Comenius University in Bratislava
Faculty of mathematics, physics and informatics

Finding Conserved Gene Clusters in Duplicated Genomes

Master Thesis

2014 **Ing. Soňa Gibaštíková**

Comenius University in Bratislava
Faculty of mathematics, physics and informatics

Finding Conserved Gene Clusters in Duplicated Genomes

Master Thesis

Study Program:	Computer Science (Conversion Programme)
Branch of Study:	2508 Computer Science
Department:	Department of Computer Science
Supervisor:	Mgr. Bronislava Brejová, PhD.

Bratislava, 2014 **Ing. Soňa Gibaštíková**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Ing. Soňa Gibaštíková
Študijný program: informatika (konverzný program) (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Finding Conserved Gene Clusters in Duplicated Genomes
Hľadanie zachovaných zhlukov génov v duplikovaných genómoch

Cieľ: Cieľom práce je vyvinúť a implementovať efektívne algoritmy na porovnávanie poradia génov v genómoch, ktoré boli v priebehu evolúcie duplikované. Úlohou je hľadať skupiny génov, ktoré sa vyskytujú blízko seba v referenčnom neduplikovanom genóme, ale aj na dvoch miestach v duplikovanom genóme.

Vedúci: Mgr. Bronislava Brejová, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 23.10.2012

Dátum schválenia: 26.11.2012 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Ing. Soňa Gibaštíková
Study programme:	Computer Science (Conversion Programme) (Single degree study, master II. deg., full time form)
Field of Study:	9.2.1. Computer Science, Informatics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Finding Conserved Gene Clusters in Duplicated Genomes

Aim: The goal of the thesis is to design and implement efficient algorithms for comparing gene orders in genomes that underwent a whole genome duplication event in their evolutionary history. The task is to find groups of genes that are located near each other in a given reference non-duplicated genome, as well as in two regions of the duplicated genome.

Supervisor:	Mgr. Bronislava Brejová, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	doc. RNDr. Daniel Olejár, PhD.
Assigned:	23.10.2012
Approved:	26.11.2012

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

.....
Student

.....
Supervisor

First of all, I want to thank my supervisor Mgr. Broňa Brejová, PhD for her guidance, generous contribution of knowledge and experience, valuable comments and encouragement while working on the thesis. I would like to thank my parents and friends for their support during my studies.

Soňa Gibaštíková

Abstrakt

Cieľom našej práce je hľadanie génových regiónov v duplikovanom genóme. Génové regióny sú skupiny génov, ktoré sú zakonzervované v niekoľkých genómoch a ktoré plnia rovnakú alebo podobnú funkciu. V našej práci formálne definujeme problém hľadania takýchto génových regiónov v duplikovanom genóme a navrhneme algoritmus založený na dynamickom programovaní, ktorý rieši daný problém v polynomiálnom čase. Predstavíme niekoľko heuristických zlepšení, ktoré znížia výpočtový čas a model otestujeme na biologických a generovaných dátach.

Kľúčové slová: celogenómová duplikácia, duplikovaný genóm, génový región, aproximačný génový región

Abstract

The purpose of this thesis is finding conserved gene clusters in genomes that have underwent the whole genome duplication. Gene clusters are regions of genes conserved in several genomes that have virtually the same function and their identifications have many applications in genomics. In out thesis we formally define the problem of finding gene clusters in duplicated genomes and propose an algorithm based on dynamic programming solving the problem in polynomial time. We develop several heuristic improvements decreasing the computation time and we test our model on biological and generated data.

Key words: whole genome duplication, duplicated genome, gene cluster, approximate gene cluster, *Magnusiomyces*

Contents

List of Figures

List of Tables

Introduction

The aim of our work is to find gene clusters in genomes that have underwent the whole genome duplication. In order to redefine this biological problem to some informatics problem we have to found suitable representations for involved biological terms. For us a genome is a sequence of genes which are represented by integers and the same genes have the same number.

One of the typical bioinformatics problem is to compare several genomes from different species and find an interval in each genome such that these intervals contain the same or almost the same set of genes. The set of these intervals then forms a gene cluster. The purpose of the finding gene clusters is to reveal functional coupling between genes and to better understand evolutionary processes. The first gene cluster models required the gene cluster to be conserved i.e. the sets of genes from the intervals in the compared genomes have to be equal. These models are too strict, because genes can be inserted or deleted during evolution. The latest approaches can be characterized by the term approximate gene clusters because they allow occurrence of outsider genes inside the cluster. An outsider gene is a gene which does not occur in every interval from the gene cluster and allows to model gene deletion and insertion. After introducing the term gene cluster we will explain what duplicated genome means.

A whole genome duplication is an evolution event when the whole genome is doubled and each gene has two copies. We can imagine it as having two identical sequences instead of one sequence. A genome after the whole genome duplication is subject to fractionation which is a process in which one of two copies of a gene in the duplicated genome is often deleted. So current organism have some genes in two copies and some in single copy. If we apply problem of finding gene clusters to duplicated genome, than is better to find a pair of two intervals instead of one. Today exists only a model of conserved gene clusters in duplicated genome and it was introduced by Sankoff [SZ12] in his consolidation algorithm. He tries to reconstruct the fractionated duplicated genome to the state when all genes had two copies. He uses the other non-duplicated reference genome to help him identify the positions where some genes were deleted. To achieve it, he is finding a pair of two disjoint intervals in the duplicated genome and one interval in the non-duplicated genome such that union of genes from intervals in the duplicated genome is the same set of genes as set from the interval in the reference genome.

In our approach we extend the problem from Sankoff's algorithm to allow approximate gene clusters with outsiders. We define a problem and called it formally as the best interval location. In the problem we are aiming to localize a pair of two intervals in the duplicated genome which are the best match for some reference interval. The

degree of similarity of the intervals in the duplicated genome to the reference interval is expressed by score. We propose our scoring scheme which reflects the quality of the best interval location and can be easily calculated. The problem of the best interval location can be reduced to the mathematical problem of maximum sum of two disjoin intervals in a sequence of integers. We prove this reduction and design an algorithm solving the problem in linear time.

We want to identify as many as possible conserved gene regions in the duplicated genome, so we iterate through the whole reference genome and for each reference interval we determine its best interval location in the duplicated genome. As many found intervals are insignificant or overlapping with others, we have designed the mechanisms selecting only the significant reference intervals and their best interval locations.

We apply our algorithm to the genomes of yeasts from the genus *Magnusiomyces*. We form several pairs of genomes, always one with a hypothesis of whole genome duplication event and one sister non-duplicated genome. We identify the best interval locations and discuss the results. Our test organisms do not have fully assembled chromosomes and the size of their genome is quite small, therefore the basic algorithm runs fast. In order to achieve fast performance in more complex and bigger genomes, we develop several heuristic approaches which modify the original algorithm and decrease the computational time. We test the heuristics by applying them to real and generated data. The data generator is designed by ourselves and produces data with different features such as the length of chromosome and the level of similarity between genomes. It allows us to compare and contrast the strengths and weakness of each heuristic approach.

The thesis is organised as follows. We start with Chapter 1 where we define the basic terminology, summarize different known gene cluster models and different algorithms reconstructing duplicated genome. In the next chapter we propose our gene cluster model in the duplicated genome, define our scoring scheme, and reduce a biological problem of the best interval location to the problem of maximum sum of two disjoin intervals. In addition, we describe the dynamic programming algorithm solving the problem of maximum sum of two disjoint intervals and discuss its complexity. In the last part of this chapter we explain our filtering mechanisms selecting only the significant interval locations, and apply the algorithm to biological data. In the Chapter 3 we propose a few heuristic improvements which either calculate the dynamic programming separately for each chromosome or skip the dynamic programming if score of the previous best interval location is too low. We apply heuristics to real and generated data, and we discuss advantages and disadvantages of different heuristics approaches. In the conclusion we outline a few open problems for the future research.

Chapter 1

Biological background and overview of algorithms

1.1 Biological background

Organism's complete genetic information is encoded via double-stranded DNA molecule. This genetic package is called genome and is usually located in the cell nucleus. Genome consists of chromosomes and the number of chromosomes differs from organism to organism. For example, bacteria has only one circular chromosome and humans have two sets of chromosomes, each set containing 23 chromosomes. Each chromosome is a sequence of genes and gene is a stretch of DNA which has a function in the organism.

A polyploidization defines a mechanism for a duplication of whole genetic material of an organism i.e. a duplication of a complete set of chromosomes. Such an organism is called polyploid, but a more specific term may be used according to the number of sets of chromosomes. Usually a eukaryotic cell has two paired chromosomal sets, and such organisms (including humans) are called diploids. If a cell has three sets of chromosomes, it is a triploid, four sets a tetraploid, five sets a pentaploid etc.

Historically it was assumed that whole genome duplication (WGD) occurs only rarely and in terms of evolution it is undesirable (deleterious) process and does not have an important role in the formation of new species. Today WGD is understood as a dynamic process, which plays a crucial role in the evolutionary process and has a strong impact on species diversification, especially among plants, fungi and lower organisms. Studies suggest that at least one event of polyploidization has occurred in 30-70 % of the flowering plants [J.94] and in addition many of them underwent several independent rounds of whole genome duplication. WGD was observed in animals as well, especially in insects, but to a lesser extent even in mammals. Ohno [Sou74] proposed that the increased complexity and genome size of vertebrates has resulted from two rounds

of whole genome duplication in early vertebrate evolution, which provided genome redundancy for new evolutionary possibilities. Lower number of polyploids in animals is caused by their sexual reproduction, because polyploids have decreased ability to find an optimal sexual partner. On the contary, development of asexual reproduction (for example in plants) has increased their chances to survive WGD. Polyploidization also offers a possible explanation for the increase of genomic content in eukaryotic cells compared to prokaryotes (bacteria) [GL]. Polyploidization can have different causes at the cellular level, either asexual when a cell is divided abnormally, or sexual(which is considered to occur more frequently) when a polyploid cell is a result of a merging of two non-reduced sexual cells (gametes).

Corresponding duplicated genes in polyploids are called paralogous genes and they can be either maintained in multiple copies, lost or may be subject to subfunctionalization or neofunctionalization. If a gene is beneficial for individuals then remains, otherwise is deleted. (Theory : "Use it or lose it !" [LB97]). Subfunctionalization means a redistribution of ancestral functions among the new copies of the given gene. Much more interesting is the process of neofunctionalization, when one copy of the duplicated gene acquires a new beneficial function, which is subject to positive selection, and the other copy retains the original function.

After the polyploids are created, they undergo early stages of instability before becoming adapted and joining the evolutionary fray as efficient competitors of their diploid relatives. Adapted polyploids that avoid extinction enter an evolutionary trajectory of diploidization during which genome redundancy is reduced [L.05](See Fig. 1.1).

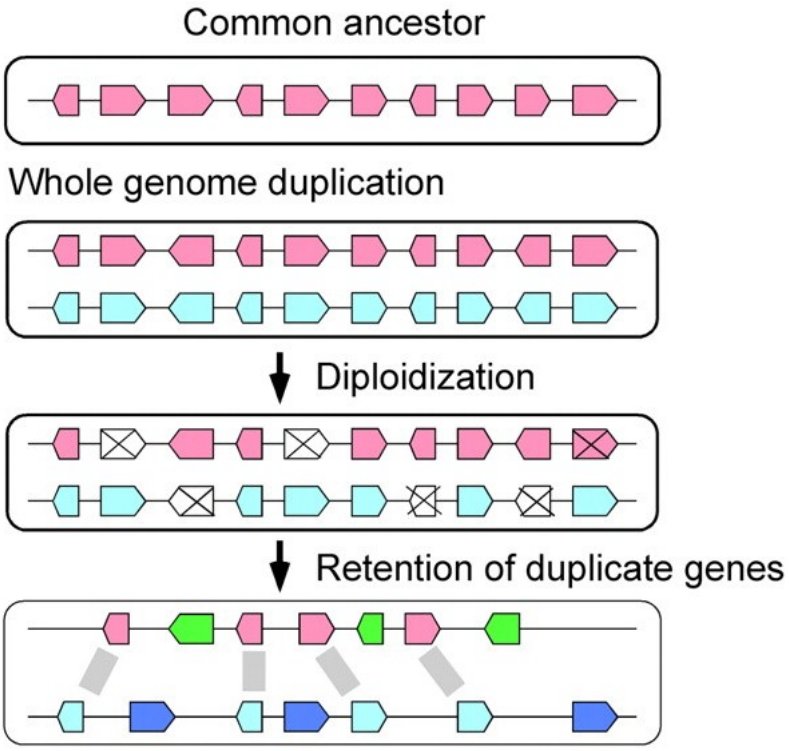


Figure 1.1: WGD and following diploidization process [HKC09]

Diploidization is a dynamic process of fractionation, shuffling, chromosomal rearrangements and divergence of duplicated portions of genome. The scale of the events ranged from loss of single genes (Fig. 1.1) to the loss of entire chromosomes. After a sufficiently long period the original polyploid is changed back to a diploid state. A current diploid whose ancestor was a polyploid is called a paleoploid or an "ancient" polyploid. The most common chromosomal rearrangements are depicted in Figure 1.2. The diploidization results in the large-scale reorganization of genomes and overall structure of the genome is significantly different compared to a non-duplicated ancestor. Immediately after the WGD the duplicated genome contained two identical copies of each original chromosome, but the order of genes and gene content has been eventually disrupted as the results of the diploidization.

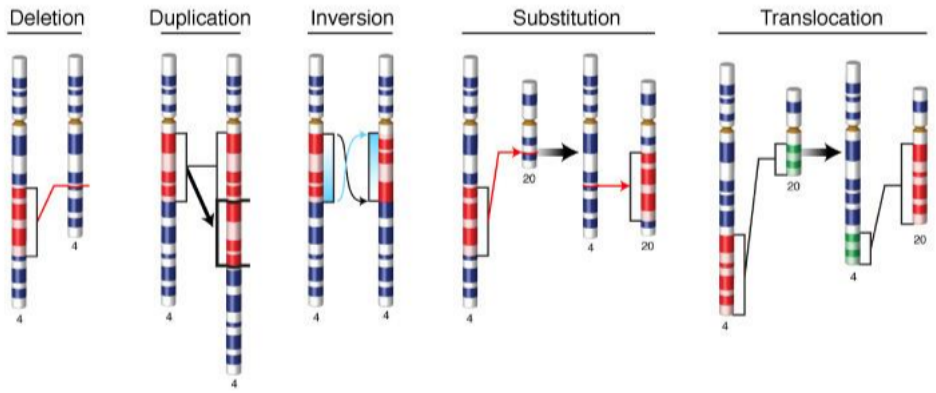


Figure 1.2: Chromosomal mutations [Glo].

Advantages of polyploids

Some plants with polyploidization history produce more biomass and have bigger fruit and seeds. Evident advantage compared to diploid organisms is their greater adaptability to new, unfavourable environmental conditions. An example of the rapid adaptation of polyploids to new and extreme niches has been provided by the study of the arctic flora [BBA+04]. It is a consequence of the presence of duplicated copies of the genes, which are not required to perform their original function, and therefore have a higher rate of mutation which leads to a more rapid adaptability to new environmental conditions. Moreover, polyploids can mask harmful (deleterious) gene mutations, if they contain non-mutated copy of the same gene.

1.2 Overview of algorithms

In this section we summarize the current state of research in two areas of bioinformatics related to our work. We start by giving mathematical definitions of basic biological terms needed in this section. Next we outline present models and algorithms for finding gene clusters. A gene cluster is a group of genes that code for the same function in the organism. Finally, two algorithms which try to reconstruct the genome after the process of diploidization to the moment immediately after the whole genome duplication, when all genes were doubled and correctly ordered, will be introduced at the end of this section. Until now, information about genome duplication was not incorporated in models of gene clusters with the exceptions of perfectly conserved clusters used for ancestral genome reconstruction by Sankoff [SZ12]. This gap in the analysis of gene clusters was a motivation for our work with the goal to improve the model of gene clusters for organisms with a whole genome duplication. The latest results in the area of finding gene clusters were starting points in the process of designing our gene cluster model specialized for duplicated genomes.

1.2.1 Basic definitions

The genetic information is stored in a strand of DNA. A gene in a biological definition is a sequence of this strand which encodes some protein that fulfils some function in the organism. However, for us a gene will be simply a marker representing a certain region of DNA. Genes that encode proteins with similar functions and sequence are grouped into one gene family. Typically, a gene family contain corresponding genes from related species, but we can also have several from the same family in one genome.

Definition 1 (Gene). *A single gene is represented as a non-negative integer. The genes from the same gene family share the same integer id.*

Definition 2 (Chromosome). *A chromosome is represented as a finite sequence of genes, i.e. a sequence over a finite set of genes ids.*

Due to the limitations of DNA sequencing technologies, we are often not able to reconstruct whole chromosomes, but only partial segments called contigs. In the thesis we define formally only the term chromosome and use it interchangeably for real chromosomes and for contigs. Genome G is in the biological sense a set of chromosomes $G = \{ch_1, ch_2, .., ch_n\}$. To simplify the problem formulation and algorithm description, the genome will be represented as one sequence of genes, in which chromosomes will be separated by some terminal character not used for representing genes.

Definition 3 (Genome). *Genome G is a linear sequence obtained as a concatenation of several chromosomal sequences joined by a terminal character #.*

The length of genome G is denoted $|G|$. The i-th position in genome G is denoted by $G[i](1 \leq i \leq |G|)$ and may correspond to a gene id or the terminal character.

Definition 4 (Interval). *Let G be a genome. Then $G[i, j]$ denotes the interval $(G[i], G[i+1], .., G[j])$ where i, j are positions in genome G, such that $1 \leq i \leq j \leq n$.*

The empty interval is a special interval with no elements and will be denoted as $G_\emptyset$.

1.2.2 Gene cluster models

A gene cluster is a genomic region, which is conserved in several genomes, which means that in each genome the cluster contains genes which encode the same or similar products. Genomes which initially have the same gene content and order become divergent because of genome rearrangements, gene duplications and deletions. Without selective pressure on these processes, gene content and order would be randomized over time. The fact that some gene clusters remain conserved during evolution indicates some functional relation of genes in these clusters. Examples are operons (group of genes that operate together) in prokaryotic genomes, or group of genes responsible for some metabolic function. Common ancestor of two species with a shared gene cluster was likely to contain the same cluster, so studying gene clusters is a method to track back ancestral genomes and compute evolutionary histories. Gene clusters have been applied in several areas of comparative genomics, including annotation of genomes, prediction of protein functions, localization of operons in prokaryotic organisms and discovery of horizontal gene transfer.

The main problem is that formal definition of gene clusters is not consistent and there exist many models and formulations. Most cluster definitions are constructive in the sense that they support an algorithm for finding such conserved regions. There are two basic aspects which gene cluster models have incorporated into their definition.

The first one is gene order. Do the identified genomic regions have to have genes in the same order or not? If yes, a cluster is usually defined as a string pattern over the alphabet of gene ids and the problem is stated as recognition of gene patterns in several genomes. Alternatively, a cluster can be characterized by a set of genes and two regions are then considered as copies of the same cluster if they contain an arbitrary permutation over this set of genes.

The second aspect of gene cluster definition is flexibility in the gene content. Do the gene clusters have to contain exactly the same set of genes? Some approaches

allow gene clusters to contain a certain number of "outsiders", i.e. genes that are not common to all copies of an identified gene cluster. The flexibility in the gene content is not arbitrary, and individual models introduce some threshold parameter to control the difference in the gene content between clusters.

Many methods define clusters in a very relaxed way and primarily do not put any condition on the length of a cluster, so algorithms for finding these gene clusters usually identify numerous intervals, but many of them are either too short or are overlapping each other. So a natural step is a modification of the basic model which selects only the the most significant or maximal clusters.

Every approach has some advantages and disadvantages. Over the time, small genome rearrangements modify even the gene clusters at a small scale, so the expectation that the gene content and order will be fully conserved is unrealistic. Methods which allow some mismatch in the gene content and order between gene clusters therefore better capture the biological reality. On the other side, approaches without this extra freedom can be more easily defined, and faster and simpler algorithm can be developed.

Conserved regions common to two genomes are the simplest scenario. However, most of the models are extended to search for gene clusters in more then two genomes, but computational complexity is usually higher. In the following text, we give the overview of existing cluster models and algorithms.

Conserved gene clusters

We start with the strictest model, which does not allow any deviation in gene order or content, and gene clusters have to be fully conserved [BCG07]. Such clusters are sometimes called co-linear clusters [Jah10]. This model assumes that genes in the genome do not repeat.

Definition 5. *(Conserved gene cluster) Let G_1 and G_2 be two different genomes. Then intervals $G_1[i,j], G_2[k,l]$ form a conserved gene cluster if interval $G_2[k,l]$ is equal to $G_1[i,j]$, i.e. $|G_1[i,j]| = |G_2[k,l]|$ and $G_1[i+z] = G_2[k+z] | z = 0,..,(j-i)$.*

In order to filter out overlapping clusters, the basic definition stated above is extended to the concept of a maximal conserved gene cluster, i.e a gene cluster which can not be extended on either side. These maximal conserved clusters are separated by breakpoints.

The algorithm for identifying these clusters works in $O(Kn)$ time, where K is the number of genomes and n is the length of one genome (assuming that genomes have the same size). The algorithm is very simple and requires some preprocessing. We

will describe the case when two genomes are compared, but the technique is easily transformable to more genomes. We select one genome as reference and for the other genome we construct a table containing position of each gene. The reference genome is then scanned, and for each gene pair $G[i], G[i+1]$ we check if they are consecutive also in the other genome. If not, we mark a breakpoint in the reference genome. Regions between breakpoints are then maximal conserved clusters.

Example 1. *Consider genomes:* $G_1 = (1,2,3,4,5,6), G_2 = (1,4,5,6,2,3)$.
The conserved gene clusters include all singletons: $(1),(2),(3),(4),(5),(6),(7)$
as well as some larger groups: $(1,2),(2,3),(3,4),(4,5),(3,4,5)$
The maximal conserved gene clusters are : $(1,2),(2,3),(3,4,5)$
Genome G_1 *after identifying breakpoints :* $G_1 = (1,2||3,4,5||2,3)$

While conserved gene clusters are very easily to detect, many biological significant clusters will be missed, because of tiny rearrangements in gene order or changes in gene content due to gene deletion or insertion.

This method does not have many practical applications, but was used in a phylogenetic study and for reconstruction of ancestral genomes [BS99].

Common intervals

The model of common intervals was first introduced by Uno and Yagiura [UY00] who defined the problem for two genomes, later it was extended to K genomes. It was the first attempt to relax gene cluster definition. This model requires the same gene content in all occurrences of the cluster, but the order of the genes can be arbitrary. Such gene clusters are called common intervals.

Definition 6. *Let* G_1, G_2 *be two genomes. Then two intervals* $G_1[a_1,b_1]$, $G_2[a_2,b_2]$ $(1 \leq a_1 \leq b_1 \leq |G_1|, 1 \leq a_2 \leq b_2 \leq |G_2|)$ *are common intervals if* $\{G_1[i] | a_1 \leq i \leq b_1\} = \{G_2[i] | a_2 \leq i \leq b_2\}$.

Uno and Yagioura proposed a basic simple algorithm for two permutations which works in $O(n^2)$ time, but the average time is $O(n)$ for two randomly chosen permutations. Input genomes are required to be permutation over the same set of gene ids and genes do not repeat in the permutations. The basic algorithm scans through all intervals in one genome and checks if the length of the interval between the rightmost and leftmost occurrence of some gene from this interval found in the second genome has the same length. If yes, between the rightmost and leftmost occurrence in the second genome are located only genes from the interval from the first genome and common interval was identified. During linear-time preprocessing, we store for all genes from

the first genome their position in the second genome. The same authors also developed algorithm working in $O(n+N)$ time, where N is the number of common intervals and $N \leq \binom{n}{2}$. The main technique is called reduced candidate (RC). The idea is to identify "wasteful" right end of an interval in the first genome from past search information and decrease the set of possible intervals which have to be checked in order to find common intervals. This algorithm is optimal because it runs proportionally to the size of input plus output. This algorithm was later improved to find all common intervals between K permutations by Heber and Stoye and it runs in time $O(Kn+N)$ [HS01]. They modify the problem to search for a set of so called "irreducible" intervals, which is a subset of common intervals such that common intervals can be easily reconstructed from irreducible intervals. However, the algorithm itself as well as used data structures are complex.

Similarly as in the previous strict model, there is an interest to filter out less significant clusters. In the work [LPW05] the authors introduced a method for find a strong intervals. Strong intervals are common intervals if they are not covered by other common intervals. We have one strong common interval of length $|S|$ and $|S|$ singleton intervals. Each bigger common strong interval contains at least two strong common intervals, so in total we have $O(n)$ strong common intervals. In the article, an algorithm with the complexity of $O(Kn)$ using a data structure called PQ tree is presented (K is the number of genomes).

Example 2. *Consider two genomes* $G_1 = (1,2,3,4,5,6,7)$ *and* $G_2 = (4,2,3,1,7,6,5)$ *which are the permutations over the same set.*
Common intervals :
$\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\}$ - *singletons*
$\{1,2,3\},\{1,2,3,4\},\{2,3\},\{2,3,4\},\{5,6\}\{5,6,7\},\{6,7\}$
$\{1,2,3,4,5,6,7\}$ - *whole set* S
Strong common intervals:
$\{2,3\},\{1,2,3,4\},\{5,6,7\},\{1,2,3,4,5,6,7\}$

A disadvantage of modelling a genome as a permutation is that this model does not allow occurrences of multiple genes. Schmidt and Stoye [ST04] introduced a model of common intervals based on genome which is represented as a string not as a permutation, so clusters with duplicated genes can be identified. Their algorithm solves the problem for two genomes in $O(n^2)$.

Even this variations does not describe gene clusters in a way which corresponds to their real biological structure, and the model is still too strict, because gene losses and insertions which can slightly change the gene content of gene clusters prevent these clusters to be identified by the model of common intervals.

r-window gene clusters

This is the simplest gene cluster model introducing a possibility of a flexibility in the gene content between the occurrences of gene cluster in different genomes. A cluster is defined as a pair of windows with size r which share at lest k genes, so there is at most $r - k$ outsider genes in every window.

Definition 7. *Consider genome G_1, G_2 ($|G_1| \leq |G_2|$) and constants r, k ($1 \leq r \leq |G_1|, 0 \leq k \leq r$) where r is the length of a window. A pair of two intervals $G_1[a_1, b_1]$, $G_2[a_2, b_2]$ ($1 \leq a_1 \leq b_1 \leq |G_1|, 1 \leq a_2 \leq b_2 \leq |G_2|$) is r-windows cluster if $|G_1[a_1, b_1]| = |G_2[a_2, b_2]| = r$ and $|\{G_1[i]|a_1 \leq i \leq b_1\} \cap \{G_2[i]|a_2 \leq i \leq b_2\}| \geq k$.*

If $k = r$ then the model is equivalent to the model of common intervals with length k. The method was used to detect duplicated blocks of genes in yeasts genomes [CGL03, FH01].

The essential part is to appropriately set the constant r and k. If k is too big, then many interesting gene clusters will be missed and if k is too small than many intervals will be detected by pure chance.

This model was not intensively studied in order to develop effective algorithms or heuristics. It is a basic model for testing the statistical significance of gene clusters. When a gene cluster is found it may show conservancy and similarity because of the pure randomness. Durand and Sankoff [dur03] created this model and they used r-windows sampling to test significance of the gene cluster in various scenarios. They expressed probability of finding a set of k genes in the window of length r and derived significance test for conserved clusters.

Max-gap clusters

Max-gap cluster is an another model [BBCR04] allowing a flexibility in the gene content. Genomes are considered as a permutations over some sets of genes, which are not required to be equal and do not contain duplicated genes. Gene cluster is modelled as a set of common genes for all occurrences of that gene cluster in the compared genomes. Each pair of adjacent mutual genes in the identified occurrence of gene cluster can not have more outsider genes between the mutual genes than a defined constant δ i.e. adjacent common genes are allowed to be separated by the gap that does not exceed a fixed threshold δ. In the original article these gene clusters were denoted as gene teams.

Definition 8. *Assume genomes G_1, G_2 and a fixed constant δ such that $\delta \geq 0$. A pair of intervals $G_1[a_1, b_1]$, $G_2[a_2, b_2]$ ($1 \leq a_1 \leq b_1 \leq |G_1|, 1 \leq a_2 \leq b_2 \leq |G_2|$) is δ*

max-gap cluster if $C = \{G_1[i]|a_1 \leq i \leq b_1\} \cap \{G_2[i]|a_2 \leq i \leq b_2\}$ *is a set of common genes,* $|G_1[x,y]| \leq \delta$ ($\forall$ *consecutive genes x,y* $\in C$ *in the interval* $G_1[a_1, b_1]$) *and the same condition holds for the all common genes in the interval* $G_2[a_2, b_2]$

If $\delta = 0$ then this model equals to the previous model of common intervals. An algorithm based on the divide and conquer technique is running in $O(n \log^2 n)$ time where n is the length of genomes. An idea behind the algorithm is based on the maximal δ max-gap clusters in terms of inclusion. We start with the whole genome and if it is not a δ max-gap cluster then the genome can be divided into two smaller intervals and the problem is solved recursively until we finally get some maximal δ max-gap cluster.

Example 3. ** denotes genes that are not in the set of common genes.*
Assume having $\delta = 1$ *and these two genomes :*
G_1 = *(1 * 2 * 3 4 * * 5 6 7 8 9)*
G_2 = *(* 3 * 1 4 * 2 5 6 7 * 9 8)*
δ max-gap clusters together with lonely genes (singletons):
$\{1\}, \{2\}, \{3\}, .., \{9\}, \{1,2,3,4\}, \{5,6\}, \{5,6,7\}, \{5,6,7,8,9\}, \{6,7\}, \{6,7,8,9\}, \{7,8,9\}, \{8,9\}$
maximal δ max-gap clusters : $\{1,2,3,4\}, \{5,6,7,8,9\}$.

Bergeron, He and Goldwasser [PBR+05, HG05] later expanded this model for genomes represented as a string, so duplicated genes can be incorporated into clusters. They implemented a modified divide and conquer algorithm to solve a newly defined problem in time $O(n + m)$, where n, m are the number of common genes in both genomes.

A disadvantage of this approach is that from the biological point of view we do not have evidence that more small gaps are more likely than few bigger gaps.

Approximate gene clusters

As was mentioned in the beginning of the section, micro rearrangements break the gene order in the gene cluster and gene insertions and deletions violate the gene content of the cluster by outsider genes. We describe now a model which has the highest generality and allows the irregularities in the gene content and order between the intervals of gene cluster, and therefore enables to detect gene clusters with different diverse conservation patterns [K.11]. Genome is represented as a string and cluster as a set of genes. The goal is to find such two intervals in the compared genomes for which the symmetrical difference between the sets of genes from these intervals (occurrences of the gene cluster) is bounded by some threshold parameter δ.

Definition 9. *Consider two genomes G_1,G_2 and threshold parameter δ such that $\delta \geq 0$. A pair of intervals $G_1[a_1, b_1]$, $G_2[a_2, b_2]$ $(1 \leq a_1 \leq b_1 \leq |G_1|, 1 \leq a_2 \leq b_2 \leq |G_2|)$ is δ-approximate gene cluster if $C_1 = \{G_1[i] | a_1 \leq i \leq b_1\}$, $C_2 = \{G_2[i] | a_2 \leq i \leq b_2\}$ are sets of genes from the intervals $G_1[a_1, b_1], G_2[a_2, b_2]$ and $|C_1 \backslash C_2| + |C_2 \backslash C_1| \leq \delta$.*

The max-gap cluster model deals with gene losses indirectly. The gene which is not presented in both intervals is considered as a outsider and is part of the gap between common genes. As a consequence, the set of genes representing gene cluster is reduced to a minimal consensus, i.e. genes which are mutual for both intervals. On the other side, the approximate gene cluster is not a minimal consensus but a set of genes which is optimized in the sense that symmetric distance between genes in the intervals is minimized [K.11].

The idea of defining a flexibility in the gene content between the intervals of the cluster as a set distance was introduced in [CDH$^+$06]. Later, a median gene cluster as the best approximate gene cluster for K genomes was defined and computed in [BJMS09]. The latest results from this topic can be found in the dissertation thesis of Katharina Jahn [Jah10]. The algorithm finding approximate gene clusters is based on Stoye and Schimdt algorithm developed for common intervals, and it has time complexity $O(K^2n^2(\delta + 1)^2)$ where K is the number of compared genomes. We do not aim to find all δ-approximate gene clusters, but only the maximal ones which are called δ-optimal approximate clusters, and therefore we can achieve polynomial time complexity. An interval of the optimal approximate gene cluster does not have at the positions directly to the right and to the left genes which are contained inside the interval.

Example 4. *Assume genomes G_1, G_2 and threshold parameter $\delta = 3$.*
G_1 = (6 2 1 4 3 9 4 5 3 8 12 2 10 6 3)
G_2 = (11 14 7 12 7 1 5 3 8 13 4 7 1 12 15 3 9 1 15 8 1 5)
If $G_1[3, 10]$ is an interval in the genome G_1 and
$G_2[6, 9], G_2[6, 11], G_2[13, 23], G_2[15, 23], G_2[16, 19], G_2[21, 23]$ are intervals in the genome G_2, then each pair of $G_1[3, 10]$ and any interval from G_2 forms a δ-optimal approximate gene cluster.

1.2.3 Algorithms reconstructing duplicated genome

In this subsection we describe two algorithms designed especially for duplicated genomes. As was already mentioned in the biological background, a duplicated genome undergoes chromosomal rearranges and the process of fractionation, after the whole genome duplication. The goal is to reconstruct a current genome of an organism with WGD

to the state immediately after the whole genome duplication where gene order was not disrupted and each gene has two copies. Algorithms differ from each other by the type of mutation which effect on the genome they try to reconstruct.

Genome halving problem

A problem was introduced by El-Mabrouk and Sankoff [EMNS98]. Assuming that a genome is duplicated and then rearranged over time, is possible to reconstruct ancestral genome from the current gene order? The reconstructed genome should have a minimal distance, that means a minimal number of rearrangements to transform the original genome to the genome we are observing today. In the original article [EMNS98] authors defined a problem of genome halving only as a number of translocations (see Fig. 1.2) needed to replace n chromosomes of current genome with n new chromosomes where is exactly $n/2$ pairs of identical chromosomes. Later, the translocation was replaced by more universal JCD (double cut and join) operation, which can model all chromosomal rearrangements [WS09]. The other model is a guided genome halving where the genome reconstruction is guided with the help of one or more reference genomes [ZZAS08].

A consolidation algorithm

Another important mechanism causing gene order disruption after the event of whole genome duplication is fractionation, when one copy of duplicated gene is deleted. If the evolutionary distance is estimated on the number of chromosomal rearrangementes, the comparison between duplicated genome and its non duplicated sister genome shows significant overestimation of the chromosomal rearrangements. When a number of the adjacent duplicated gene pairs lose some genes from one chromosome and the other pairs lose genes from the other chromosome, then all these places are considered as the rearrangement break points (Fig. 1.3).

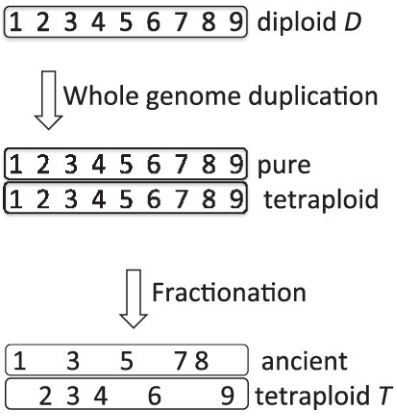


Figure 1.3: Fractionation leading to different adjacencies in current diploid and its ancestor [SZ12]

A problem of an exaggerate amount of chromosomal rearrangements because of the diploidization was formulated by Sankoff and Zheng [SZ12]. Their algorithm reconstructing the places of diploidization is called a consolidation algorithm and has $O(n^2)$ time complexity. Subsequently, the algorithm was improved to run in linear time and to deal with genomes with higher degree of polyploidization [JZKS12]. The consolidation algorithm is based on the identification of fractionation intervals, regions in duplicated genome and its sister non duplicated reference genome. Two intervals in duplicated genome correspond to a single region in sister genome if their union of genes is a set equal to the set of genes from the reference interval.

These fractionation intervals correspond to conserved gene clusters in the duplicated genome, although Sankoff does not formulate his problem as a problem of finding gene clusters. In our work we take these fractionation regions and modify them to resemble the approximate gene clusters.

Chapter 2

Model and algorithms

2.1 Problem definition

In this section, we formally define the problem of our interest and describe algorithms for solving it. As it was mentioned previously,the main goal is to find a pair of conserved gene clusters in the duplicated genome for some specific interval in the reference genome.
Before defining our problem, it is necessary to introduce additional terminology.

Definition 10 (Double location). *A double location is a pair of two disjoint intervals in genome G.*

If both of the intervals from the double location are non-empty, it will be denoted as $G[a_1, b_1][a_2, b_2]$, where interval $G[a_1, b_1]$ occurs to the left of $G[a_2, b_2]$ in genome G. If one of the intervals from the pair is empty, then the double location is equivalent to the non-empty interval. If both intervals are empty, the double location is an empty interval as well.

Definition 11 (Interval location). *An interval location for an interval $R[i, j]$ from the reference genome R is defined as a pair $(R[i, j], D[a_1, b_1][a_2, b_2])$ where $D[a_1, b_1][a_2, b_2]$ is a double location in the duplicated genome D.*

Definition 12 (Scoring function). *A scoring function S is a function which assigns a real valued score to each interval location $(R[i, j], D[a_1, b_1][a_2, b_2])$.*

Definition 13 (Best interval location problem). *The goal of the best interval location problem is to find the interval location $L_{i,j} = (R[i, j], D[a_1, b_1][a_2, b_2])$ in the duplicate genome D for a fixed reference interval $R[i, j]$ such that its score is maximal.*

Definition 14 (All locations problem). *For each interval in the reference genome R find its best location in the duplicated genome D. Solution of this problem P ($SOL(P)$) is the set of the best interval locations $L_{i,j}$.*
$SOL(P) = \{L_{i,j} : \forall i, j$ *such that* $R[i, j]$ *is an interval in* $R\ \}$

A disadvantage of the All location problem is that the final set of solutions is too large and many of its elements are not meaningful. Many of the reference intervals $R[i, j]$ are too short to be considered as significant, many have very low score and many intervals are overlapping each other. Therefore these intervals have to be filtered under some set of conditions, which will be denoted as C_{filt} and described later (Section 2.7).

Definition 15 (Significant solution). *A significant solution to the above defined problem P is the set of the best interval locations $L_{i,j}$ from $SOL(P)$ which satisfy each condition defined in C_{filt}. This set is denoted as $\overline{SOL}(P)$.*

An unanswered question is, how score of each interval location will be evaluated. The problem of defining scoring function S is outlined in the next section.

2.2 Scoring system

Defining a relevant scoring function can be complicated because a score assigned to each interval location should express its biological quality and should allow us to differentiate the bad interval locations from the good ones. Moreover, score calculation has to be efficient in terms of time complexity.

Before giving the formula for scoring function itself, we define gene content of an interval and discuss a few key ideas leading to our scoring scheme.

Definition 16 (Gene content). *Let $G[i, j]$ be an interval in genome G. Then the gene content of this interval $GC(G[i, j])$ is the set of unique genes from this interval:*
$GC(G[i, j]) = \{G[k] | i \leq k \leq j\}$.
The gene content of the whole genome G is defined in the similar way and contains all genes from this genome:
$GC(G) = \{G[i] | 1 \leq i \leq |G|\}$.

What is the best interval location we can get for some reference interval? In the ideal case, we would expect to find two disjoint intervals in the duplicated genome which are both equal to the original interval in the reference genome. Assumption that an order of the paralogous genes in the double location and the order of genes in the reference interval is identical, is too rigorous. Both Sankoff in his interval

reconstruction algorithm [SZ12] and Jahn [K.11] in her scoring system for gene clusters detections disregard gene order inside intervals and emphasize only their gene content. To translate it to our problem, gene content of the reference interval and the best interval location should equal in optimal case, i.e $GC(R[i,j]) = GC(D[a_1,b_1][a_2,b_2])$. Note that here we compare gene content of $R[i,j]$ with the union of genes in $D[a_1,b_1]$ and $D[a_2,b_2]$. Thus a gene from $R[i,j]$ can have one or more copies in the double location.

Longer intervals $R[i,j]$ conserved in the duplicated genome are more significant, because they are less likely a result of chance than in the case of short reference intervals. Over time, every genome undergoes several minor or bigger mutations, so perfect content conservation is observed only at the early stages after the WGD. Therefore we allow some differences in the gene content thus extended model of [SZ12] who seek perfect gene content conservation. Divergence in the gene content is due to extra genes (outsiders) either in the reference interval or in the double location. Such an outsider can be result of gene deletion in the other genome, insertion of the outsider itself or rearrangements which moved the outsider to a new position in one of the two genomes.

In the scoring system used by Jahn [K.11], every extra gene in the gene content of the reference interval or localized gene cluster decreases the score equally. Straightforward adaptation of this scoring system to our problem of the best interval location leads to several problems. Comparison of gene content of the double location to the gene content of the reference interval requires a calculation of the union of the gene sets from the double location intervals and a calculation of symmetric difference between gene sets from the double location and reference interval. In this scenario, it is difficult to devise an algorithm which finds the best interval location efficiently. Another problem is that imperfections in the interval location can be observable even if the gene content of the reference interval and double location are equal and Jahn's scoring system would not penalize it. For example, if one copy of a duplicated gene in the duplicated genome is outside the identified double location and the other inside, then the copy inside would cause that the gene content does not change, and the score of this location is the same as a score of a double location where both copies are included within the location. Naturally we would like to give some penalty for this discrepancy and thus distinguish between these two interval locations. We propose our own scoring scheme which is not based on differences between sets of genes, but is based on an additive penalization assigned to each gene causing divergence from the perfect interval location. Later we show that computation of this scoring function is very simple and can be done in constant time.

From the biological point of view, proposed scoring system does not give optimal

score all the time. One example is, when some gene has two or more copies in the reference genome as a consequence of an older WGD event or some mutation. If a reference interval does not contain all copies of such a gene, then how should be penalized copies of the gene which are outside the identified best interval location in the duplicated genome? Some of them could originate from the copies which are not inside the reference interval. Our scheme penalizes them because it recognizes them as copies of the gene from the reference interval and in an ideal interval location they would be localized inside. On the other side, Jahn's scoring system would not penalize them as long as there is at least one copy inside the double location, so gene content is matched to the gene content of the reference interval. An other example happens when some outsider gene inside the double location is excessively copied so we have many copies of the same extra gene inside the double location. Our scoring scheme penalizes each copy separately, but Jahn's scheme would penalize only one, because all of the copies increase the gene content set by the same gene.

Definition 17 (Scoring function formula). *Let $R[i,j]$ be a reference interval and $L = (R[i,j], D[a_1,b_1][a_2,b_2])$ an interval location of this interval in the duplicated genome D. Scoring function S assigns the following value to the interval location L:*

$S(L) = |R[i,j]| - \alpha S_{IN} - \beta S_{OUT} - \gamma S_{MIS}$, *where*

$$S_{IN} = \sum_{k \in [a_1,b_1] \cup [a_2,b_2]} (1 - I[k]) \text{ and } I[k] = \begin{cases} 1 \iff D[k] \in GC(R[i,j]) \\ 0 \text{ else} \end{cases}$$

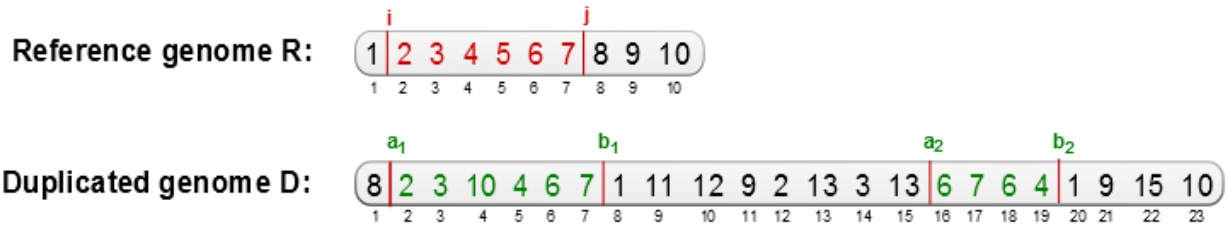
$$S_{OUT} = \sum_{k \in [1,a_1-1] \cup [b_1+1,a_2-1] \cup [b_2+1,|D|]} I[k]$$

$$S_{MIS} = \sum_{k \in [i,j]} J[k] \text{ and } J[k] = \begin{cases} 1 \iff R[k] \notin GC(D) \\ 0 \text{ else} \end{cases}$$

Constants α, β, γ are parameters of the scoring function.

The first term of the scoring system is the length of the reference interval which determines the probability that the double location for a given interval would be conserved by chance. Term $-\alpha|S_{IN}|$ decreases the score by the amount equivalent to the number of outsider genes in the double location $D[a_1,b_1][a_2,b_2]$, i.e. the number of genes which are not found in the gene content of the reference interval $GC(R[i,j])$. Term $-\beta|S_{OUT}|$ decreases the score for each gene from the reference interval occurring outside the identified double location in D, i.e in the intervals $D[1,a_1-1], D[b_1+1,a_2-1], D[b_2+1,|D|]$. The last term $-\gamma|S_{MIS}|$ deals with the genes from the reference interval which are not found in the whole duplicated genome, so are considered as missing.

Example 5. *Consider reference interval* $R[2,7]$ *and its interval location* $L = (R[2,7], D[2,7][16,19])$



For the sake of score calculation for L*, the following values were computed:*
$|S_{MIS}| = |\{5\}| = 1$
$|S_{IN}| = |\{10\}| = 1$
$|S_{OUT}| = |\{2,3\}| = 2$
If parameters $\alpha = \beta = \gamma = 1$ *then a score of the interval location* L *equals to :*
$S(L) = |R[i,j]| - \alpha S_{IN} - \beta S_{OUT} - \gamma S_{MIS} = 6 - 1 - 1 - 2 = 2$
Note that for these values of α, β, γ*,* L *is the best interval location* $L_{2,7}$*.*

2.3 Simplified problem formulation

In this section we introduce a mathematical formulation for the maximum sum of two disjoint intervals problem and show the equivalence of this problem with the best interval location problem.

Definition 18 (Maximum sum of two disjoint intervals problem - MS2DIP). *Given a sequence* S *of integers* $s_1, s_2, .., s_n$ *find at most two disjoint intervals* $S[a_1, b_1]$ *and* $S[a_2, b_2]$ *such that they maximize the sum* $MS2DIP(S[a_1, b_1][a_2, b_2]) = \sum_{i=a_1}^{b_1} s_i + \sum_{j=a_2}^{b_2} s_j$*.*

Solution to this problem consists of the maximum sum as well as the identified intervals $(S[a_1, b_1], S[a_2, b_2])$. If all integers in the sequence are positive, then the solution is trivially the whole sequence. In contrast, if all integers are negative then the solution is the empty interval and the sum is set to zero.

In order to show the equivalence of this problem with the proposed definition of the best interval location, the genome as a sequence of gene ids has to be transformed to a sequence of integers, where each integer should characterize a gene in terms of its quality for some reference interval. This integer will correspond to the partial score assigned to each gene and the score of the i-th gene in the duplicated genome will be denoted as $c[i]$.

Lemma 1. *Let* $R[i,j]$ *be a reference interval,* $L_{i,j} = (R[i,j], D[a_1, b_1][a_2, b_2])$ *its best interval location in the duplicated genome and* S *scoring function with parameters* α, β, γ*.*

Partial score $c[k]$ is assigned to each gene in the duplicated genome D so that $c[k] = \beta$ iff $D[k] \in GC(R[i,j])$ or $c[i] = -\alpha$ otherwise. If the newly obtained sequence from duplicated genome D is denoted D', then solution $L' = (D'[a_1, b_1], D'[a_2, b_2])$ of $MS2DIP$ is equal to the solution of the best interval location $L_{i,j}$.

To prove this lemma, we have to show the equivalence between the score of $L_{i,j}$ and the score of L'. The next lemma gives the calculation of the score L' and it claims that such computed value is equal to the score of $S(L_{i,j})$. The statement formulated in the lemma is even stronger and claims, that for the reference interval $R[i,j]$ is the score of any double location $D[a_1, b_1][a_2, b_2]$ in the duplicated genome equals to the sum of corresponding intervals $D'[a_1, b_1], D'[a_2, b_2]$ in the sequence D'. We will denote as $occ(R[i])$ the number of occurrences of reference gene $R[i]$ in the duplicated genome D. Further, let $occ(R[i,j]) = \sum_{k=i}^{j} occ(R[k])$ be the number of occurrences of genes from the whole reference interval in the duplicated genome. S_{MIS} is the number of missing genes from the reference interval as explained in the definition of the scoring function (Definition 8).

Lemma 2. *Let $R[i,j]$ be a reference interval and $L' = (D'[a_1, b_1], D'[a_2, b_2])$ a pair of disjoint intervals in the sequence D'. Score of L' calculated as $|R[i,j]| - \gamma S_{MIS} - \beta occ(R[i,j]) + MS2DIP(D'[a_1, b_1][a_2, b_2])$ has the same value as the score of interval location $(R[i,j], D[a_1, b_1][a_2, b_2])$ given by scoring function S with parameters α, β, γ.*

Proof. Terms $|R_{i,j}|$ and γS_{MIS} are the same as in the scoring system for the interval location. To prove the identity of scores, negative value of the $-\alpha S_{IN}$ and $-\beta S_{OUT}$ in the formula for scoring function S has to be equal to the value of $-\beta occ(R[i,j]) + MS2DIP(D'[a_1, b_1][a_2, b_2])$. All integers in the intervals $D'[a_1, b_1], D'[a_2, b_2]$ have either value $-\alpha$ or β, so the maximum possible score is the sum of the lengths of these intervals multiplied by constant β , i.e. $|(D'[a_1, b_1]| + |D'[a_2, b_2]|)\beta$. The score is lowered by the the number of $-\alpha$ integers inside the intervals $D'[a_1, b_1], D'[a_2, b_2]$ and the number of negative α integers corresponds to the cardinality of set S_{IN}. Moreover, the difference between $\beta occ(R[i,j])$ and the number of positive β integers in the intervals $D'[a_1, b_1]D'[a_2, b_2]$ agrees with the cardinality of the set S_{OUT}. This finishes the proof about the score equivalence. □

Lemma 1 follows from Lemma 2 because terms $|R[i,j]| - \gamma S_{MIS} - \beta occ(R[i,j])$ depend only on reference interval $R[i,j]$, but not on $[a_1, b_1], [a_2, b_2]$. Therefore score of the interval location is maximized by interval pair maximizing $MS2DIP$.

To sum it up, if genes in the duplicated genome are assigned the partial score $-\alpha$ or β according to their membership in the gene content of the reference interval $R[i,j]$, the problem of the best interval location can be reduced to the problem of the maximal

sum of two disjoint intervals in the sequence obtained by replacing gene ids by their partial scores. Identified intervals $D'[a_1, b_1]$,$D'[a_2, b_2]$ as a solution to the $MS2DIP$, correspond to the best interval location, and the score of the best interval location $L_{i,j}$ can be computed from the solution of the $MS2DIP$ by the formula from lemma 2. We will call intervals $D'[a_1, b_1]$,$D'[a_2, b_2]$ conserved intervals, because they represent conserved sequences of reference genes in the duplicated genome. To be strict with formalism, terminal characters # in the duplicated genome have score $-\infty$ in the sequence D'.

2.4 Dynamic programming

In this section we describe an efficient algorithm based on dynamic programming for finding the maximum sum of two disjoint intervals. The idea of the algorithm is to compute maximum sum of two intervals gradually for longer and longer prefixes of the input sequence. Conserved intervals from $MS2DIP$ will be denoted as $C_1 = D'[a_1, b_1]$ and $C_2 = D'[a_2, b_2]$. One or even both conserved intervals in $MS2DIP$ can be empty, but these special cases will be handled by the algorithm without any special calculations. Parts of the sequence not included in intervals C_1, C_2 form at most three intervals (see Figure 2.1). We will call them residual intervals and denote them by R_1, R_2, R_3, where: $R_1 = D'[1, a_1 - 1]$, $R_2 = D'[b_1 + 1, a_2 - 1]$, $R_3 = D'[a_2 + 1, n]$, where n is the length of D'.

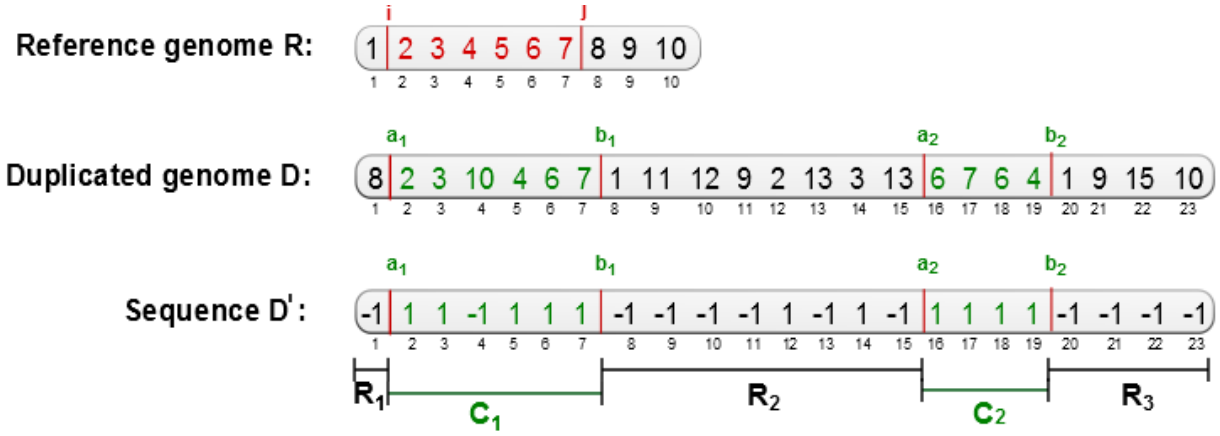


Figure 2.1: Example of an reference interval, intervals in the duplicated genome and in the sequence D' if $\alpha = -1$, $\beta = 1$.

Recurrence equations for dynamic programming are slightly different for conserved

intervals and residual intervals.

Conserved intervals:

$$C_i[j] = \max \begin{cases} C_i[j-1] + c[j] \\ R_i[j] \end{cases}$$

$$C_0[j] = 0 \; j = 1, .., n$$

Residual intervals:

$$R_i[j] = \max \begin{cases} R_i[j-1] \\ C_{i-1}[j] \end{cases}$$

$$R_1[j] = 0 \; j = 1, .., n$$

C referrers to conserved and R to residual interval. Index i references corresponding interval and j position in the sequence. Two subproblems for dynamic programming are defined by the above recurrences. The first subproblem $C_i[j]$ is the maximum sum of i conserved intervals in the subsequence $D'[1, j]$ if the i-th conserved interval ends at the position j. The second subproblem $R_i[j]$ is defined in a similar way as the maximum sum of $i - 1$ conserved intervals in the subsequence $D'[1, j]$ if the i-th residual interval ends an the position j.

To compute value $C_i[j]$ recursively from the values of the already solved subproblems, the algorithm chooses the better alternative from two options. We can enlarge the i-th conserved interval by adding the integer an the j-th position ($c[j]$) to the optimal value calculated for sequence $D'[1, j-1]$ which is stored in $C_i[j-1]$. The other option is to start the i-th conserved interval at the position j and take the best score reached in the previous residual interval as its initial score. Perhaps we would expect to take the value stored in the $R_i[j-1]$ and add the value $c[j]$, but because this $i - th$ interval can be empty, we consider the interval as empty at the beginning. So we do not add integer $c[j]$ and the initial score is equal to the score $R_i[j]$ from the previous residual interval ending at position j. The idea behind the computing score for $i - th$ residual interval is the same. Again we can extend the optimal solution for $i - th$ residual interval from the subsequence $D'[1, j-1]$ just by copying this value to $R_i[j]$. We do not add any integer value stored in the position j, because now we do not extend conserved interval. The other option is just to start the i-th residual interval at the j-th position, so we take the best score we reached by previous conserved interval C_{i-1}. Because residual interval can again be empty, we take the score $C_{i-1}[j]$ of the conserved interval ending at the position j.

Dynamic programming in our implementation does not calculate two separate matrices for C and R, but merges them together into one final matrix F. The goal of the

problem of the maximum sum of two disjoint interval is to divide the sequence into at most five intervals, starting and ending with residual interval. Conserved intervals and residual intervals alternate in the sequence, and each has its matching row in matrix F. Values in F are calculated gradually from left top to right bottom and value $F[i,j]$ can be evaluated if values from $F[i-1,j]$ and $F[i,j-1]$ are known. The final score of the maximum sum is stored in the bottom-right cell. The following equations are simply merged from the previously defined recursive equations.

Recurrence equations:

$$F_i[j] = \max \begin{cases} \begin{cases} F_i[j-1] & \text{if i is odd, i.e. it corresponds to a residual interval} \\ F_i[j-1] + c[j] & \text{if i is even i.e it corresponds to a conserved interval} \end{cases} \\ F_{i-1}[j] \end{cases}$$

$$F_0[j] = 0 \ j = 1, .., n$$

$$F_i[0] = 0 \ \forall i$$

An identification of the positions of the conserved intervals is equivalent to the identification of a trajectory of choices in the matrix F which led to the final score. During evaluation of the matrix F, pointers showing from which cell the value of $F_i[j]$ was calculated are stored in another matrix T. At the end, the algorithm uses trace-back to recover the positions of the conserved and residual intervals from matrix T. We start at the bottom-right cell and follow the pointers until we get to the left-top cell. There are only two possible moves: up, if a value in a cell was derived from the above cell or left if the value was calculated using the left cell. Detections of the ends points of the intervals is pretty simple. We follow pointers to the left until we reach pointer to the upper row, then the beginning of the currently identified interval is the position next to the cell with up pointer from the right and the end of the interval is the position where we started our back-trace in the current row. If in the starting cell for trace back in the current row contains a pointer to the upper row, the interval is empty.

We will illustrate the algorithm on the following example.

Example 6. *Let consider reference interval* $R[1,4] = \{1,2,3,4\}$.
Our problem is to find the best interval location for this interval in the duplicated genome $D = (5,1,3,7,4,2,6,8,3,2,9)$ *by the algorithm presented earlier.*
Let us score the genes in the duplicated genome with $\alpha = 1$ *and* $\beta = 1$*, obtaining a sequence of scores* $D' = (-1,1,1,-1,1,1,-1,-1,1,1,-1)$
The next step is computation of matrices F *and* T.

Matrix F:

D		*5*	*1*	*3*	*7*	*4*	*2*	*6*	*8*	*3*	*2*	*9*
D'		*-1*	*1*	*1*	*-1*	*1*	*1*	*-1*	*-1*	*1*	*1*	*-1*
		0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
$\mathbf{R_1}$	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
$\mathbf{C_1}$	*0*	*0*	*1*	*2*	*1*	*2*	*3*	*2*	*1*	*2*	*3*	*2*
$\mathbf{R_2}$	*0*	*0*	*1*	*2*	*2*	*2*	*3*	*3*	*3*	*3*	*3*	*3*
$\mathbf{C_2}$	*0*	*0*	*1*	*2*	*2*	*3*	*4*	*3*	*3*	*4*	*5*	*4*
$\mathbf{R_3}$	*0*	*0*	*1*	*2*	*2*	*3*	*4*	*4*	*4*	*4*	*5*	*5*

Matrix T:

D	*5*	*1*	*3*	*7*	*4*	*2*	*6*	*8*	*3*	*2*	*9*
$\mathbf{R_1}$	$\downarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$
$\mathbf{C_1}$	$\downarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$
$\mathbf{R_2}$	$\downarrow$	$\downarrow$	$\downarrow$	$\rightarrow$	$\downarrow \rightarrow$	$\downarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\downarrow \rightarrow$	$\rightarrow$
$\mathbf{C_2}$	$\downarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow$	$\rightarrow$	$\rightarrow$	$\downarrow \rightarrow$	$\downarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$
$\mathbf{R_3}$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow \rightarrow$	$\downarrow$	$\downarrow$	$\rightarrow$	$\rightarrow$	$\downarrow \rightarrow$	$\downarrow$	$\rightarrow$

Using matrix T we can identify the best interval location of the reference interval $L_{1,4} = (R[1,4], D[2,6][9,10]) = ((1,2,3,4),(1,3,7,4,2)(3,2))$

Solution of $MS2DIP$ is the sum of the identified conserved intervals and is stored in the bottom-right cell in matrix F. Conserved intervals are equivalent with the best interval location and lemma 2 shows how to calculate the score of the best interval location from the solution of the $MS2DIP$.

Example 7. *Calculate a score of the best interval location identified in Example 2.* $L_{i,j} = (R[1,4], D[2,6][9,10])$, $MS2DIP(D') = 5$ *and* $\alpha = \beta = \gamma = 1$
$S_{MIS} = 0$
$occ(R[i,j]) = 6$
$|R[i,j]| = 4$
Now the score of the best interval location $L_{1,4}$ *equals :*
$S(L_{1,4}) = |R[i,j]| - \gamma S_{MIS} - \beta occ(R[i,j]) + MS2DIP(D'[2,6][9,10]) = 4 - 0 - 6 + 5 = 3$

All positions of the genes from the reference genome in the duplicated genome are preprocessed and stored before the running the dynamic programming algorithm. The second output form this preprocessing step is a set of missing genes, i.e the genes not found in the duplicated genome. After this pre calculations a number of occurrences of the genes from reference interval in the duplicated genome $occ(R[i,j])$ and the number of missing genes S_{MIS} can be determined easily.

2.5 Time complexity

Lemma 3. *Maximum sum of two disjoint intervals problem has time complexity $O(n)$, where n is the length of the sequence.*

Dynamic programming algorithm evaluates matrix F which have a constant number of rows, because sequence from $MS2DIP$ is divided at most into two reserved and three residual intervals. Length of F is equal to the length of the duplicated genome, so complexity of the dynamic programming is $O(cn) = O(n)$, where c is the number of conserved and residual intervals. The preprocessing step of getting number of occurrences and missing genes from the reference interval can be calculated in the linear time.

2.6 General problem of the maximum sum of k disjoint intervals

A whole genome duplication is the most frequent case of a polyploidization event and the result is that the whole set of chromosomes is doubled. However some species have undergone polyploidization of higher orders. After a WGD an organism has four chromosomal sets, but for example strawberries can have four, six, eight or even ten chromosomal sets. An other plant example is wheat which can have four or six chromosomal sets.

We can easily extend our model to localize gene clusters not only in duplicated genomes but also in the genomes with higher polyploidy, where the genes are in more than in two copies. We will parametrize our model by parameter k which denotes the order of the polyploidization. For $k = 2$ we get a problem of the whole genome duplication studied and defined in the previous chapters. For $k = 3$ we deal with hexaploidization, $k = 4$ octaploidization and so on.

The general problem of the best interval location is defined in the following way. For some reference interval $R[i, j]$ we want to localize k disjoint intervals with maximum score in the polyploid genome. Scoring function from Definition 12 is easily modified to deal with the higher number of intervals. This biological problem can be transformed to the mathematical problem of the maximal sum of k disjoint intervals and solved by the same algorithm based on dynamic programming as for $MS2DIP$. The only difference is that sequence will be divided into at most k conserved intervals and $k + 1$ residual intervals so the number of rows in the matrix F will increase to $2k + 1$ rows plus the initial row. Complexity of this generalized problem solved by dynamic programming is $O(kn)$.

Problem of the maximal sum for k disjoint intervals was studied by Csűrös [Csu04] as Maximal scoring segment sets problem. They do not use dynamic programming to solve problem but an algorithm based on set covering. Complexity was the same as our $O(kn)$ in the basic implementation. He also proposed an algorithm running in $O(n \log n)$ time, but in our application domain the size of k is incomparably smaller than the length of a genome so this complexity is worse than $O(kn)$. Bengtsson and Chen [BJ07] developed a linear-time algorithm. Owing to the fact that parameter k acquires just small values, our dynamic programming will run sufficiently fast with the advantages of a simple implementation and using simple data structures.

2.7 All locations problem

The goal of the All location problem is to find the best interval locations for all possible reference intervals. To solve the problem, we will simply iterate through the reference genome via two nested loops. The outer loop has fixed starting position i of the reference interval, and the inner loop is then changing the ending position j gradually in every iteration. Adopting this strategy will allow us to use a modified version of sequence D' computed for the reference interval $R[i, j]$ in calculation for $R[i, j + 1]$. Preprocessing phase is then reduced only to changing the values from $-\alpha$ to β in all positions in D' corresponding to occurrences of the newly added gene $R[j + 1]$. If the reference genome has length m, then we consider $O(m^2)$ different reference intervals, and each round of $MS2DIP$ linearly depends on the length of duplicated genome n, so the overall complexity is $O(m^2 n)$.

In order to get only significant solutions according to Definition 15, we need to specify filtering conditions C_{filt}. In our implementation we defined one non-parametrized and two parametrized conditions.

Condition 1: The first parameter is the minimal score,denoted $minScore$. From all the best interval locations in $SOL(P)$ we select into a set P_1 only those whose score achieves at least the specified minimal value.

Condition 2: The second condition considers reference intervals covering each other. (See Figure 2.2). Many reference intervals in set $SOL(P)$ are covered by other reference intervals or cover these reference intervals. We say that interval $R[a, b]$ covers an interval $R[c, d]$ if $a < c$ and $b > d$. To set P_2 we select only the best locations from $SOL(P)$ whose reference interval does not cover or is not covered by other reference interval, whose best location has a higher score. More precisely, no two reference intervals in P_2 cover each other and for each interval which is

in $SOL(P)$ but not in P_2, there exists a reference interval in the set P_2 which covers or is covered by that interval and its score is higher or equal.

Condition 3: The third parametrized condition refers to overlapping reference intervals. (See Figure 2.3). We introduce a parameter *overlay* which defines what length of overlaps is still admissible, and for pairs with longer overlaps we select only the higher scoring location into a set P_3.

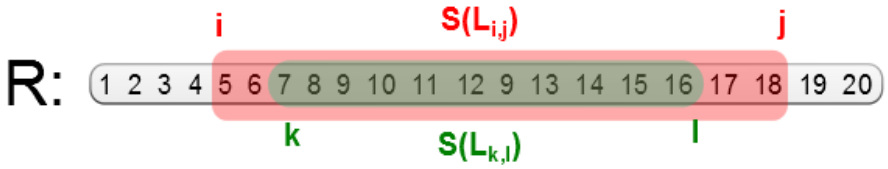


Figure 2.2: Example of reference intervals where an interval $R[i, j]$ covers an interval $R[k, l]$ or the interval $R[k, l]$ is covered by the interval $R[i, j]$.

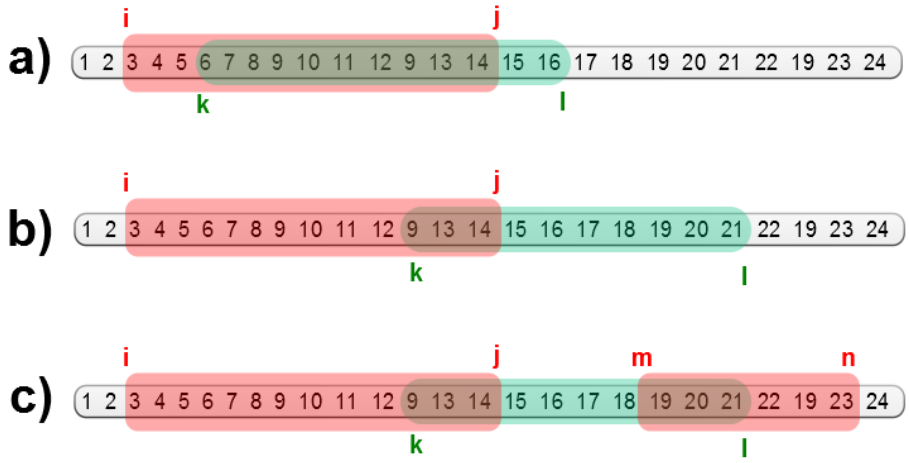


Figure 2.3: Example of overlapping reference intervals.

We could apply each condition to set $SOL(P)$ independently, and then define the set of significant solution of the All interval locations problem P as the intersection of the output sets from all conditions, i.e. $\overline{SOL}(P) = P_1 \bigcap P_2 \bigcap P_3$. In our implementation, we instead filter the set $SOL(P)$ in the order as conditions were stated, and use the output set from one condition as an input to the next one, so at the end the final set of significant solution is equal to the set P_3, i.e. $\overline{SOL}(P) = P_3$.

Selecting the best locations using the condition of minimal score does not require any special algorithm. All the best locations are simply scanned, and the ones not

meeting *minScore* requirement are filtered out. Filtering procedures for Condition 2, Condition 3 and setting of *overlay* parameter we be will briefly described in more detail in the following subsections.

2.7.1 Filtering covering intervals

In this subsection we introduce our filtering algorithm for choosing the best locations with the highest score whose reference interval does not cover or is not covered by intervals with higher score.

Prior to describing the algorithm itself, concept of the right and left neighbour to some interval is introduced. Consider a set X of intervals
$R[a_1, b_1], R[a_2, b_2], .., R[a_n, b_n]$, which are ordered by their starting coordinates a_i. Then the left neighbour of interval $R[a, b]$ is an interval $R[a_l, b_l]$ from X such that $a_l \leq a$ and a_l is the biggest one satisfying this condition (no interval from X starts between a_l and a). The right neighbour, $X[a_r, b_r]$ is an interval from X such that $a \leq a_r$ and a_r is the smallest value satisfying this condition (no interval form X starts between a and a_r).

Let a set I be the set of all reference intervals from the best locations in the set P_1, which are ordered in descending order by the score of their corresponding best locations. The set I is an input to a proposed filtering Algorithm 1. An output from the Algorithm 1 is a set O which contains reference intervals ordered by the start coordinates such that no interval covers any other interval from O or is covered by any interval from O. To get the corresponding output set P_2 from Condition 2, reference intervals from O have to be associated back with their best interval locations.

Algorithm 1 Filtering covering intervals

Input: Input set I of n reference intervals $R[a_1, b_1], R[a_2, b_2], ..., R[a_n, b_n]$ ordered by the score in descending order

Input: Output set O of reference intervals ordered by the start coordinates where no interval covers an other interval from O, neither is covered by any interval from O

1: insert the first interval from I to O and remove it from I;
2: **while** $I \neq \emptyset$ **do**
3: $i = R[a_i, b_i] \leftarrow$ first interval from I;
4: $l = R[a_l, b_l] \leftarrow$ left neighbour of i from the set O or $NULL$ if not exists;
5: $r = R[a_r, b_r] \leftarrow$ right neighbour of i from the set O or $NULL$ if not exists;
6: **if** $(l = NULL) \vee (b_l < b_i) \wedge (r = NULL) \vee (b_i < b_r)$ **then**
7: add interval i to the set O;
8: **end if**
9: remove interval i from set I;
10: **end while**
11: **return** O

Lemma 4. *Consider an input set I of n reference intervals $R[a_1, b_1], R[a_2, b_2], ..R[a_n, b_n]$ ordered by the score of their corresponding best locations from the most valuable to the least one. Algorithm 1 filters these intervals and outputs result set $O \subseteq I$ such that no interval from set O is covered by other interval from O. In addition, for each interval $R[a_x, b_x]$ from the set $I \backslash O$, there exists an interval $R[a_y, b_y]$ in the final set O which covers $R[a_x, b_x]$ or is cover by $R[a_x, b_x]$ and the score of $R[a_y, b_y]$ is higher or equal to the score $R[a_x, b_x]$.*

Proof. The invariant is that set O never contains two intervals such that one covers the other. This invariant is valid during the whole algorithm, because in line 7 only non conflicting intervals are added to the set O.
To test if interval i from line 3 is covered by some interval in O, we consider its left neighbour.

1. If interval i does not have a left neighbour, no interval starting earlier than i exists in O. For that reason, no interval from O can cover i.

2. If the interval i has a left neighbour l from the set O (line 3), then it is necessary to test if i is not covered by l by comparing end points b_l and b_i. If $b_l < b_i$ then the same is true for all intervals $R[a_x, b_x]$ form O which start earlier than l $(a_x < a_l < a_i)$ because of the assumption of valid invariant$(b_x < b_l < b_i)$.

Similarly, to test if interval i is covering any interval from O, we consider its right neighbour.

1. If interval i does not have a right neighbour i, no interval from O can be covered by i.

2. If interval i has a right neighbour r, we check if i is covering r by testing whether $b_i < b_r$. For every other interval $R[a_x, b_x]$ from O with $a_x > a_r$ holds (due to the valid invariant) that $b_x > b_r > b_i$ and $R[a_x, b_x]$ can not be covered by i.

If condition form line 3 is true, then the interval i is added to the set O.
Every interval from $I \backslash O$ was considered as interval i at some point and because it was filtered out, at the time of its investigation, there existed an interval in the set O which covered this interval or i covered it. Intervals are in I ordered by their score in descending order, so at the time of the investigation, all intervals in the set O have the same or higher score compared to i. This finishes the proof of the last part of the lemma. □

The input set I from the algorithm 1 has n reference intervals and is obtained from the set P_1 by sorting the reference intervals by the score in descending order. It can be implemented in $O(n \log n)$ time. The worst case scenario for the algorithm 1 is, that all reference intervals from I will be added to the set O. If the set O is stored in an effective data structure, retrieving a left and right neighbour from O can be done in $O(\log n)$ time. This is done for each reference interval form I, therefore the overall time complexity of the filtering algorithm 1 is $O(n \log n)$.

2.7.2 Filtering overlapping intervals

Overlapping best locations are such locations whose reference intervals intersect, i.e. the end of one reference interval overlaps start of an other one or vice versa. For simplicity, we will assume that no interval covers another, which is the case after applying filtering algorithm from the previous section. As we can see from Figure 2.3,a reference interval can overlap with one interval (case a and b) or with two intervals from both sides (case c). An important aspect of our design was to define a rule for removing insignificant overlapping intervals. If two intervals have a large overlap as case a in Figure 2.3, we want to save only the interval with the higher score. Parameter *overlay* defines the maximum length of the overlap. Setting *overlay* to a constant value would be difficult, because intuitively the threshold depends on the interval length. If the intervals from case a in Fig. 2.3 would be two times longer, then the same long overlap would not be seen as significant, because both intervals would have sufficiently long

regions that are not shared. The best solution appears to set the *overlay* dynamically so that the *overlay* would be a constant fraction ($0 \leq overlay < 1$) of the length of the overlapping intervals. In our implementation, the *overlay* is half of the length of the shorter of the two intervals. For example, in case b from Fig. 2.3 it means, that the distance $|j - k|$ has to be less than half of the length of the reference interval $R[i, j]$ and the reference interval $R[k, l]$, i.e. $|j - i| \leq |R[i, j]|/2$ and $|j - i| \leq |R[k, l]|/2$. If this conditions is fulfilled, we preserve both intervals $R[i, j]$, $R[k, l]$ and if not, then the interval with the smaller score is thrown away. Situation in case c from Fig. 2.3 is handled n the same way.

We outline the algorithm used to handle the overlapping intervals. Input to the Algorithm 2 is a set I of the reference intervals obtained as set P_2 from Condition 2. The intervals in I are ordered by score in descending order. The other input parameter is *overlay*. The output set O contains reference intervals which either do not overlap at all or if they overlap, then the length of their intersection is less than the specified fraction of the lengths of both overlapping intervals.

Algorithm 2 Filtering overlapping intervals

Input: Input set I of n reference intervals $R[a_1, b_1], R[a_2, b_2], ..., R[a_n, b_n]$ sorted by score in descending order and parameter *overlay*

Input: Output set O of correctly overlapping reference intervals ordered by the beginnings

1: insert the first interval from I to O and remove it from I;
2: **while** $I \neq \emptyset$ **do**
3: $i = R[a_i, b_i] \leftarrow$ first interval from I;
4: $l = R[a_l, b_l] \leftarrow$ left neighbour of i from set O or $NULL$ if not exists;
5: $r = R[a_r, b_r] \leftarrow$ right neighbour of i from set O or $NULL$ if not exists;
6: **if** $\Big(l = NULL \vee (b_l - a_i + 1 < (b_l - a_l + 1) * overlay \wedge$
$\wedge\ b_l - a_i + 1 < (b_i - a_i + 1) * overlay)\Big) \wedge \Big(r = NULL \vee$
$\vee\ b_i - a_r + 1 < (b_l - a_l + 1) * overlay \wedge b_i - a_r + 1 < (b_i - a_i + 1) * overlay\Big)$ **then**
7: add interval i to set O;
8: **end if**
9: remove interval i from set I;
10: **end while**
11: **return** O

We investigate each interval i from the set I and find its right and left neighbour from the current set O. It is sufficient to check conflicts only with these neighbours, because if in the set O would be other interval more to the left or to the right causing

conflict with i, it would also have a conflict with i's right or left neighbour, and that is not possible because the set O always contains only non conflicting intervals. We assume the same kind of invariant as in Algorithm 1. In line 6 we check for overlaps the left and right neighbour. Note that, if interval i does not overlap with its left neighbour, $b_l - a_1 + 1$ will be negative and thus automatically less than overlay fraction of lengths of i and l. If the overlay limit is not violated for interval i and its neighbours, then in the line 7 we add this interval i into the set O.

2.8 Application of our algorithm to biological data

We run the algorithm for All location problem on several pairs of yeast genomes and filtered best locations to obtain the significant solution. Our data set contains genomes of four species from the genus *Magnusiomyces* and one species from the sister genus *Yarrowia* which both belong to the family *Dipodascaceae*, class *Saccharomycetes* and the kingdom *Fungi*. Genomes from the *Magnusiomyces* were recently sequenced by the group of Prof. Jozef Nosek from the Faculty of Natural Sciences of Comenius University. The genome of *Yarrowia lipolytica* was downloaded from the Genebank database. Figure 2.4 shows the phylogenetic tree of the genus *Magnusiomyces*([KFT]). Micro-organisms in the red rectangles are species where we hypothesize that a whole genome duplication occurred, and their genomes are refereed to as duplicated in our test. Species in the green rectangles are used as reference organisms. Genomes of our available *Magnusiomyces* are not fully assembled into the chromosomes, but only into contigs. In order to test the designed algorithm and the concept of identifying gene clusters on more complex data, we also use the genome of *Yarrowia lipolytica* as a reference, because its genome is available as whole chromosomes.

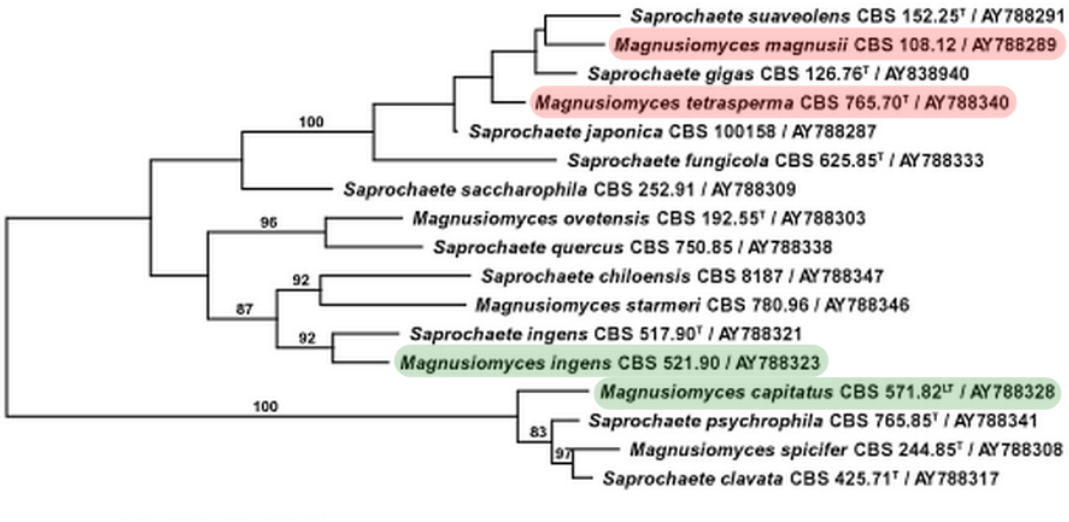


Figure 2.4: Phylogenetic tree of species from the genus *Magnusiomyces* [KFT].

We run a test for every combination of the duplicated and reference genome. For all combinations we used value $minScore = 10$ as the minimal score threshold from Condition 1, except for those combinations where *Yarrowia lipolytica* was used as the reference organism. *Yarrowia* is from a sister genus, so the phylogenetic distance is longer compared to the other reference genomes of *M. ingens* and *M. Capitatus*, so in these tests we set $minScore = 5$. We implemented a small interactive application for displaying identified significant best locations. Figure 2.5 shows a snapshot of the results for the combination *M. magnusii* - *M. ingens*. A whole genomes is drawn as a sequence of contigs which are numbered in ascending order. Each reference interval in the reference genome has its color and number and its corresponding best double location in the duplicated genome has the same color and number. Best locations are numbered in descending order by their scores. In the example from the picture, the best location with the highest score is highlighted in the circles.

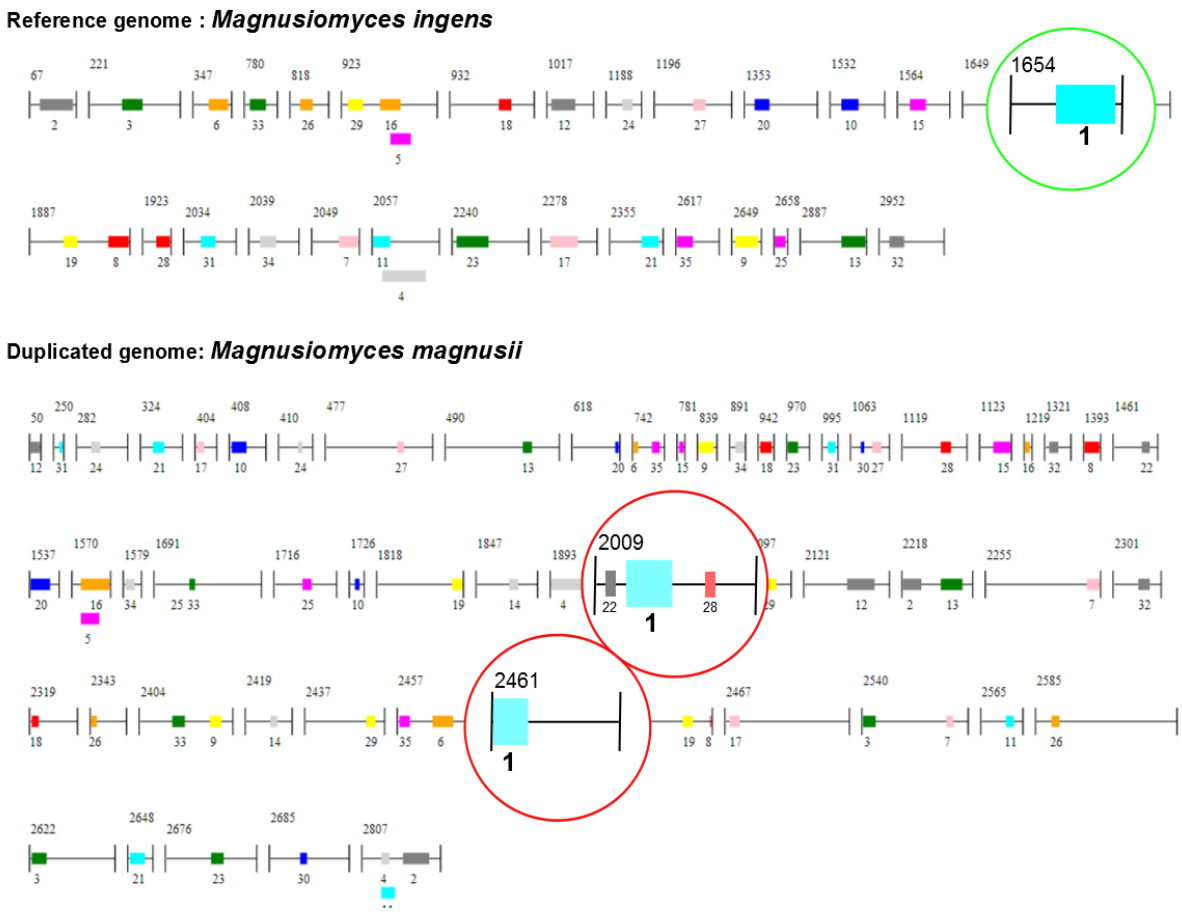


Figure 2.5: Significant best interval locations for *M. magnusii*, if *M. ingens* is used as the reference organism. The best interval location with the highest score is highlighted.

All results are summarized in the following histograms. Histogram number 2.6 reveals how many best interval locations of some score (minimal value is set to be $minScore = 10$) are found in *M. magnusii*, if either *M. ingens* or *M. capitatus* are used as the reference genomes. *M. ingens* appears to be a better reference genome with significantly more identified best locations. The second histogram 2.7 shows the same results but for the duplicated genome of *M. tetrasperma*. Again *M. ingens* turns out to be a better reference genome. This is even supported by the phylogenetic tree from Fig. 2.4 which shows that *M. ingens* is phylogenetically closer to both duplicated yeasts compared to *M. capitatus*, which is located on the branch which was separated from the rest at the beginning. The last histogram 2.8 depicts the number of the best locations identified in both duplicated genomes when *Yarrowia lipolytica* is used as the reference genome. In this case, the minimal score is set to $minScore = 5$, and because of *Yarrowia* being phylogenetically further, only fewer interval locations are determined.

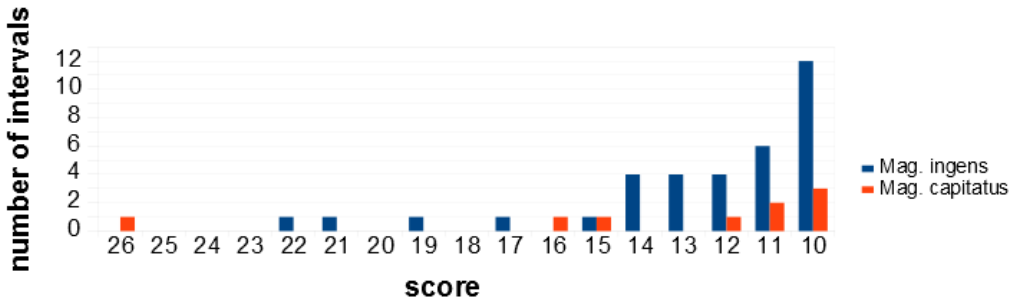


Figure 2.6: Histogram showing the number of the best interval locations identified in *M. magnusii* using two different reference genomes: *M. ingens* and *M. capitatus*

.

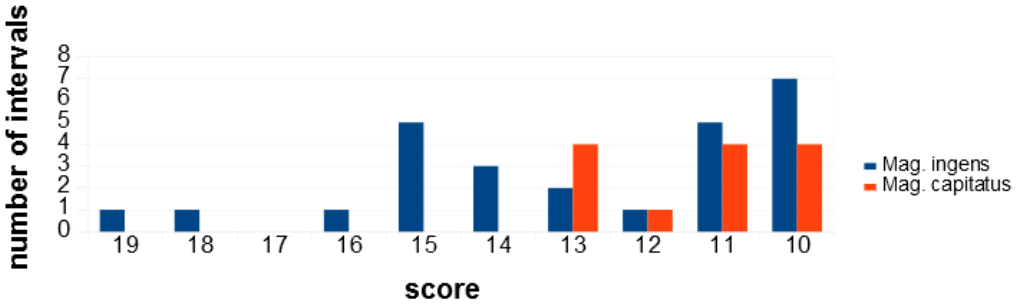


Figure 2.7: Histogram showing the number of the best interval locations identified in *M. tetrasperma* using two different reference genomes: *M. ingens* and *M. capitatus*

.

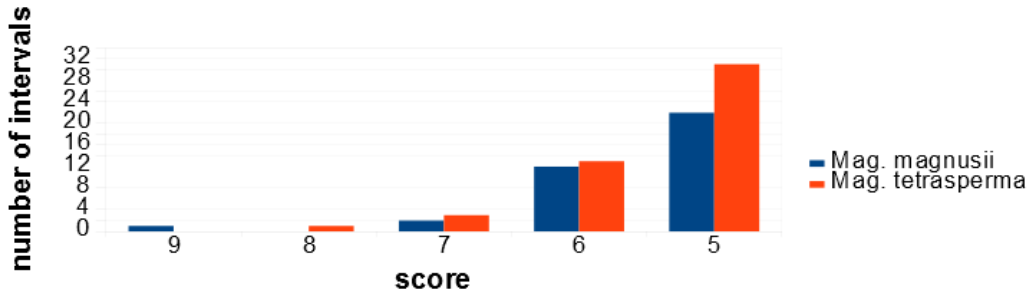


Figure 2.8: Histogram showing the number of the best locations identified in *M. magnusii* and*M. tetrasperma* using a reference genome of *Yarrowia lipolytica*.

Trying to get better results in terms of the number of the best interval locations with high score, we implemented two improvements.

Reduced genome The first strategy is based on a modification of the input genomes into a form resembling ideal genomes after the event of the WGD, when all reference genes were in the duplicated genome in two copies and no other genes

were present. We scan the reference and duplicated genome and retain only genes that occur in one copy in the reference genome and in one or two copies in the duplicated genome.

Triple location The second strategy is based on the fact that our available duplicated genomes consist only of contigs. thus a region which would be significant interval location in the whole region can be missed by our algorithm, because one of the duplicated copy is spilt into to different contigs. Our algorithm is easily extensible to look for more than two intervals in the duplicated genome and so we run it to localize three intervals in the duplicated genome with the best score for some reference interval.

Both strategies were proven to be successful in our tests on real data, and we present only one illustrative table 2.1 the genomes of *M. Magnusii* and *M. ingens*. In the second column is the number of intervals of a given score from the first column for the standard run without any data adjustments. In the third column are numbers for the test with adapted strategy Reduced genome and in the forth for Triple location. Both strategies result in an increase of the identified best interval locations with the score meeting the required minimal value $minScore = 10$. All locations from the second column occur in the third and forth column either as the same location with a higher score or as a part of a bigger and more valuable interval location. Score of some reference intervals was improved by applying only the two adjusted methods. On the other side, for many reference intervals the best locations with sufficient score are found by applying both strategies. In this case, each strategy corrects different deficiencies, but both are efficient enough in increasing the score so the intervals are then detected by the algorithm.

It is difficult to determine which strategy is better. Both have their disadvantages, but on this input Triple location produces more significant interval locations than Reduced genome method. Ideally the algorithm should be run again once the contigs are joined into the chromosomes to avoid to avoid the necessity of Triple method. The Reduced genome strategy changes the content of the genomes so that the reference and duplicated genome show fewer differences, but we ignore the current divergence of the genomes, so it is difficult to estimate biological relevance of the results. The Triple location method is based on the assumption that two of the found intervals should be assembled together.However this might not be a case in the reality.

Score	**Standard**	**Reduced**	**Triple**
28	0	0	1
27	0	0	0
26	0	0	0
25	0	0	1
24	0	0	1
23	0	0	1
22	1	2	1
21	1	1	0
20	0	1	0
19	1	2	1
18	0	0	1
17	1	0	0
16	0	6	3
15	1	3	6
14	4	5	5
13	4	4	2
12	4	8	11
11	6	5	8
10	12	17	16
Total	35	54	58

Table 2.1: The number of identified best locations in *M. magnusii*, when *M. ingens* is the reference organism, using Standard method with no additional adjustments to the data set, Reduced genome and Triple location strategy. The minimal score parameter is set to be $minScore = 10$.

Chapter 3

Heuristic improvements

The goal of this chapter is to design strategies decreasing runtime of the algorithm solving the All location problem. We develop few heuristics which improve the original algorithm and are based either on breaking down the Best interval location problem to sub-problems defined for each chromosome, or on skipping the dynamic programming if the score of the best interval location is predicted to achieve too low value. All developed heuristics do not reduced time complexity in the worst case scenario, but they considerably speed up the algorithm applied to biological and generated data.

The relatively fast execution of the dynamic programming solving $MS2DIP$ problem in yeast genomes is due to the few reasons. The first reason is, that we do not solve the Best locations problem for the reference intervals excessing the boundary of chromosomes. The second argument is, that tested yeasts from the genus *Magnusiomyces* have short contings instead of chromosomes. These two facts cause that $MS2DIP$ is executed only for quite small subset of all $O(m^2)$ possible reference intervals.

However, motivation for developing efficient heuristics still exists. Firstly, if tests in previous chapter are performed using *Yarrowia lipolytica* with whole chromosomes as reference genome, the computation time is several times longer. Secondly, the lengths of used genomes are pretty small compared to other species, and therefore in case of applying our model to longer genomes we want to achieve feasible computation time. The last motivation comes from the statistical significance of the best interval location. The statistically significant score of the best interval location can be determine experimentally by solving the All location problem repeatedly on randomized data, and this method is computationally expensive.

In the following subsections we describe four heuristic improvements.

3.1 Simple heuristics

We describe two simple improvements which reduce the set of reference intervals for the All location problem and shorten the sequence D' for $MS2DIP$. The first improvement use the value $minScore$ to skip the execution of dynamic programming for many reference intervals. According to the scoring function S from Definition 17, the maximal possible score of the reference interval $R[i, j]$ is equal to its length. As a result, the reference intervals shorter than $minScore$ will never achieve $minScore$ and will be filtered out. This fact enables us to reduce the set of reference intervals for the All location problem by those reference intervals which length is too short, i.e. $|R[i, j]| < minScore$.

The second improvement shortens the sequence D' for $MS2DIP$ to such maximal subsequence which contains all genes from the reference interval $R[i, j]$. Precisely, the starting and ending position of subsequence correspond to the leftmost and rightmost occurrence of some gene from the reference interval and can be detected from the outcome of preprocessing phase.

Both these simple heuristics require only small modifications in the algorithm for $MS2DIP$ and no additional data structures are needed.

3.2 Low score heuristic

An idea behind this heuristic is to mathematically characterize the situation, when the score of the best interval location $S(L_{i,j})$ is so low, that the score of the best interval location for the next reference interval $R[i, j + 1]$ will not achieve sufficiently hight value to meet the $minScore$ requirement. (Figure 3.1)

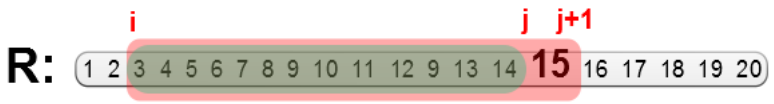


Figure 3.1: A reference interval $R[i, j]$ and an interval $R[i, j + 1]$ extended by gene $R[j + 1]$.

If we know $S(L_{i,j})$ and the number of occurrences of the gene $R[j + 1]$ in the duplicated genome, then we can define an upper bound for the score of the best interval location for the next reference interval $R[i, j + 1]$ according to the following lemma.

Lemma 5. *Assume that the score $S(L_{i,j})$ for the reference interval $R[i, j]$ was calculated by the scoring function S with parameters α, β, γ and let $occ(R[j+1])$ denote the number*

of occurrences of the reference gene $R[j+1]$. Then the score of the best interval location for the reference interval $R[i, j+1]$ is bounded by this inequality:
$S(L_{i,j+1}) \leq S(L_{i,j}) + \alpha occ(R[j+1]) + 1$.

Proof. We prove the lemma backwards. Consider the best interval location $L_{i,j+1}$ for reference interval $R[i, j+1]$. Let us estimate the score of its double location in combination with reference interval $R[i, j]$. The worst scenario is after removing $R[j+1]$ gene is that all occurrences of $R[j+1]$ remain as outsider genes inside the double location and would be counted in term S_{IN} from the scoring function S. Then the score would be decreased by the value $\alpha occ(R[j+1])$ compared to $S(l_{i,j+1})$. This score is denoted as $\tilde{S}$. However, in the optimal solution for $R[i, j]$, some occurrences of the gene $R[j+1]$ may be localized outside the double location and not be counted as a part of the set S_{IN}, so the optimal score $S(L_{i,j})$ can not be worse than $\tilde{S}$. As a result the following inequality holds: $S(L_{i,j}) \geq \tilde{S} = S(L_{i,j+1}) - \alpha occ(R[j+1]) - 1$.
-1 term on the left side is due to the fact that interval $R[i, j]$ is shorter by one gene compared to $R[i, j+1]$. We get the inequality from the lemma statement by applying simple mathematical operations to the previous inequality. □

The incorporation of this heuristic to the original algorithm for $MS2DIP$ is not difficult and requires only few modifications. Before solving $MS2DIP$ for the reference interval $R[i, j+1]$, we calculate its predicted best score according to formula from Lemma 5. If it is smaller than $minScore$, then we skip this reference interval. We store the computed best possible score for the omitted interval $R[i, j+1]$ ($S(L_{i,j}) + \alpha occ(R[j+1]) + 1$), so the value can be used in the next iterations for the longer reference intervals. As soon as the predicted score achieves the $minScore$ value, we run the dynamic programming for $MS2DIP$ again.

3.3 Contig heuristic

The standard method of solving $MS2DIP$ runs the dynamic programming in the whole sequence D'. The terminal character using to join the chromosomes in the duplicated genome into the sequence D' has the score equals to $-\infty$. Thus, no interval from the double location is localized in two chromosomes. In more details, interval in the duplicated genome never exceeds the boundaries of chromosome, because its score would be close to $-\infty$ and an empty interval with the score 0 would be chosen instead. This fact enables us to break down the $MS2DIP$ problem into sub-problems and solve the instance of $MS2DIP$ for each chromosome separately. We call this heuristic as Contig heuristic because genomes of available yeasts consisting of contigs were the first organisms where this improvement was tested.

If $MS2DIP$ is calculated for every chromosome independently, then we have to correctly combine these partial solutions to give the final solution to the original $MS2DIP$ problem. An interval location for some reference interval has at most two disjoint intervals in the duplicated genome. These two intervals in the duplicated genome can be either localized together in the same chromosome or separately in two different chromosomes. Therefore each modified instance of $MS2DIP$ has to find in each chromosome the best two intervals with the maximum score which are referred to as the best local double location, and also the best single interval with the maximum score which is called the best local single location. If the solution of the Best interval location problem has the double location situated in one chromosome from the duplicated genome, then we return the best global double location, i.e the best local double location having the highest score over the all best local double locations. If the intervals from the double location are in the different chromosomes, then we return the first two best global single locations, i.e two best local single locations which have the highest and the second highest score over the all best local single locations.

In the Figure 3.2 are shown few possibilities how the best local single location and the best local double location can be situated in the chromosomes.

The best local double location in some chromosome is obtained as the standard solution of $MS2DIP$ problem by replacing the sequence D' by its subsequence which corresponds to the given chromosome. Matrices F and T used in dynamic programming will be helpful in solving the problem of the maximum sum of one interval. To find the best local single location in the same chromosome, we do not have to developed a new algorithm, because the solution can be simply determined from already calculated matrices F and T. If we want to find only one interval with the maximum sum, then the sequence for $MS2DIP$ is divided into one reserved interval and two residual intervals situated on the sides(see Fig. 3.2). The maximum sum of one interval is stored in the rightmost cell in the row from matrix F which corresponds in the original problem to the second residual interval. For more details see section 2.4.

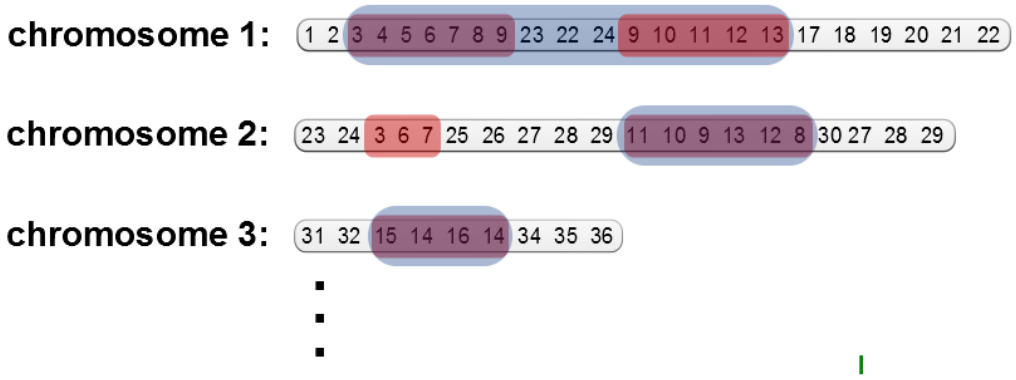


Figure 3.2: A schematic picture of the identified best local double (red) and best local single (blue) location in the chromosomes in the duplicated genome for the reference interval $R[i,j] = (3,4,5,6,7,8,9,10,11,12,13,14,15,16)$

When it comes to the implementation of this heuristic, there are required few modifications to the standard algorithm. We run $MS2DIP$ in each chromosome and as the result we get the best local double location ($L_{i,j}^{double}$) and best local single location ($L_{i,j}^{single}$). To give the final solution we need to store only the best global double location($GL_{i,j}^{double}$) and two best global single locations ($L_{i,j}^{single1}$,$L_{i,j}^{single2}$) selected from the all chromosomal solutions. After $MS2DIP$ returns the partial results for some chromosome, we have to compare the values of $L_{i,j}^{double}$ and $L_{i,j}^{single}$ with the corresponding global values:

if $S(L_{i,j}^{double}) > S(GL_{i,j}^{double})$ **then**
  $GL_{i,j}^{double} = L_{i,j}^{double}$;
end if
if $S(L_{i,j}^{single}) > S(GL_{i,j}^{single1})$ **then**
  $GL_{i,j}^{single1} = L_{i,j}^{single}$;
  $GL_{i,j}^{single2} = GL_{i,j}^{single1}$;
end if
if $S(L_{i,j}^{single}) > S(GL_{i,j}^{single2})$ **then**
  $GL_{i,j}^{single2} = L_{i,j}^{single}$;
end if

At the end we test if $\mathbf{S(GL_{i,j}^{double}) > S(GL_{i,j}^{single1}) + S(GL_{i,j}^{single2})}$ and return as the final solution either the best global double location or combination of two best global single locations.

The other crucial advantage of this heuristic is that $MS2DIP$ does not have to be usually executed in every chromosome. Let $occ^i(R[i,j])$ denotes the number of occurrences of the genes from the reference interval $R[i,j]$ in the chromosome i. If we

order the chromosomes by $occ^i(R[i,j])$ in descending order and perform $MS2DIP$ in the chromosomes in the same order, then we can formulate the condition of terminating the heuristic execution.

Lemma 6. *Consider n chromosomes $ch_1, ch_2, .., ch_n$ ordered by the number of occurrences of the genes from the reference interval $R[i,j]$ in descending order. Assume that $MS2DIP$ is performed in the chromosomes in the same order. Let S be the scoring function with the parameters α, β, γ. The variable $GL_{i,j}^{single2}$ stores the score of the second best global single location. As soon as the chromosome ch_i chosen for $MS2DIP$ satisfies the condition: $\beta occ^i(R[i,j]) \leq GL_{i,j}^{single2}$, the execution of Contig heuristic can terminated.*

Proof. When all genes from the reference interval $R[i,j]$ occur in the best local single interval in the chromosome i and the interval does not contain any outsider genes, than the score is equal to $\beta occ^i(R[i,j])$. This is the maximum possible score for the given reference interval $R[i,j]$ in the chromosome i. If this value is smaller or equal than the score of the second global best single location $GL_{i,j}^{single2}$, than the result of $MS2DIP$ in this chromosome i will not change the global values. The other chromosomes $ch_{i+1}, .., ch_n$ will have even lower score due to their ordering, so from this moment the global values of $GL_{i,j}^{double}$, $GL_{i,j}^{single1}$, $GL_{i,j}^{single2}$ remain unchanged so we can terminate the execution of this heuristic approach. The comparison to the best global double location $GL_{i,j}^{double}$ is not needed, because the score of the global double location is always higher or equal than the score of the global single location, i.e $GL_{i,j}^{global} \geq GL_{i,j}^{single1} \geq GL_{i,j}^{single2}$, and it can be easily proved. The inequality holds also for the best local double and single location. If we have some outsider genes inside the best local single location, then by splitting the interval in the position where at least one outsider gene is localized, we get the local double location with better score. (See chromosome 1 in the Figure 3.2). □

3.4 Segmentation of duplicated genome into runs

This heuristic does not modify the original algorithm, but changes the data structure used in the dynamic programming. In the standard implementation of $MS2DIP$, the sequence D' is stored in an array where each cell has either value $-\alpha$ or β (resp. $-\infty$ for the terminal character). If we look closer at the sequence D' (Fig. 3.3), we see continuous subsequences of either $-\alpha$ or β values. We can merge them into the one object and we refer to this positive (or negative) subsequences as positive (negative) runs (terminology is adopted from[BJ07]). As a result, an individual gene score does

not have to be wastefully stored in an array as a single value, but is sufficiently to remember only the beginning, end and the score of the run containing that gene. The sequence D' is then segmented into the runs(see Fig. 3.3).

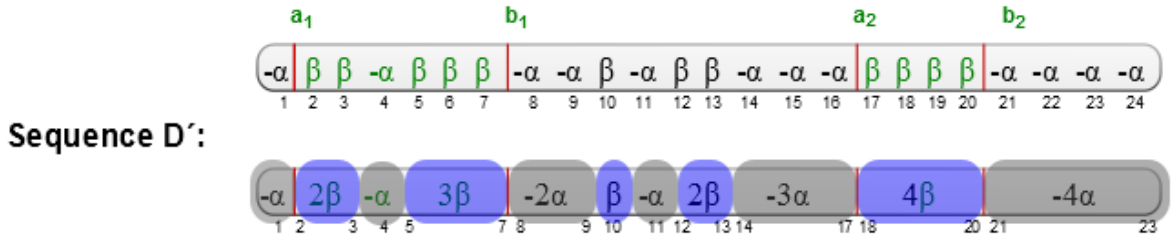


Figure 3.3: A schematic picture of the sequence D' either as a sequence of $-\alpha$, β values or as a sequence of positive and negative runs, where each run is characterized by its start, end point and score.

If we have the sequence D' for the reference interval $R[i, j]$ represented as an array, then solving $MS2DIP$ for the next interval $R[i, j + 1]$ requires changing the values form $-\alpha$ to β in the all occurrences of the new gene $R[j + 1]$. This can be done in $O(occ(R[j + 1]))$ time. In order to have the best time complexity for the operations maintaining the runs up to date, we do not store them in an array, but in the binary tree sorted by the start coordinates of the runs. To change the value from $-\alpha$ to β for some occurrence of the gene $R[j + 1]$, we have to look up the run containing this occurrence in the binary tree. The found run is always negative and two situation can happen.

1. An occurrence of $R[j+1]$ gene is inside the negative run. Then this run is divided into three new runs, one positive containing only one β value for $R[j + 1]$ gene occurrence, and two runs on both sides with appropriate negative score.

2. An occurrence of $R[j + 1]$ gene is the end point of the identified negative run. Then we enlarge the left or right positive run localized next to this negative run and add β to its positive score. At the end we modify the score of the negative run by adding the value α.

Update of all occurrences of $R[j+1]$ gene in the binary tree takes $O(occ(R[j + 1])log(occ(R[i, j + 1]))$ time. Every gene from the reference interval $R[i, j]$ can create a new positive run in the duplicate genome, so the maximum number of runs is proportional to the number of occurrences of all genes from the reference interval $R[i, j]$. Time complexity of the insertion to the binary tree is logarithmic, therefore $log(occ(R[i, j + 1]))$.

The following lemma states the equivalence of the solutions calculated by using the array and the binary tree of runs.

Lemma 7. *Assume that the sequence D' for $MS2DIP$ is represented as an array and as a binary tree of runs ordered by the start coordinates. If the dynamic programming solving $MS2DIP$ is performed in the array and in the tree of runs, then it outputs the same best interval location $L_{i,j}$ for the reference interval $R[i, j]$.*

Proof. It is necessary to show that the intervals from the double locations, calculated by the dynamic programming using an array to store the sequence D', do not start or end at such positions which are located in the middle of the corresponding positive runs. If the interval from the best location begins or ends at the position which is in the middle of the positive run, then the score of such double location would not be the highest and there would be a conflict with its maximality (we can get better solution just by including this remaining (outside) β values into the interval). □

3.5 Comparison between heuristics applied to biological data

In this section we compare the running time of standard algorithm from the section2.4 with some combinations of introduced heuristic improvements. We will show results of two simulations for the duplicated organism *Magnusiomyces magnusii*. Due to the simplicity we will refer to this genome as duplicated, although this hypothesis was not officially published. The first test uses *Magnusii ingens*(both are in the same genus) as the reference organism and the second *Yarrowia lipolytica* because of its fully assembled genome, so the running time is longer.
We have these combinations of heuristics :

standard Standard dynamic programming algorithm where the sequence D' is stored in an array.

simple Standard algorithm with the implemented simple heuristics from the section 3.1.

simple+lowScore Standard algorithm with the simple heuristics and the low score improvement from the section 3.2.

simple+lowScore+contig Standard algorithm as in the simple+lowScore combination, but $MS2DIP$ is solved in each chromosome separately (See section 3.3).

simple+runs Standard algorithm with the simple heuristics running in the binary tree of the positive and negative runs (See section 3.4).

3.5.1 *Magnusiomyces magnusii - Magnusiomyces ingens*

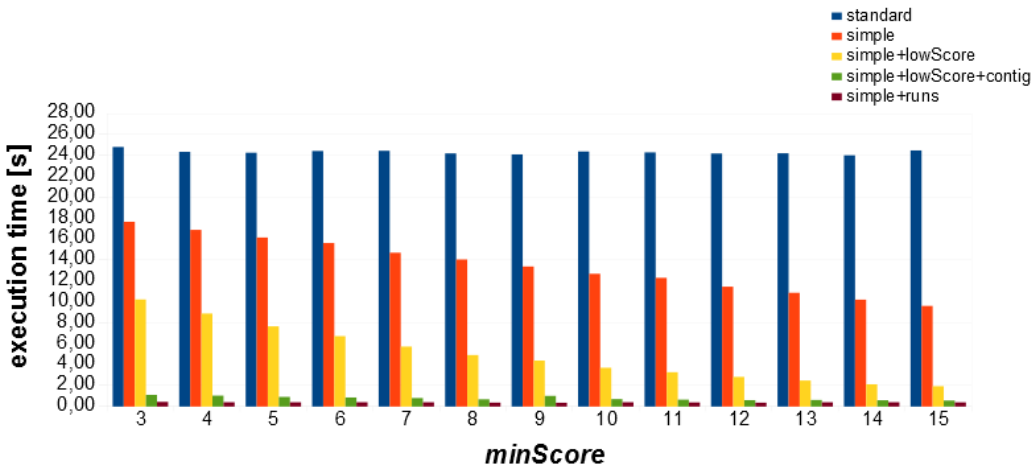


Figure 3.4: Graph for the time comparison of the standard algorithm with different heuristic combinations in test for *M. magnusii - M. ingens*.

In the next graph is shown another aspects of the heuristic improvements. We compare how many times the dynamic programming is executed during the calculation of the All location problem. The difference between the standard algorithm and either the simple heuristics or the combination of simple heuristics and low score heuristic correspond to the number of skipped calls of the dynamic programming. The table shows the percentage of the skipped calls compared to the standard algorithm where none calls are omitted.

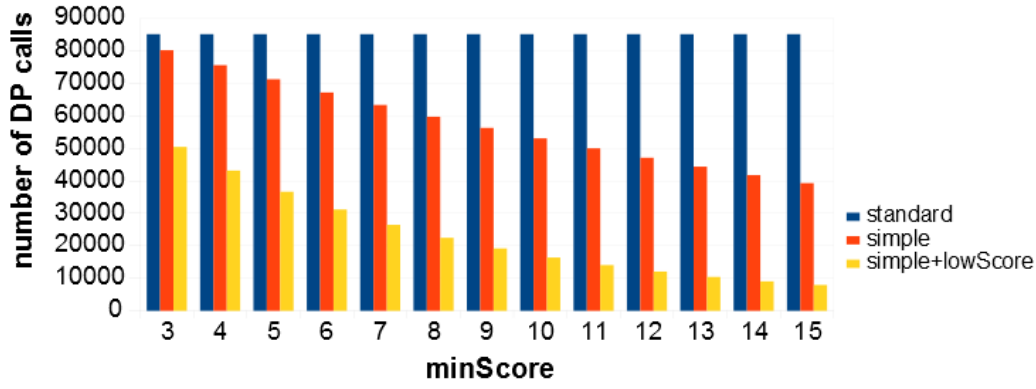


Figure 3.5: Number of the dynamic programming (DP) calls in the standard algorithm and in the algorithms implementing simple heuristics or the combination of simple and low score heuristics.

minScore	**standard[s]**	**simple[s]**	**simple+lowScore[s]**	**simple+lowScore+contig[s]**	**simple+runs[s]**
3	24,80	17,66	10,23	1,12	0,45
4	24,34	16,88	8,89	1,05	0,42
5	24,26	16,15	7,67	0,93	0,41
6	24,43	15,63	6,73	0,87	0,41
7	24,44	14,70	5,73	0,82	0,41
8	24,19	14,06	4,92	0,69	0,38
9	24,10	13,39	4,39	1,02	0,37
10	24,37	12,69	3,71	0,71	0,42
11	24,29	12,30	3,28	0,66	0,40
12	24,18	11,46	2,83	0,62	0,37
13	24,21	10,88	2,49	0,63	0,41
14	24,01	10,22	2,13	0,61	0,42
15	24,47	9,62	1,94	0,58	0,41

Table 3.1: Time comparison of the different heuristic combinations with the standard algorithm in test for *M. magnusii - M. ingens*. Tests are performed for the different values of *minScore*

minScore	simple[%]	simple+lowScore[%]
3	5,8	40,8
4	11,2	49,3
5	16,3	57,0
6	21,1	63,4
7	25,6	68,9
8	29,8	73,6
9	33,9	77,5
10	37,7	80,8
11	41,3	83,5
12	44,7	85,8
13	47,9	87,7
14	50,9	89,4
15	53,8	90,7

Table 3.2: Percentage of the omitted dynamic programming (DP) calls from the total amount calculated in the standard algorithm for the different values of *minScore* by implementing simple heuristics and the combination of simple and low score heuristics.

3.5.2 *Magnusiomyces magnusii - Yarrowia lipolytica*

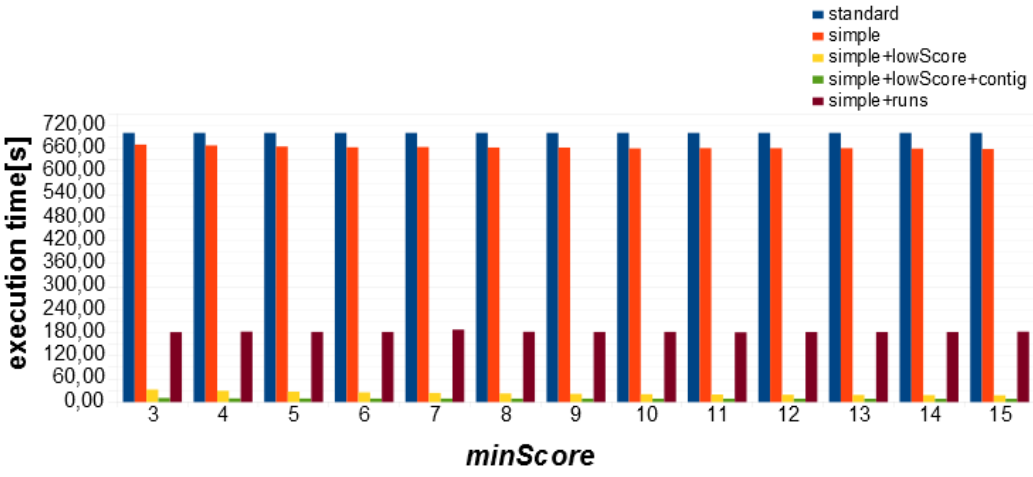


Figure 3.6: Graph for the time comparison of the standard algorithm with the different heuristic combinations.

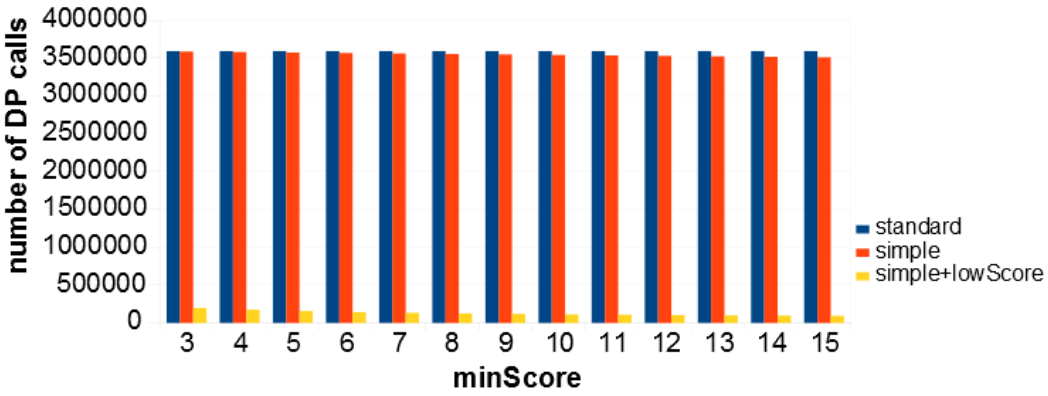


Figure 3.7: Number of the dynamic programming (DP) calls in the standard algorithm and in the algorithms implementing simple heuristic or the combination of simple and low score heuristics.

minScore	**simple**[%]	**simple+lowScore**[%]
3	0,18	94,51
4	0,36	95,16
5	0,53	95,64
6	0,71	96,01
7	0,89	96,29
8	1,06	96,52
9	1,24	96,70
10	1,42	96,86
11	1,59	96,99
12	1,77	97,11
13	1,95	97,21
14	2,12	97,30
15	2,30	97,39

Table 3.3: Percentage of the omitted dynamic programming (DP) calls from the total amount computed in the standard algorithm for the different values of *minScore* by implementing simple heuristics and the combination of simple and low score heuristics.

3.5.3 Assessments of heuristic improvements

From the graph 3.4 is evident that all heuristic combinations help to decrease the execution time of the All location problem comparing to the standard algorithm. The

simple heuristics and lowScore heuristic show better results with the higher value of the parameter *minScore*. This is due the fact that, the higher *minScore* is then there are more reference intervals for which $MS2DIP$ is not calculated. The best improvement was reached either by performing dynamic programming in each contig separately or by using a binary tree of runs instead of an array. The effect of the *minScore* parameter is minimal so these heuristics appear to be more robust. The reference and duplicated genome do not have the whole chromosomes so even simple heuristics save many calls of dynamic programming(Table 3.2). In the Fig. 3.6 is graph for *Yarrowia lipolytica* showing comparison of heuristic approaches. We can see that heuristic *minScore* shows the significant decrease in the execution time. The reason for it is, that we compare phylogenetically further organisms and less resemblance occurs between two genomes and therefore many of the best locations have very low score and could be filtered out. The addition of *contig's heuristic* then cause only a small improvement. This is visible in the table 3.3 where the *lowScore* heuristic causes that more than 90% of dynamic programming calls are omitted even for small values of *minScore* parameter. Heuristic approach based on *runs* does not improve execution time so much as it did in the previous simulation with *M. ingens* when the reference genome was phylogenetically closer to *M. magnusii*. On the other side, we may expect better performance by combining *run* heuristic with *lowScore* heuristic.

3.6 Generated data

There were two main reasons which forced us to design a model for generating simulated data. The first one was that available biological data are not 100% assembled into the chromosomes and even *Yarrowia lipolytica* with the whole chromosomes has a quite short genome compared to other organisms. Tests of the developed heuristics in longer data should better demonstrate the achieved time improvement. The other reason is, that we can set up the parameters of the model and generate data with different features. This enables us to test advantages of different heuristics for various types of data.

3.6.1 Model of data generator

We introduced few model parameters which are connected to biological events influencing the real biological data.

numOfGenes: A number of genes in the reference genome. This parameter designate the size of the reference and also of the duplicated genome.

endOfChrom: A probability that in the given position in the sequence of genes will be the end of chromosome or contig. The bigger value is the shorter chromosomes are.

delProb: A probability that a gene in the given position will be deleted from the chromosome. This parameter simulates the mutation of gene deletion which affects both the reference and duplicated genome.

numOfShuffle: A number of processes, where we randomly choose two position in the genome and flip the gene content between these positions. The goal is to violate the order conservation of genes. If both positions are from the same chromosome, then this process is equal to chromosomal inversion. However, in our model the positions could be chosen from different chromosomes, so we are trying to simulate a more complicated mutations when the structure of chromosomes can be disrupted.

A process of generating simulated data has the following steps:

1. According to the defined parameter $numOfGenes$ and probability $endOfChrom$ we generate the sequence with $numOfGenes$ different genes divided into chromosomes.

2. We create a duplicated genome which is the product of WGD and contains two copies of the reference genome form the previous step. Both reference and duplicated genome are perfectly conserved so far with no divergence between them.

3. We iterate through the entire duplicated and reference genome and gene in the each position is deleted with the probability $delProb$.

4. We perform the $numOfShuffle$ shuffles in both reference and duplicated genome. We select two positions uniformly and flip the content between the positions.

After the data generation, we test standard algorithm and developed heuristics.

3.6.2 Comparison between heuristics applied to generated data

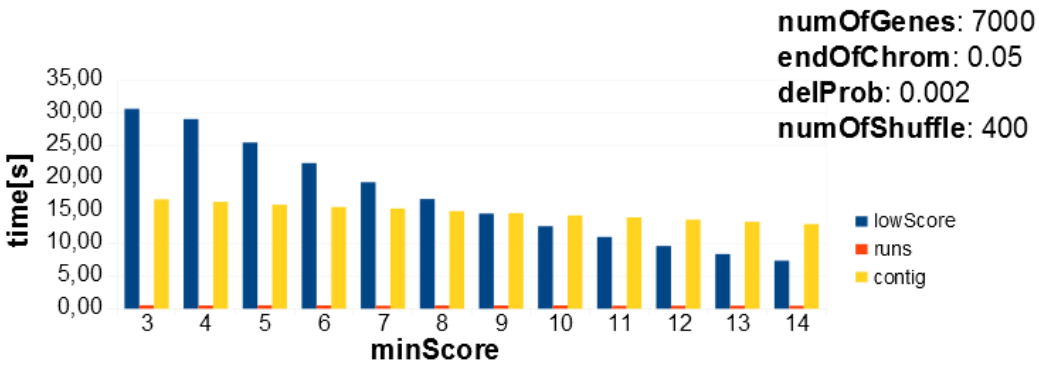
In this section we try to compare the behaviour of the different types of heuristics for various types of data. To shows the advantages and disadvantages of each heuristic improvement itself, we do not combine them together as we did in the biological tests where primary task was to solve the All location problem as fast as possible. We chose two basic characteristics of the generated data whose effects to heuristics we want to examine.

length of chromosome An average length of chromosome in the reference and in the duplicated genome. This characteristic is controlled by the parameter *endOfChrom*. The higher the value is then we have higher probability that chromosome will finish at current position during the data generation and subsequently the higher probability of shorter chromosomes. On the contrary, the smaller the value is, the longer chromosomes are generated.

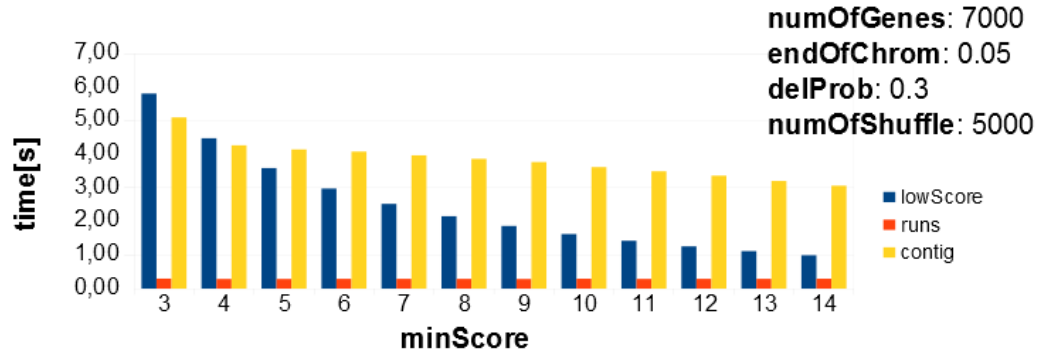
similarity A level of similarity between the reference and duplicated genome, i.e. how much two genomes differ in the gene content and gene order. This characteristic is set by adjusting two values which can increase or decrease how the reference and duplicated genome are similar to each other. The first one is *delProb* which has indirect correlation with the level of similarity and *numOfShuffle* which has a direct correlation, i.e. the higher value results in the higher similarity.

We set parameters in our model in a way that we got data with all combinations of the characteristics *length of chromosome* and *similarity*. Every combination has its graph of time results.

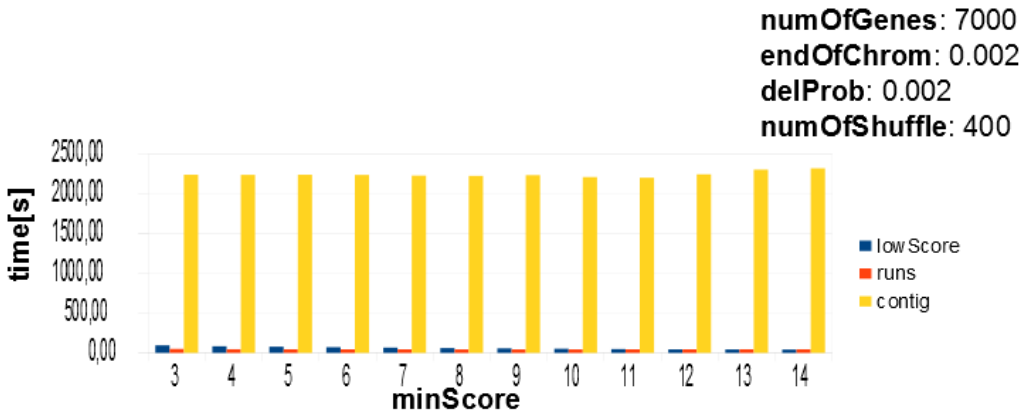
1. **small length of chromosome + high level of similarity**



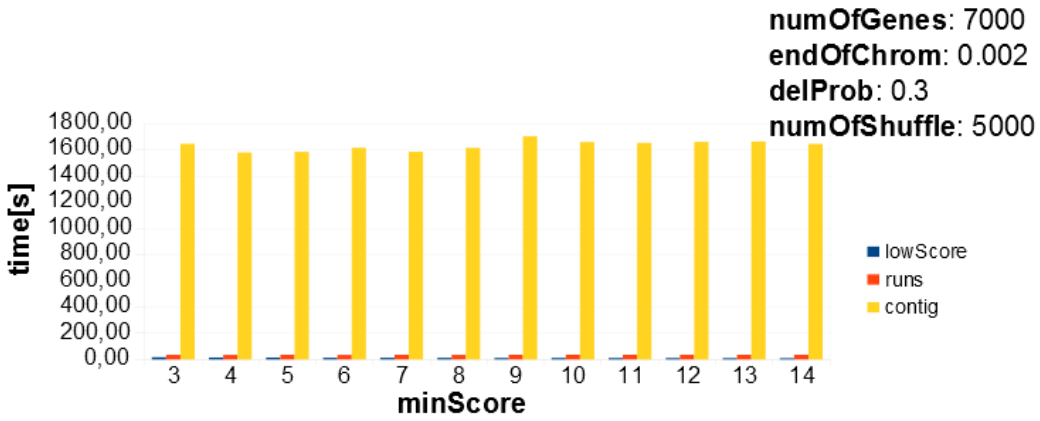
2. **small length of chromosome + low level of similarity**



3. **large length of chromosome + high level of similarity**



4. **large length of chromosome + low level of similarity**



From the tests in generated data we get these key results.

1. Contig heuristic is not suitable heuristic for data with the longer chromosomes, because the execution of $MS2DIP$ in long chromosome is pretty computational expensive. On the other side, this technique is more practical for data with the short chromosomes. The best results for *contig heuristic* are observed when the compared genomes are phylogenetically closed. The gene content is still quite conserved and the groups of genes are localized together only in few chromosomes, so we do not run $MS2DIP$ many times.

2. *lowScore* heuristic is observed to be the most advantageous especially in the data with the long chromosomes and long phylogenetic distance. In the data with the short chromosomes, this heuristic is few times faster in genomes with low level of the similarity. It is difficult to find best interval location with some significant score in genomes which are not similar enough, so many reference intervals are filtered out and problem $MS2DIP$ is not computed for them.

3. The heuristic *run* is the winner in almost all combinations excepts from the last one, where low level of similarity causes that duplicated genome is segmented into too many runs.

Conclusion

In our thesis we defined a formal model of approximate gene clusters in a duplicated genome which we referred to as the best interval location. We reduced this biological problem to the mathematical problem of maximum sum of two disjoint intervals and we designed a dynamic programming algorithm with linear time complexity. In addition, we proposed a scoring system which assigns a score to each interval location and helps us to filter out the less significant interval locations. We ran the algorithm on a biological data set which consists of two duplicated genomes and three reference genomes. As a result we identified the significant best interval locations in organisms which are assumed to have underwent a whole genome duplication event. In the last chapter we developed four heuristic approaches which improved the execution time of our algorithm. Heuristics proved their benefits especially in the tests with *Yarrowia lipolytica*, the yeast with genome consisting of whole chromosomes, where computational time was significantly reduced. In the very last part, we tested heuristic improvements on generated data with different features and showed their advantages and disadvantages.

There are few directions for extending this work. The first one is to develop a faster algorithm for the All location problem to get better time complexity than our $O(m^2n)$, where m is the length of reference and n the length of duplicated genome. We assume that it may be achieved by using more complex data structures. The second direction is to redefine the scoring function of the best interval location. In particular, the score of the interval location can be computed as symmetric difference between the gene content of the double location and the gene content of the reference interval, similarly as in the work of [K.11] for non-duplicated genomes. The third one is to define and calculate the statistical significance of the best interval location. As soon as we know how high the score of the statistically significant best location is, we can correctly set parameter $minScore$ and choose appropriate heuristic to calculate the All location problem as fast as possible.

Bibliography

[BBA+04] C. Brochmann, A.K. Brysting, I.G. Alsos, L. Borgen, H.H. Grundt, A.C. Scheen, and R. Elven. Polyploidy in arctic plants. *BIOLOGICAL JOURNAL OF THE LINNEAN SOCIETY*, 82:521–536, August 2004.

[BBCR04] Marie-Pierre Béal, Anne Bergeron, Sylvie Corteel, and Mathieu Raffinot. An algorithmic view of gene teams. *Theoretical Computer Science*, 320(2–3):395 – 418, 2004.

[BCG07] Anne Bergeron, Cedric Chauve, and Yannick Gingras. *Formal Models of Gene Clusters*, pages 175–202. John Wiley & Sons, Inc., 2007.

[BJ07] F. Bengtsson and Ch. Jingsen. Computing maximum-scoring segments optimally. Technical report, Research Report, Luleå University of Technology, 2007.

[BJMS09] Sebastian Böcker, Katharina Jahn, Julia Mixtacki, and Jens Stoye. Computation of median gene clusters. *Journal of Computational Biology*, 16(8):1085–1099, 2009.

[BS99] T. Blanchette, M. nad Kunisawa and D. Sankoff. Gene order breakpoint evidence in animal mitochondrial phylogeny. In *In Proc. of COCOON*, 1999.

[CDH+06] Cedric Chauve, Yoan Diekmann, Steffen Heber, Julia Mixtacki, Sven Rahmann, and Jens Stoye. On common intervals with errors, 2006.

[CGL03] Andre R O Cavalcanti, Zhenglong Gu, and Wen-Hsiung Li. Patterns of gene duplication in saccharomyces cerevisiae and caenorhabditis elegans. *Journal of molecular evolution*, 56:28–37, January 2003.

[Csu04] M. Csuros. Algorithms for finding maximal-scoring segment sets. *Algorithms in Bioinformatics*, pages 62–73, 2004.

[dur03] Tests for gene clustering. *J Comput Biol*, 10:453–482, 2003.

[EMNS98] Nadia El-Mabrouk, Joseph H. Nadeau, and David Sankoff. Genome halving. In *Combinatorial Pattern Matching*, pages 235–250. Springer, 1998.

[FH01] R Friedman and A L Hughes. Gene duplication and the structure of eukaryotic genomes. *Genome research*, 11:373–381, March 2001.

[GL] D. Graur and W.H. Li. *Fundamentals of Molecular Evolution*. Sinauer Associates, 2 edition, January.

[Glo] National Human Genome Research Institute's Talking Glossary.

[HG05] Xin He and Michael H Goldwasser. Identifying conserved gene clusters in the presence of homology families. *Journal of computational biology : a journal of computational molecular cell biology*, 12:638–656, July–August 2005.

[HKC09] M. Ha, E.D. Kim, and Z.J. Chen. Duplicate genes increase expression diversity in closely related species and allopolyploids. *PNAS*, 106(7):2295–2300, February 2009.

[HS01] Steffen Heber and Jens Stoye. Finding all common intervals of k permutations. In Amihood Amir, editor, *Combinatorial Pattern Matching*, volume 2089 of *Lecture Notes in Computer Science*, pages 207–218. Springer Berlin Heidelberg, 2001.

[J.94] Masterson J. Stomatal size in fossil plants: Evidence for polyploidy in majority of angiosperms. *Science*, 264(5157):421–424, 1994.

[Jah10] Katharina Jahn. *Approximate Common Intervals Based Gene Cluster Modelsg*. PhD thesis, Faculty of Technology, Bielefeld University, Germany, 2010.

[JZKS12] Katharina Jahn, Chunfang Zheng, Jakub Kováč, and David Sankoff. A consolidation algorithm for genomes fractionated after higher order polyploidization. *BMC bioinformatics*, 13 Suppl 1(Suppl 19):S8, 2012.

[K.11] Jahn K. Efficient computation of approximate gene clusters based on reference occurrences. *Comaparative genomics*, 6398:264–277, 2011.

[KFT] C. Kurtzman, J.W. Fell, and Boekhout T. *The Yeasts A Taxonomic Study*. Elsevier.

[L.05] Comai L. The advantages and disadvantages of being polyploid. *Nature Reviews Genetics*, 6:836–846, November 2005.

[LB97] I.J. Leitch and M. D. Bennet. Polyploidy in angiosperms. *Trends in Plant Science*, 2:470–476, December 1997.

[LPW05] G. M. Landau, L. Parida, and O. Weimann. Gene proximity analysis across whole genomes via pq trees. *Journal of Computational Biology*, 12:1289–1306, 2005.

[PBR+05] Sophie Pasek, Anne Bergeron, Jean-Loup Risler, Alexandra Louis, Emmanuelle Ollivier, and Mathieu Raffinot. Identification of genomic features using microsyntenies of domains: domain teams. *Genome research*, 15:867–874, June 2005.

[Sou74] Shirley W. Soukup. Evolution by gene duplication. s. ohno. springer-verlag, new york. 1970. 160 pp. *Teratology*, 9(2):250–251, 1974.

[ST04] Stoye J. Schmidt T. Quadratic time algorithms for finding common intervals in two and more sequences. In SuleymanCenk Sahinalp, S. Muthukrishnan, and Ugur Dogrusoz, editors, *Combinatorial Pattern Matching*, volume 3109 of *Lecture Notes in Computer Science*, pages 347–358. Springer Berlin Heidelberg, 2004.

[SZ12] D. Sankoff and C. Zheng. Fractionation, rearrangement and subgenome dominance. *BMC Bioinformatics*, 13, 2012.

[UY00] T. Uno and M. Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26:2000, 2000.

[WS09] Robert Warren and David Sankoff. Genome aliquoting with double cut and join. *BMC Bioinformatics*, 10(Suppl 1):1–11, 2009.

[ZZAS08] Chunfang Zheng, Qian Zhu, Zaky Adam, and David Sankoff. Guided genome halving: hardness, heuristics and the history of the hemiascomycetes. *Bioinformatics*, 24:i96–104, July 2008.