contain one atom of length L in the region, which aligns to the sequence X_l for the satisfied literal l and waste region elsewhere. Note that there is always exactly one satisfied literal l for each variable y and each clause c_i.

It is easy to see that the cost of $\mathcal{A}$ is at most B. The total number of wasted bases is $2L(|U|+|C|) = B-1/2$. Since each atom has length at least $L > 2$, the total number of atoms is less than $n/2$, where n is the total length of the concatenated sequence of the APD instance. Therefore the contribution of the atom number to the total cost of $\mathcal{A}$ is less than $\epsilon \cdot n/2 = 1/2$ and the total cost $\mathrm{c}_{(\epsilon,1)}(\mathcal{A})$ is at most B. □

$\Leftarrow$: Now assume that we have an atomization $\mathcal{A}$ with cost at most B. We will prove that then the corresponding instance (U, C) of one-in-three 3SAT is satisfiable.

The interface sequence of each gadget has alignment endpoint every L symbols, and thus each atom in the interface can have length of at most L. However, by Lemma 3.3.1, each gadget contains exactly one atom in its interface to fulfill the requirement that atomization $\mathcal{A}$ costs at most B. The remaining regions within interface will be therefore waste.

We will construct truth assignment t for instance (U, C) so that literals corresponding to the atoms of $\mathcal{A}$ will be satisfied and the literals corresponding to waste regions will not be satisfied. In particular, let A_y be the atom within the interface of gadget $\mathcal{U}_y$. Atom A_y aligns to sequence X_l, where l is either y or $\neg y$. We will assign truth value $t(y)$ so that literal l is satisfied. Since atom A_y aligns to X_l and X_l has length L, it has to be an atom as well. Similarly, $X_{\neg l}$ aligns to a waste region in the interface of $\mathcal{U}_y$, and thus $X_{\neg l}$ is also completely covered by a waste region. Therefore, $t(l)$ is true for some literal l if sequence X_l is an atom and $t(l)$ is false if X_l is a waste region.

The interface of gadget $\mathcal{C}_i$ contains exactly one atom B_i and either two waste regions of length L or one waste region of length $2L$ due to Lemma 3.3.1. The alignment for the atom must map to another atom, and an alignment for a waste region must map to a waste region for $\mathcal{A}$ being an atomization. Suppose that clause c_i is $l_1 \vee l_2 \vee l_3$. We denote by l the literal of c_i, for which the sequence X_l is an atom. The truth value for l have been already set to true in the previous paragraph, because X_l is an atom, and the truth value of the other two literals of c_i have been already set to false as their corresponding sequences are waste regions. We have thus proven that the assignment t satisfies the corresponding one-in-three 3SAT instance (U, C). □

Since one-in-three 3SAT is NP-complete, decision version of atomization problem is also NP-complete, and therefore optimization version of atomization problem is NP-hard.

3.4 Heuristic Approach to Atomization Problem

Since the atomization problem is NP-hard, we have designed two heuristic algorithms for practical use. These algorithms create an atomization satisfying Definition 7, but possibly not the optimal one.

Input preprocessing. Our algorithms get as their input a set of evolutionary related sequences or a sequence which contains segmental duplications. We use the LASTZ program (Harris, 2007) to align every sequence to every other and also to itself. This way, we obtain significant local alignments scoring higher than the chosen threshold in some similarity measure of alignments. We later use these highly significant local alignments to delimit potential atoms.

Before running our algorithms, we make minor changes to the set of alignments. First, we split each alignment, which contains an indel longer than a given threshold max_indel, at the position of every long indel. Second, alignments that are shorter than a given threshold $min\_alignment$ after splitting are discarded. This way, we avoid low-quality short atoms and atomic classes that contain very dissimilar atoms.

Now we create initial waste regions of zero length at both ends of each alignment, as our problem definition requires that there are no alignment boundaries inside atoms. Thereafter, we join each pair of consecutive waste regions that are closer than L bases to each other and create a larger waste region. The reason is that if the distance between the smaller waste regions is smaller than L, no atom can be located between them.

After joining nearby waste regions along input sequence, we obtain atoms between each pair of consecutive waste regions. This set of atoms does not necessarily have to be an atomization as one atom possibly maps to a region overlapping two or

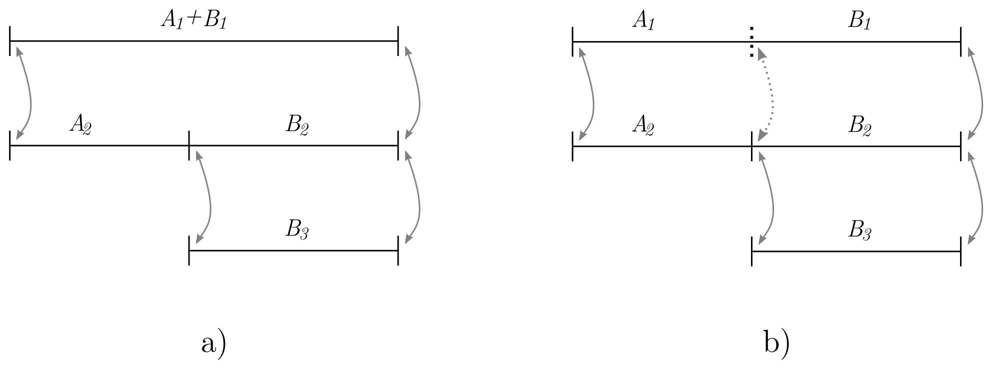


Figure 3.8: Iterative separation of atoms: a) Initial set of proto-atoms, where a waste region between atoms A_2 and B_2 aligns inside a proto-atom $A_1 + B_1$. b) The situation is solved by creating an additional waste region that splits proto-atom $A_1 + B_1$ to atoms A_1 and B_1.

more other atoms of the set. Our algorithms iteratively split the atoms and change their borders to get an atomization. We call *proto-atomization* the set of atoms at hand before each iteration and we call *proto-atoms* the atoms of proto-atomization.

Iterative mapping. The general idea of both algorithms is iterative mapping of waste regions. This approach is similar to iterative homology mapping introduced in Brejová et al. (2011), except that here we map whole waste regions instead of mapping each breakpoint independently. After each mapping iteration we gain a new set of proto-atoms that are separated from each other by newly expanded or newly created waste regions. If the number of proto-atoms and their lengths do not change in two consecutive iterations, the whole process ends and a set of proto-atoms from the last iteration becomes the final atomization of the input data.

An example of a situation resolved by this iterative approach is shown in Figure 3.8. Two atoms A_1 and B_1 are initially placed to the same proto-atom, because neither of them is covered by a short alignment that would separate them. But if one of the regions aligned to proto-atom $A_1 + B_1$ contains a waste region, we can map this waste region and thus separate atoms A_1 and B_1. Creation of new atoms A_1 and B_1, can have influence on some other atoms; therefore we repeat the re-mapping process while any changes of proto-atomization occur.

Situations similar to the previous example can be caused by the choice of similarity threshold, which we use to distinguish between significant and non-significant alignments. If alignment scores between pairs of atoms in some class are close to the similarity threshold, some of the alignments within the class pass the threshold, whereas others do not.

3.4.1 Waste Region Mapping Algorithm

The main focus of the first algorithm is to avoid mapping alignment ends into any atom. As this condition is not fulfilled for the initial set of proto-atoms, the algorithm iteratively creates new waste regions until the condition holds.

The algorithm stores for each waste region a tag indicating, whether the region has to be processed. During the first iteration, all the initial waste regions are tagged for processing. In each iteration, the algorithm handles each waste region W that has to be processed. For each alignment a that overlaps the first or the last position of region W, the algorithm identifies region $R = \text{map}(a, W)$ that aligns to W through alignment a. We do not require covering of whole waste region W by alignment a, as many alignments start and end in waste regions.

If region R is covered by a proto-atom, the proto-atom is split according to position of R into up to two smaller proto-atoms and a new waste region W_R identical to R. If any of the new proto-atoms is shorter than L, it is joined with W_R and with the waste region at the other end of the short proto-atom. The new waste region W_R is tagged for processing in the following iteration. On the other hand, if region R overlaps with an already existing waste region W', the algorithm distinguishes between the following two cases:

1. Waste region W' covers region R: nothing has to be done.

2. Otherwise: waste region W' is replaced by the union of W' and R. If any of the proto-atoms neighboring to the new waste region is shorter than L, the new waste region is extended so that it covers the short proto-atom and also the waste region at the other end of the short proto-atom. The new waste region is tagged for processing in the next iteration of the algorithm.

In this manner, the algorithm expands the set of waste regions until no waste regions remain to be processed. After that, we set atomization to be a set of sequences between each pair of consecutive waste regions.

This algorithm produces atomization fulfilling Definition 7, as each atom of the atomization is either fully covered by an alignment or the atom and the alignment do not overlap. In addition, each waste region of the atomization maps through an alignment overlapping the waste region inside another waste region. However, the created atomization is not necessarily the optimal one according to cost function $c_{(\epsilon,1)}(\cdot)$. The obtained atomization cannot contain any pair of atoms A and B mapping to each other such that the ends of A maps to the waste regions neighboring with B. The algorithm in this case shortens A and extends waste regions neighboring with A even in the cases when it increases cost of the output atomization. From this point of view the algorithm is more conservative than the atomization definition itself. If atoms A and B obtained by the algorithm map to each other through some alignment a, the first base of A maps first base of B and the last base of A maps to the last base of B.

3.4.2 Inverse Mapping to Proto-atom

This algorithm iteratively splits proto-atoms or extends existing waste regions until we get a proper atomization. In a situation depicted in Figure 3.8, where a single proto-atom $A_1 + B_1$ maps to two proto-atoms A_2 and B_2, we typically want to split the proto-atom $A_1 + B_1$ into two. However, if atoms A_2 and B_2 are separated by a longer waste region, there could be several possible places, where $A_1 + B_1$ could be split. One option would be to choose arbitrarily one of those places. However, this choice might lead to higher cost once we consider additional alignments. Instead, we collect such splitting requirements from all alignments covering the currently studied proto-atom, and choose the optimal set of new waste regions to satisfy all of them. Instead of splitting the proto-atom into two, we may also expand the waste region at one of its ends.

The algorithm iteratively processes every proto-atom P of the current proto-atomization and searches for a set W_{new} of new waste regions within the range of P. This set W_{new} satisfies requirements specified in Definition 10. A set of proto-atoms for the next iteration will be formed from the sets of the new waste regions. The process repeats until new waste regions do not form within any proto-atom and the proto-atomization from the last iteration becomes the final atomization.

Definition 10. *A set of new waste regions W_{new} within a range of proto-atom P satisfies the following requirements:*

1. *Each waste region that aligns with a part of proto-atom P overlaps with some waste region from W_{new}.*

2. *No region between two successive waste regions from W_{new} is completely covered by a waste region aligning to P.*

3. *Every region between two successive waste regions from W_{new} has length at least L bases.*

4. *The new set W_{new} has the lowest cost with respect to the cost function that penalizes each base within a waste region by 1 and each waste region by $\epsilon = 1/\text{length}(S)$.*

Search for the optimal set of waste regions. Now we describe in detail how to construct the optimal set W_{new} satisfying requirements of the previous definition. Let us consider a proto-atom $P = p_1 p_2 \dots p_m$ and a set of alignments $a_1, a_2, \dots, a_r$ that cover P. Note that in the rest of this section we refer to a subsequence of proto-atom P by the numbered spaces in P in accordance with Definition 1. We denote a subsequence of P from i to k by $P(i,k)$ for $0 \le i \le k \le m$. Our method of initial proto-atom construction guarantees that each alignment that overlaps with P by at least one base has to cover whole P. We denote as $R_1, R_2, \dots, R_r$ the regions that align to P through the alignments: $R_i = \text{map}(a_i, P)$, $1 \le i \le r$. Let W_i be the set of waste regions that overlap with R_i. We want to map W_i to proto-atom P but possibly some of the waste regions in W_i could have length zero, which is not considered in mapping Definition 5. Therefore, we extend the mapping definition in the following way:

Definition 11. *Let $P = p_1 p_2 \dots p_m$ be a proto-atom and $R = r_1 r_2 \dots r_z$ be a region that aligns with P through alignment a. Let $X = R(i,i)$ be a subsequence of R with length zero, on position i.* Mapping of zero length region X to P *through alignment a is the maximal region $P(j,k)$ such that each base r_y mappable through alignment a for $y < i$ maps to a base prior to p_j, and for $y \ge i$ maps to a base following p_k. We denote the mapping by* $\text{map}(a, X)$.

Let W_i' be the set of regions obtained by mapping each region of W_i into proto-atom P. We denote the union of all waste regions mapped to P by W: $W = \cup_{i=1}^{n} W_i'$.

We will use set W as an input for the algorithm computing set W_{new} of new waste regions inside proto-atom P.

We approach the task of computing W_{new} by dynamic programming: We sequentially process positions of proto-atom P and for each position i we compute the cost of the optimal waste region set for prefix $P(0, i)$ with i being part of the last waste region. This waste region set needs to overlap only those regions from W that start before i. We say that region $X = P(j, k)$ from W is the *preceding processed region* of position i if $k < i$ and for any other region $Z = P(r, s)$ from W either $s < k$, or $i \leq s$.

Simple dynamic programming. We will first describe details of dynamic programming for a simpler case when the set W does not contain any pair of mapped waste regions such that one of them covers the other. The algorithm processes only those positions in P that are covered by at least one region from W. Let i be covered by a region from W and let region $X = P(j, k)$ be the preceding processed region of i. Note that since we assume that no region from W covers another, there is only one preceding region. We denote the cost of the prefix $P(0, i)$ as $\mathrm{cost}(i)$ and calculate it as

$$\mathrm{cost}(i) = \min_{j \leq l \leq k} \mathrm{c}(i, l).$$

Each of the values $\mathrm{c}(i, l)$ corresponds to the optimal set of new waste regions for prefix $P(0, i)$ given that both l and i are waste positions in the set. We calculate these values in the following way:

$$\mathrm{c}(i, l) = \begin{cases} \mathrm{cost}(l) + (i - l) & \text{if } i - l \leq L, \\ \mathrm{cost}(l) + (i - l) & \text{if } i - l > L \text{ and some region } w \in W \text{contains} \\ & \text{both } i \text{ and } l, \\ \mathrm{cost}(l) + \epsilon & \text{if } i - l > L \text{ and none of the regions in } W \\ & \text{contains both } i \text{ and } l. \end{cases}$$

In the first two cases, the waste regions, which contain i and l are joined together. Otherwise, we would create a new proto-atom which would be too short or align to a waste region. In the third case, distance between i and l is more than L bases, which allows us to create a new proto-atom between them. As this possibility costs less than putting l and i into the same waste region, it is preferred.

For each i we also store a pointer to the position l in P which is to the left of i and have been used for computing the cost of the optimal waste region set on position i. After processing the last position m, we get the cost of the optimal set of waste regions for the whole proto-atom P. By tracing backward through the stored pointers, we identify the start and end positions of the regions of W_{new}. Since we process each region of W while constructing W_{new}, and we pick one position from each processed region to calculate the optimal cost of the succeeding region, for each region $Y \in W$ there is a region $Z \in W_{new}$, such that Y and Z overlaps.

Time complexity of processing P. Straightforward implementation of this algorithm runs in $O(|P|^2)$. Consider region $X = P(j,k) \in W$ and all positions such that X is preceding processed region for each of the positions. We denote these positions by $Y = P(a,b)$. In the following text, we describe how some additional values for each position of X can be precalculated so that calculation of cost(i) lasts constant time for each i such that $a \leq i \leq b$. Then, the calculation of the optimal cost of proto-atom P runs in time $O(|P| + |W|)$ but due to our simplifying assumption that no region in W is covered by another $|W| \leq |P|$. We sort all regions from W according to their first position. For each position i we store the first region of the sorted list, which covers i. We denote the region by F_i. This preprocessing can be done in $O(|P|)$ time.

For a fixed $i \in Y$ we can split X to two regions V_1 and V_2. Region V_1 is the first part of X such that for each $l \in V_1$ holds $\mathrm{c}(i,l) = \mathrm{cost}(l) + \epsilon$. Conversely, for each position $l \in V_2$ the value $\mathrm{c}(i,l)$ is computed as $\mathrm{cost}(l) + (i - l)$. With increasing i, the boundary between V_1 and V_2 moves towards k. Suppose now that l is a position in V_2 and i is a fixed position of Y. We can write the term $\mathrm{c}(i,l)$ as a sum of two values $\mathrm{cost}(l) - l$ and i, so that the first value does not depend on i and the second on l. We compute values $\mathrm{cost}(l) - l$ for each position $l \in X$ such that $i - l \leq L$ or F_i covers l. For each such l we store minimum $m[l]$ of the computed values over positions of $P(l,k)$, suffix of X starting on position l. We also store the position $p[l]$ for which the attained value is $m[l]$. Further, we compute values $\mathrm{cost}(t)$ for each position $t \in X$ and for each l store $n[l]$ being the minimum of values $\mathrm{cost}(t)$ over positions of $P(j, l-1)$, prefix of X which ends one position before l. We also store the position $q[l]$ for which the attained value is $n[l]$.

While searching the optimal $\mathrm{cost}(i)$, we check values $m[l]$ and $n[l]$ for a particular position l and choose value $\min(m[l]+i, n[l])$ to become $\mathrm{cost}(i)$. The position l is minimum from $i-L$ and the first position of F_i.

Extension to arbitrary set W. The situation becomes more complicated, when set W is allowed to contain pairs of regions, such that one of them covers the other. To handle such inputs, we split the set W into two parts. Set of covering regions C consists of the regions from W, which cover some other region of W. We denote by W' the set $W \setminus C$. If requirement 1 of Definition 10 is satisfied for every region of W', the requirement will be satisfied also for every region of C and therefore for all regions of W. But to satisfy requirement 2 of the definition, we have to consider all regions of W. Therefore, we will run the dynamic programming on regions from the set W', but values of function $\mathrm{c}(\cdot,\cdot)$ will be calculated with respect to all regions of W:

$$\mathrm{c}(i,l) = \begin{cases} \mathrm{cost}(l) + (i-l) & \text{if } i-l \leq L, \\ \mathrm{cost}(l) + (i-l) & \text{if } i-l > L \text{ and some region } w \in W \text{ contains} \\ & \text{both } i \text{ and } l, \\ \mathrm{cost}(l) + \epsilon & \text{if } i-l > L \text{ and none of the regions in } W \\ & \text{contains both } i \text{ and } l. \end{cases}$$

Atom Classification

As atom classes, we use connected components of the alignment graph. Sometimes the result contains a pair of classes A and B such that each atom of class A neighbors with an atom of class B from the left side and each atom of class B neighbors with an atom of class A from the right side. In that case, we join each pair of these atoms to a larger atom, and create one class of the joined atoms instead the two classes A and B.

Time Complexity of the Algorithms

The number of iterations of both algorithms is polynomial, since in each iteration we either detect that no further changes were made for any atom, or we increase either the number of atoms or the number of wasted bases at least by one. In reality, the number of iterations is relatively low for both algorithms. On the simulated data, the *iterative mapping to proto-atom* (IMP) performs $1 - 4$ iterations, the *waste region mapping* (WRM) performs $2 - 22$ iterations. On the real data described in the following section, the algorithms perform 3 and 42 iterations, respectively.

The difference between the iteration counts of the two algorithms is caused by the fact, that IMP processes for each proto-atom boundaries of all alignable proto-atoms at once, while WRM processes them sequentially in several iterations. Another distinction is that WRM has stricter requirements on atoms, which may take longer to establish. Consider proto-atom A, its neighboring waste regions w_L and w_R, proto-atom B and its neighboring waste regions v_L and v_R, and several different alignments mapping the region containing w_L,A, and w_R to the region containing v_L, B, and v_R. Suppose that some alignment maps bases located close to the ends of proto-atom A to bases in v_L and v_R, while some other alignment maps the same bases of A to the bases within B. Similarly, some alignment maps bases of B into A and another one maps the same bases to the surrounding regions w_L and w_R. In this situation, WRM will iteratively shorten A and B and extend the waste regions until boundary bases of A and B are found such that alignments map them to each other. As IMP allows mapping bases from an atom to the bases belonging to a waste region, this algorithm does not perform additional iterations in such a situation.

3.5 Experiments

In this section, we present comparison of the proposed algorithms both on the simulated and also on the real sequences. Advantage of the simulated data is that we already know the true atomization and can evaluate performance of the algorithms. It is not so easy to evaluate performance on the real sequences, but it is important to keep in mind that even when performance of the algorithms on the simulated data are often similar for the algorithms, this is not the case when the complex evolutionary processes take place.

3.5.1 Measures of the Predicted Atomization Quality

For generated data, we have the true atomizations available, and therefore we can evaluate quality of the predicted atomization directly by comparison with the true atomization. We will use two sets of measures. The first one, based on reciprocal best matches, was defined for the segmentation quality comparison previously in Brejová et al. (2011). We propose the second set of measures for more precise differentiation of the results.

To compare two atomizations, we first compute *reciprocal best matches* (BRM) and *boundary fitting matches* (BFM) for their atoms. Suppose that atom A belongs to the true and atom B to the predicted atomization. Atoms A and B are BRM to each other, if they overlap and no other atom overlaps with any of them by a larger amount than A and B overlap each other. Atoms A and B are BFM to each other, if starts of the atoms are close to each other, as well as their ends. We consider the boundaries of A and B to be close enough, if their distance is less than some fraction of the minimal atom length L; in the experiments we used $\lceil L/4 \rceil$ for simulated data sets and $\lceil L/2 \rceil$ for real data.

We use BRM and BFM to define quality measures for predicted atoms and also for atom classes. Let b be the number of BRM pairs between two atomizations, c be the number of BFM, p be the number of predicted atoms, and t be the number of true atoms. We use the following quality measures for atoms:

Measure	Value	Measure	Value
BRM specificity	b/p	BFM specificity	c/p
BRM sensitivity	b/t	BFM sensitivity	c/t

The BRM specificity decreases for example when several predicted atoms overlap one true atom. The true atom creates a BRM pair with only one of the predicted atoms, while all the predicted atoms are included in p. In this situation, a BFM pair does not have to exist for the true atom, which decreases the BFM specificity further. Conversely, if a predicted atom covers several true atoms, only one of the true atoms creates a BRM pair with the predicted atom, but all the true atoms are summed in t. This decreases a BRM sensitivity of the predicted atomization. Again, this effect can be even more significant for the BFM sensitivity, as BFM pair does not have to exist.

Now we describe four measures, which evaluate correctness of the grouping of the predicted atoms into classes. Let A be a class of true atoms and B be a class of predicted atoms. We consider B to be the correct prediction of A, if each atom of B has its BRM/BFM pair in A and each atom of A has its BRM/BFM pair in B. Let d be a number of correctly predicted classes according to BRM, e be a number

	Sequence	Segmentation	
Data set	length (kb)	No. atoms	No.classes
Slow	253	153	55
Fast	444	385	113
No indels	210	611	78

Table 3.1: Overview of simulated data sets. Each group contains ten data sets. The table shows mean number of atoms and atomic classes over these data sets, and a mean length of a simulated sequence for one species.

of correctly predicted classes according to BFM, p be a number of predicted classes, and t be a number of true classes. Then sensitivity and specificity of the predicted classes will be:

Measure	Value	Measure	Value
BRM class specificity	d/p	BFM class specificity	e/p
BRM class sensitivity	d/t	BFM class sensitivity	e/t

3.5.2 Simulated Data

Data sets. We have evaluated performance of the algorithms on 30 simulated data sets, taken from Brejová et al. (2011). These data sets are divided into three groups according to different parameter settings of the used generators. Each data set contains three sequences that follow known features of human, chimp, and macaque genomic sequences as well as their evolutionary relationship. The Fast and Slow data sets were produced by simulation of sequence evolution, allowing substitutions according to the HKY model (Hasegawa et al., 1985), short insertions and deletions, as well as large-scale deletions and duplications. The data sets differ in the rate of large-scale events per site, where the Fast data sets use the rate 1.5 faster than the Slow data sets. The No Indels data sets were taken form Vinar et al. (2010) and they were produced by a simpler generator. Short insertions and deletions were not simulated and the rate of large-scale duplication and deletion did not depend on the sequence length. Table 3.1 gives us a brief overview of the simulated data structure.

Parameter settings. We have compared performance of our two new algorithms *waste region mapping* (WRM) and *iterative mapping to proto-atoms* (IMP) with the algorithm *iterative homology mapping* (IHM) presented in Brejová et al. (2011).

The IHM algorithm solves the atomization problem by iterative mapping individual breakpoints (waste regions of length zero) given that at most one breakpoint can be located within a window of size W. The process repeats until no new breakpoints are identified. An atom forms between every pair of consecutive breakpoints. These atoms are further divided into classes by a integer linear program.

IHM does not create any waste regions of non-zero length and in problematic regions it tends to create atoms of length W. To make the IHM comparable with the other two algorithms, we have used a relatively small parameter W and then discarded classes containing atoms shorter than minimal required length L. We use the short atom filtering as a substitute for identification of the regions where alignment boundaries (or their mappings) occur frequently.

We have tested performance of the three algorithms (WRM, IMP, IHM) with four different minimal atom lengths $L \in \{50, 100, 250, 500\}$. Parameters max_indel and $min\_alignment$ of WRM and IMP have been set to $M = \lceil L/4 \rceil$, and window size parameter W has been set to the same value as M.

For comparison, we also evaluate the true atomization from which the atoms shorter than L have been discarded. This shows us the best possible performance of an algorithm which does not overestimate lengths of atoms. This data are displayed in the Tables 3.2 and 3.3 in the columns TRUE.

No Indels data sets (Table 3.2): All three algorithms have similar accuracy under the BRM measures. However, there are significant differences in the BFM measures. At the level of atoms, IHM consistently outperforms IMP, which is always better than WRM. For each of the programs, the BFM sensitivity is higher for $L \in \{100, 250\}$ than for $L \in \{50, 500\}$.

Let us now look at the data for $L = 50$. The BFM sensitivity of WRM is low, but the BRM measures are as high as for any other program and on average almost whole input sequence is covered by the predicted atomization. The WRM atomization usually contains atoms approximately at the correct positions, but the boundaries of the predicted atoms are not close enough to the boundaries of the true atoms. Most of the times, the predicted atom is shorter. If several alignments which cover the same true atom, WRM and IMP create the predicted atom at the region which is an intersection of these alignments. Detection of the correct homology boundaries is difficult for the alignment tool; therefore some of the alignments may be shorter

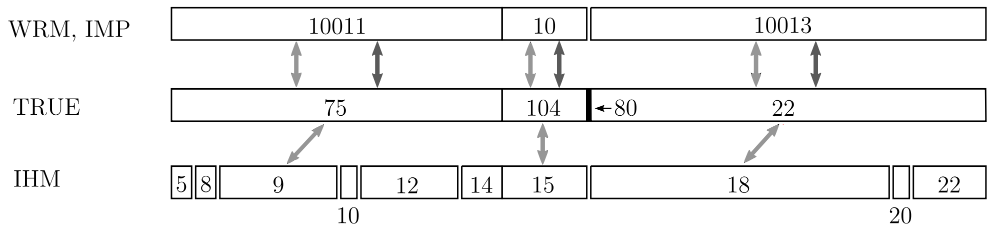


Figure 3.9: Atomizations of short region from the set obtained by algorithms at $L = 100$. Atoms are shown as numbered rectangles, where the number is the ID of the predicted or true class. The BRM pairs shown as light arrows and the BFM pairs as dark ones. The IHM predicts many short atoms, leading to low specificity. The BRM specificity is higher than the BFM specificity for IHM, as the BRM pairs exist but the BFM pairs do not. The BRM and the BFM specificities equals for WRM and IMP atomizations.

than the true atom. This is the reason, why WRM and IMP predict shorter atoms and have lower values of BFM measures compared to IHM. The WRM, unlike the IMP, then propagates distribute the boundaries of the short atom to all atoms of the same class, which causes the difference between performance of the two algorithms.

Fast and Slow data sets (Table 3.3): Similarly to No Indels data sets, the BRM sensitivities for all three algorithms are at the level of atoms and also at the class level similar to each other and very close to the BRM sensitivity of TRUE data sets. The exception is the BRM specificity of IHM, which is significantly lower than the BRM specificity of WRM and IMP particularly at low values of L. This is caused by IHM predicting many small atoms which do not have their BRM pairs in the true atomization (see Figure 3.9). For bigger L the specificity difference get smaller and for $L = 500$ the BRM specificity of IHM approaches 100%. Figure 3.9 also illustrates the difference between BFM and BRM specificities.

The BFM/BRM class specificities are usually lower for IMP than for WRM atomizations, even when the BFM/BRM specificities for atoms are the same or even better. This is typically caused by several atoms predicted by IMP, which are not found by WRM. These atoms usually belong to the classes which are missed by WRM. The problem is that not every atom from the class is identified by IMP. Therefore the number of the predicted classes is higher for IMP than for WRM, but the number of the correctly predicted classes stays the same for both algorithms, leading to lower class specificity.

In Fast and Slow data sets simulation, short indels were taken into account. This is the biggest difference between these data sets and the No indels data, and it makes these two data sets more realistic than the No indel ones. For Fast and Slow data sets, our algorithms IMP and WRM outperform previous IHM algorithm, particularly for the short atomic lengths.

		dataset: No indels															
		$L = 50, W = M = 13$				$L = 100, W = M = 25$				$L = 250, W = M = 63$				$L = 500, W = M = 125$			
measure		WRM	IMP	IHM	TRUE	WRM	IMP	IHM	TRUE	WRM	IMP	IHM	TRUE	WRM	IMP	IHM	TRUE
	coverage	98 %	99 %	100 %	100 %	98 %	99 %	99 %	100 %	95 %	96 %	97 %	97 %	89 %	90 %	90 %	90 %
BRM	sp	94 %	93 %	99 %	100 %	100 %	99 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %
	sn	94 %	94 %	94 %	95 %	89 %	90 %	91 %	91 %	70 %	71 %	74 %	73 %	53 %	54 %	56 %	56 %
	class sp	97 %	92 %	98 %	100 %	100 %	94 %	100 %	100 %	100 %	97 %	100 %	100 %	100 %	97 %	100 %	100 %
	class sn	94 %	92 %	94 %	96 %	89 %	88 %	91 %	92 %	74 %	74 %	78 %	77 %	58 %	58 %	61 %	62 %
BFM	sp	57 %	70 %	87 %	100 %	78 %	89 %	96 %	100 %	93 %	96 %	98 %	100 %	98 %	98 %	100 %	100 %
	sn	57 %	71 %	82 %	95 %	70 %	81 %	87 %	91 %	65 %	68 %	72 %	73 %	52 %	53 %	55 %	56 %
	class sp	61 %	57 %	84 %	100 %	80 %	76 %	95 %	100 %	94 %	92 %	98 %	100 %	97 %	94 %	99 %	100 %
	class sn	59 %	57 %	81 %	96 %	71 %	71 %	87 %	92 %	69 %	70 %	76 %	77 %	57 %	56 %	61 %	62 %

Table 3.2: Performance of the WRM, IMP, and IHM algorithms on the No Indels generated data sets.

measure		dataset: Slow															
		$L = 50, W = M = 13$				$L = 100, W = M = 25$				$L = 250, W = M = 63$				$L = 500, W = M = 125$			
		WRM	IMP	IHM	TRUE	WRM	IMP	IHM	TRUE	WRM	IMP	IHM	TRUE	WRM	IMP	IHM	TRUE
	coverage	100 %	100 %	100 %	100 %	100 %	100 %	99 %	100 %	100 %	100 %	100 %	100 %	99 %	99 %	99 %	99 %
BRM	sp	63 %	62 %	34 %	100 %	100 %	99 %	53 %	100 %	100 %	100 %	86 %	100 %	100 %	100 %	100 %	100 %
	sn	97 %	98 %	98 %	99 %	96 %	96 %	96 %	97 %	93 %	93 %	94 %	95 %	84 %	84 %	86 %	86 %
	class sp	63 %	61 %	36 %	100 %	100 %	97 %	59 %	100 %	100 %	100 %	88 %	100 %	100 %	100 %	100 %	100 %
	class sn	98 %	98 %	98 %	99 %	97 %	97 %	97 %	98 %	94 %	94 %	95 %	96 %	86 %	87 %	89 %	89 %
BFM	sp	31 %	33 %	11 %	100 %	81 %	83 %	25 %	100 %	91 %	93 %	69 %	100 %	98 %	97 %	99 %	100 %
	sn	47 %	52 %	32 %	99 %	78 %	81 %	45 %	97 %	85 %	87 %	76 %	95 %	82 %	82 %	85 %	86 %
	class sp	33 %	31 %	14 %	100 %	82 %	78 %	32 %	100 %	93 %	92 %	73 %	100 %	97 %	96 %	99 %	100 %
	class sn	52 %	49 %	39 %	99 %	79 %	78 %	53 %	98 %	87 %	87 %	79 %	96 %	84 %	84 %	87 %	89 %

measure		dataset: Fast															
		$L = 50, W = M = 13$				$L = 100, W = M = 25$				$L = 250, W = M = 63$				$L = 500, W = M = 125$			
		WRM	IMP	IHM	TRUE	WRM	IMP	IHM	TRUE	WRM	IMP	IHM	TRUE	WRM	IMP	IHM	TRUE
	coverage	100 %	100 %	100 %	100 %	99 %	100 %	99 %	100 %	99 %	99 %	99 %	100 %	98 %	98 %	99 %	99 %
BRM	sp	77 %	75 %	46 %	100 %	100 %	98 %	67 %	100 %	100 %	100 %	90 %	100 %	100 %	100 %	100 %	100 %
	sn	96 %	96 %	97 %	97 %	93 %	94 %	95 %	95 %	87 %	88 %	89 %	90 %	76 %	77 %	79 %	80 %
	class sp	74 %	72 %	47 %	100 %	100 %	98 %	71 %	100 %	100 %	99 %	92 %	100 %	100 %	99 %	100 %	100 %
	class sn	97 %	97 %	97 %	98 %	95 %	95 %	96 %	97 %	90 %	90 %	92 %	93 %	82 %	83 %	85 %	86 %
BFM	sp	44 %	47 %	19 %	100 %	80 %	83 %	42 %	100 %	93 %	93 %	77 %	100 %	97 %	98 %	99 %	100 %
	sn	55 %	61 %	40 %	97 %	75 %	79 %	59 %	95 %	80 %	82 %	76 %	90 %	74 %	75 %	78 %	80 %
	class sp	45 %	42 %	22 %	100 %	81 %	79 %	46 %	100 %	93 %	92 %	79 %	100 %	97 %	96 %	99 %	100 %
	class sn	59 %	57 %	45 %	98 %	77 %	77 %	63 %	97 %	84 %	84 %	80 %	93 %	80 %	80 %	83 %	86 %

Table 3.3: Performance of the WRM, IMP, and IHM algorithms on the Fast and Slow generated data sets.

	atoms	classes	coverage
WRM	87	17	67%
IMP	126	24	77%
IHM	134	22	88%

Table 3.4: Summary information about performance of the algorithms on cluster UGT1A.

3.5.3 Real Data

We have applied the three algorithms also to sequences of the primate gene cluster UGT1A (UDP-glucuronosyltransferase) of human, chimpanzee, and orangutan. This gene cluster, consists in the human genome of at least 13 duplicated copies of one short exon or its parts (Bellemare et al., 2010). Before running alignments and atomization algorithms, we discarded sequence repeats from the UGT1A sequences by RepeatMasker (Harris, 2007). After this, the human sequence has length 89 971 base pairs, sequences of chimpanzee and orangutan are 87 239, and 108 475 base pairs long, respectively.

We have run the WRM, IMP and IHM algorithms on the data with parameters $L = 500, W = M = 250$. Several statistics of the resulting atomizations can be seen in Table 3.4. We also compare the results of our two algorithms to the results of IHM by computing BRM and BFM measures under the assumption that IHM is the true atomization. The results in Table 3.5 indicate that differences between the algorithms pronounced in real data than for simulated data sets. As we can see, WRM is more specific and IMP has higher sensitivity, when compared to IHM.

Consider now only human sequence and its atomizations, as sequence of the considered protein (UDP-glucuronosyltransferase) is best characterized for this species of the three. Now we discuss a number of the duplicated copies contained in atoms of WRM, IMP, and IHM atomizations and also a number of classes into which these atoms are assigned. We have downloaded a protein sequence of the duplicated exon from UCSC genome browser (Kuhn et al., 2012). We have also used BLASTX alignment program (Gish and States, 1993) to search for similarities between genomic sequence of UGT1A gene cluster and the exon sequence; 13 regions of the genomic sequence were identified as significantly similar to the protein sequence. We will call these regions *putative exons*. Length of a putative exon is on average 820

	BRM				BFM			
	sp	sn	class sp	class sn	sp	sn	class sp	class sn
WRM	100 %	65 %	94 %	70 %	32 %	21 %	41 %	30 %
IMP	90 %	84 %	67 %	70 %	31 %	30 %	33 %	35 %

Table 3.5: Comparison of WRM and IMP atomizations to IHM atomization.

bases and pairwise sequence similarity among these putative exons on the nucleotide level is on average 71 % according to their multiple sequence alignment obtained by ClustalW2 (Goujon et al., 2010). As the average pairwise similarity is relatively low we can expect that some of the putative exons will belong to a different class than other exons.

We say that an atom *fits* a putative exon, if an overlap of the atom and the putative exon contains at least 50 % bases of each. If an overlap between an atom and a putative exon contains more than 50 % bases of the putative exon but less than 50 % of the atom, we say that the atom *coincide* with the putative exon. We ignore overlaps of an atom and a putative exon containing less than 10 % of the putative exon. If an atomic class contains only atoms which fit putative exons, we call it *class of fitting atoms* and a class consisting of only coinciding atoms is *class of coinciding atoms.*

In Table 3.6, we can see distribution of putative exons in atomizations produced by WRM, IMP, and IHM. Fitting atoms of IMP have been divided to two classes; one containing 6 atoms and the other 2. Of the correctly predicted atoms by IMP and IHM, 5 are common for both, 2 are unique for IHM and 3 for IMP. One of the three atoms unique for IMP belongs to the same class as the five atoms common for IHM and IMP, the other two atoms create isolated class. On average, IMP fitting atoms are shorter by 120 bases than the corresponding IHM fitting atoms; average length of IHM fitting atoms is 845 bases. When we increase minimal alignment length parameter M to 500, IMP detects also the two correct atoms, which are unique to IHM for $M = 250$, we show data for this atomization in column IMP' of Table 3.6. We expect, that the number of fitting IMP' atoms increased thank to elimination of the input alignments of lengths between 250 and 500 base pairs.

In this section we have seen, that WRM cannot correctly predict the putative exons of UGT1A gene cluster even when performance on the simulated data sets

	WRM	IMP	IMP'	IHM
Fitting atoms	0	8	11	7
Classes of fitting atoms	0	2	2	1
Coinciding atoms	2	1	0	6
Classes of coinciding atoms	2	1	0	3
Putative exons in waste regions	11	4	2	0

Table 3.6: Distribution of UGT1A exons in atomizations obtained by the three algorithms.

were comparable to the other two algorithms. This is caused by alignment ends scattered over a putative exon what leads WRM algorithm to put the whole area into a waste region. As we can see in Table 3.4, in addition to the sequences coding the protein, many non-coding regions also occur in several copies of any of the three atomizations. This is in accordance with our previous knowledge that the considered genomic sequence is highly structured.

Conclusion

This thesis deals with two problems connected to similarity search in genomic databases. In particular, we consider sequences which do not follow the traditional sequence models assuming random distribution of bases within a sequence. Instead, we expect that the sequence at hand is internally structured, containing several almost identical regions. We study possibility of using this information to improve methods for solving two computational problems.

First, we address the problem of estimating statistical significance to database search for a database consisting of many similar sequences. We have considered methods for more accurate estimation of P-values in the context of sequence homology search. In particular, we propose to adjust the size of the database to compensate for the structure present in the database due to the fact that individual sequences are related by evolution.

We have explored the idea of using compression to estimate the effective database size. We evaluate this approach in a simple scenario where we can compare our result with exactly computed P-values. We have demonstrated by experiments that the use of the compression algorithms leads to non-conservative P-value estimates for small P-values. This is at least partially caused by the fact that besides identifying longer repeated substrings, the compression algorithms also take advantage of sequences with low entropy. We have suggested a simple way to disentangle the portion of the compression coming from locally low entropy and shown that the correction leads to better estimates of P-value.

The compression is a simple and efficient way of estimating the effective database size. Our scheme consist of several existing modules (compression, homology search, P-value estimation) which can be easily substituted with other alternatives. With sequences of many related species becoming available, our method could be a practical alternative for P-value estimation.

Finally, there is a space for testing and adjusting our approach in more complex scenarios of homology search, including longer query sequences, insertions, deletions, more complex scoring schemes, and local alignment results. It would be also useful to address further theoretical problems, for example, analytical P-value estimates for redundant databases.

Second, we have studied a sequence segmentation problem arising in comparative analysis of related DNA sequences. We are given a genomic sequence, and our task is to identify non-overlapping segments of the sequence and to partition these segments into classes so that segments within each class are similar to each other, and there are no significant similarities between segments from different classes. We have defined the problem formally, proved its NP hardness and provided two practical heuristic algorithms, the simpler one using only iterative remapping of waste regions, and the other using dynamic programming to select an optimal combination of several local changes. We have implemented and evaluated the algorithms on simulated data to show that our problem definition and algorithmic approach lead to reasonable results with respect to a simple model of evolution generating our data. We have also run the two algorithms on genomic sequences of a real gene cluster and mention weak points of the heuristics.

This area offers both practical and theoretical questions for further study. From a theoretical point of view, it would be interesting to study approximation or fixed-parameter algorithms for our problem, or to study its variants. From the practical point of view, improvement of the IMP algorithm leading to better identification of the real segments would be useful.

Bibliography

Altschul, S. F., Bundschuh, R., Olsen, R., and Hwa, T. (2001). The estimation of statistical parameters for local alignment score distributions. *Nucleic Acids Research*, 29(2):351–361.

Altschul, S. F., Gish, W., et al. (1996). Local alignment statistics. *Methods in enzymology*, 266(2):460–480.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410.

Bailey, D. and Crandall, R. (2002). Random generators and normal numbers. *Experimental Mathematics*, 11(4):527–546.

Behzadi, B. and Le Fessant, F. (2005). DNA compression challenge revisited: a dynamic programming approach. In *Combinatorial Pattern Matching (CPM 2005)*, pages 190–200. Springer.

Bellemare, J., Rouleau, M., Girard, H., Harvey, M., and Guillemette, C. (2010). Alternatively spliced products of the UGT1A gene interact with the enzymatically active proteins to inhibit glucuronosyltransferase activity in vitro. *Drug Metabolism and Disposition*, 38(10):1785–1789.

Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., and Wheeler, D. L. (2008). GenBank. *Nucleic acids research*, 36(Database issue):D25–30.

Blanchette, M., Kent, W. J., Riemer, C., Elnitski, L., Smit, A. F., Roskin, K. M., Baertsch, R., Rosenbloom, K., Clawson, H., Green, E. D., et al. (2004). Aligning multiple genomic sequences with the threaded blockset aligner. *Genome research*, 14(4):708–715.

Blanchette, M., Kunisawa, T., and Sankoff, D. (1999). Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution*, 49(2):193–203.

Bourque, G., Pevzner, P. A., and Tesler, G. (2004). Reconstructing the Genomic Architecture of Ancestral Mammals: Lessons From Human, Mouse, and Rat Genomes. *Genome Research*, 14:507–516.

Brannan, D. A. (2006). *A first course in mathematical analysis*. Cambridge University Press.

Bray, N. and Pachter, L. (2004). Mavid: constrained ancestral alignment of multiple sequences. *Genome research*, 14(4):693–699.

Brejová, B., Brown, D. G., and Vinař, T. (2005). Vector seeds: an extension to spaced seeds. *J. Comput. Syst. Sci.*, 70(3):364–380.

Brejová, B., Burger, M., and Vinař, T. (2011). Automated Segmentation of DNA Sequences with Complex Evolutionary Histories. In Przytycka, T. M. and Sagot, M.-F., editors, *Algorithms in Bioinformatics, 11th International Workshop (WABI)*, volume 6833 of *Lecture Notes in Computer Science*, pages 1–13, Saarbrücken, Germany. Springer.

Brown, D. G. (2006). *Computational Genomics: Current Methods*, chapter 5, pages 95–120. Taylor & Francis.

Brudno, M., Do, C. B., Cooper, G. M., Kim, M. F., Davydov, E., Green, E. D., Sidow, A., Batzoglou, S., et al. (2003a). Lagan and multi-lagan: efficient tools for large-scale multiple alignment of genomic dna. *Genome research*, 13(4):721–731.

Brudno, M., Malade, S., Poliakov, A., Do, C., Couronne, O., Dubchak, I., and Batzoglou, S. (2003b). Glocal alignment: finding rearrangements during alignment. *Bioinformatics*, 19(1):i54–i62.

Burrows, M. and Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California.

Chen, X., Kwong, S., and Li, M. (1999). A compression algorithm for DNA sequences and its applications in genome comparison. In *Genome Informatics (GIW'99)*, pages 51–61. Universal Academy Press.

Chen, X., Li, M., Ma, B., and Tromp, J. (2002). DNACompress: fast and effective DNA sequence compression. *Bioinformatics*, 18(12):1696–1698.

Choi, V., Zheng, C., Zhu, Q., and Sankoff, D. (2007). Algorithms for the extraction of synteny blocks from comparative maps. In Giancarlo, R. and Hannenhalli, S., editors, *Algorithms in Bioinformatics*, volume 4645 of *Lecture Notes in Computer Science*, pages 277–288. Springer Berlin Heidelberg.

Collins, J. F., Coulson, A. F., and Lyall, A. (1988). The significance of protein sequence similarities. *Computer applications in the biosciences : CABIOS*, 4(1):67–71.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1989). *Introduction to algorithms.* McGraw-Hill.

Darling, A., Mau, B., and Perna, N. (2010). progressiveMauve: multiple genome alignment with gene gain, loss and rearrangement. *PLoS ONE*, 5(6):e11147.

Dembo, A. and Karlin, S. (1991a). Strong limit theorems of empirical distributions for large segmental exceedances of partial sums of markov variables. *The Annals of Probability*, 19(4):1756–1767.

Dembo, A. and Karlin, S. (1991b). Strong limit theorems of empirical functionals for large exceedances of partial sums of iid variables. *The Annals of Probability*, pages 1737–1755.

Dembo, A., Karlin, S., and Zeitouni, O. (1994). Limit distribution of maximal non-aligned two-sequence segmental score. *The Annals of Probability*, pages 2022–2039.

Deorowicz, S. and Grabowski, S. (2011). Robust relative compression of genomes with random access. *Bioinformatics*, 27(21):2979–2986.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA.

Giancarlo, R., Scaturro, D., and Utro, F. (2009). Textual data compression in computational biology: a synopsis. *Bioinformatics*, 25(13):1575–1576.

Gish, W. and States, D. J. (1993). Identification of protein coding regions by database similarity search. *Nature genetics*, 3(3):266–272.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of molecular biology*, 162(3):705–708.

Goujon, M., McWilliam, H., Li, W., Valentin, F., Squizzato, S., Paern, J., and Lopez, R. (2010). A new bioinformatics analysis tools framework at embl–ebi. *Nucleic acids research*, 38(suppl 2):W695–W699.

Graham, R., Knuth, D., and Patashnik, O. (1989). *Concrete mathematics*. Addison-Wesley, Reading, MA.

Hachiya, T., Osana, Y., Popendorf, K., and Sakakibara, Y. (2009). Accurate identification of orthologous segments among multiple genomes. *Bioinformatics*, 25(7):853–860.

Harris, R. (2007). *Improved pairwise alignment of genomic DNA*. PhD thesis, Pennsylvania State University.

Hartl, D. L. and Clark, A. G. (2006). *Principles of Population Genetics, Fourth Edition*. Sinauer Associates.

Hasegawa, M., Kishino, H., and Yano, T.-a. (1985). Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *Journal of molecular evolution*, 22(2):160–174.

Höhl, M., Kurtz, S., and Ohlebusch, E. (2002). Efficient multiple genome alignment. *Bioinformatics*, 18(suppl 1):S312–S320.

Holloway, P., Swenson, K., Ardell, D., and El-Mabrouk, N. (2012). Evolution of genome organization by duplication and loss: An alignment approach. In Chor, B., editor, *Research in Computational Molecular Biology*, volume 7262 of *Lecture Notes in Computer Science*, pages 94–112. Springer Berlin Heidelberg.

Holmes, R. and Jobling, M. (1996). *Medical Microbiology*, chapter 5. University of Texas Medical Branch at Galveston, 4th edition. Available from: http://www.ncbi.nlm.nih.gov/books/NBK7908/.

Hutchison, C. (2007). DNA sequencing: bench to bedside and beyond. *Nucleic acids research*, 35(18):6227–6227.

Jaitly, D., Kearney, P., Lin, G., and Ma, B. (2002). Methods for reconstructing the history of tandem repeats and their application to the human genome. *Journal of Computer and System Sciences*, 65(3):494–507.

Karlin, S. and Altschul, S. F. (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87(6):2264–2268.

Karlin, S. and Dembo, A. (1992). Limit distributions of maximal segmental score among markov-dependent partial sums. *Advances in Applied Probability*, pages 113–140.

Karlin, S., Dembo, A., and Kawabata, T. (1990). Statistical composition of high-scoring segments from molecular sequences. *The Annals of Statistics*, 18(2):571–581.

Kent, W. J., Sugnet, C. W., Furey, T. S., Roskin, K. M., Pringle, T. H., Zahler, A. M., and Haussler, D. (2002). The human genome browser at UCSC. *Genome Research*, 12(6):996–1006.

Korodi, G. and Tabus, I. (2005). An efficient normalized maximum likelihood algorithm for dna sequence compression. *ACM Transactions on Information Systems*, 23(1):3–34.

Kováč, J., Brejová, B., and Vinař, T. (2011). A practical algorithm for ancestral rearrangement reconstruction. In Przytycka, T. and Sagot, M.-F., editors, *Algorithms in Bioinformatics*, volume 6833 of *Lecture Notes in Computer Science*, pages 163–174. Springer Berlin Heidelberg.

Kuhn, R. M., Haussler, D., and Kent, W. J. (2012). The ucsc genome browser and associated tools. *Briefings in bioinformatics*.

Li, M., Badger, J. H., Xin, C., Kwong, S., Kearney, P., and Zhang, H. (2001). An information based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17(2):149–154.

Li, M. and Vitányi, P. (2008). *An Introduction to Kolmogorov Complexity and Its Applications*. Springer.

Li, W. and Godzik, A. (2006). Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659.

Ma, B., Tromp, J., and Li, M. (2002). PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445.

Mika, S. and Rost, B. (2003). Uniqueprot: creating representative protein sequence sets. *Nucleic Acids Research*, 31(13):3789–3791.

Morgulis, A., Gertz, E. M., Schäffer, A. A., and Agarwala, R. (2006). A fast and symmetric dust implementation to mask low-complexity dna sequences. *Journal of Computational Biology*, 13(5):1028–1040.

Nadeau, J. and Taylor, B. (1984). Lengths of chromosomal segments conserved since divergence of man and mouse. *Proceedings of the National Academy of Sciences of the United States of America*, 81:814–818.

Nalbantoglu, O. U., Russell, D. J., and Sayood, K. (2010). Data compression concepts and algorithms and their applications to bioinformatics. *Entropy (Basel, Switzerland)*, 12(1):34.

Ogurtsov, A. Y., Roytberg, M. A., Shabalina, S. A., and Kondrashov, A. S. (2002). Owen: aligning long collinear regions of genomes. *Bioinformatics*, 18(12):1703–1704.

Ovcharenko, I., Loots, G. G., amd Minmei Hou, B. M. G., Ma, J., Hardison, R. C., Stubbs, L., and Miller, W. (2005). Mulan: Multiple-sequence local alignment and visualization for studying function and evolution. *Genome Research*, 15(1):184–194.

Pevzner, P. and Tesler, G. (2003). Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. *Genome research*, 13:37–45.

Pinho, A. J., Pratas, D., and Garcia, S. P. (2012). Green: a tool for efficient compression of genome resequencing data. *Nucleic acids research*, 40(4):e27–e27.

Rissanen, J. and Langdon, G. (1979). Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162.

Schäffer, A. A., Aravind, L., Madden, T. L., Shavirin, S., Spouge, J. L., Wolf, Y. I., Koonin, E. V., and Altschul, S. F. (2001). Improving the accuracy of psi-blast protein database searches with composition-based statistics and other refinements. *Nucleic acids research*, 29(14):2994–3005.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197.

Suzek, B. E., Huang, H., McGarvey, P., Mazumder, R., and Wu, C. H. (2007). Uniref: comprehensive and non-redundant uniprot reference clusters. *Bioinformatics*, 23(10):1282–1288.

Vinar, T., Brejova, B., Song, G., and Siepel, A. C. (2010). Reconstructing Histories of Complex Gene Clusters on a Phylogeny. *Journal of Computational Biology*, 17(9):1267–1279. Early version appeared in RECOMB-CG 2009.

Višňovská, M., Nánási, M., Vinař, T., and Brejová, B. (2010). Estimating effective DNA database size via compression. In *Information Technologies, Applications and Theory (ITAT)*, volume 683 of *CEUR-WS*, pages 63–70, Smrekovica, Slovakia.

Višňovská, M., Vinař, T., and Brejová, B. (2013). DNA Sequence Segmentation Based on Local Similarity. In *Information Technologies, Applications and Theory (ITAT)*, Donovaly, Slovakia. Accepted.

Wang, C. and Zhang, D. (2011). A novel compression tool for efficient storage of genome resequencing data. *Nucleic acids research*, 39(7):e45–e45.

Waterman, M. S. and Eggert, M. (1987). A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. *Journal of molecular biology*, 197(4):723–728.

Waterman, M. S. and Vingron, M. (1994). Rapid and accurate estimates of statistical significance for sequence data base searches. *Proceedings of the National Academy of Sciences of the United States of America*, 91(11):4625–4628.

Waterston, R. H. et al. (2002). Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420(6915):520–522.

Wilbur, W. J. and Lipman, D. J. (1983). Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences of the United States of America*, 80(3):726–730.

Wootton, J. C. and Federhen, S. (1993). Statistics of local complexity in amino acid sequences and sequence databases. *Computers & chemistry*, 17(2):149–163.

Zhang, J. (2003). Evolution by gene duplication: an update. *Trends in Ecology and Evolution*, 18:292–298.

Zhang, L., Ma, B., Wang, L., and Xu, Y. (2003). Greedy method for inferring tandem duplication history. *Bioinformatics*, 19(12):1497–1504.