# Variants of the Consecutive Ones Property: Algorithms, Computational Complexity and Applications to Genomics

by

**Ashok Rajaraman**

M.Sc., Simon Fraser University, 2011
B.Tech., Indian Institute of Technology Roorkee, 2009

Dissertation Submitted in Partial Fulfillment
of the Requirements for the Degree of

Doctor of Philosophy

in the
Department of Mathematics
Faculty of Science

## APPROVAL

| | |
|---|---|
| **Name:** | Ashok Rajaraman |
| **Degree:** | Doctor of Philosophy (Mathematics) |
| **Title of Thesis:** | *Variants of the Consecutive Ones Property: Algorithms, Computational Complexity and Applications to Genomics* |
| **Examining Committee:** | **Chair:** Dr. Jonathan Jedwab<br>Professor, Department of Mathematics |
| | **Dr. Cedric Chauve**<br>Senior Supervisor<br>Professor, Department of Mathematics |
| | **Dr. William S. Davidson**<br>Supervisor<br>Professor, Department of Molecular Biology and Biochemistry |
| | **Dr. Bojan Mohar**<br>Supervisor<br>Professor, Department of Mathematics |
| | **Dr. Tamon Stephen**<br>Supervisor<br>Associate Professor, Department of Mathematics |
| | **Dr. Cenk Sahinalp**<br>Internal/External Examiner<br>Professor, Department of Computing Science |
| | **Dr. Lucian Ilie**<br>External Examiner<br>Professor, Department of Computer Science, University of Western Ontario |
| **Date Defended:** | 15 January 2015 |

## ABSTRACT

Genome mapping problems in bioinformatics can be modelled as problems of finding sequences of vertices in hypergraphs, subject to consecutivity constraints. These problems are related to the *consecutive ones property*, a well-studied structural property on binary matrices. Many variants of this property have been introduced to include subtleties in the model, such as upper bounds on the number of times a vertex may appear in a sequence, the distance of the input from having the property, and confidence values for the consecutivity constraints. Most problems involving these variants are intractable, and efficient solutions call for restrictions on the structure of the input, exponential time algorithms, or approximations. The following document discusses these problems, from both a theoretical perspective, and from the genomics point of view.

We encounter two main classes of problems, divided into models which account for repeated elements in genomes, and those which do not. Orthogonally, we divide the problems into decision and optimization questions. For models with repeats, we discuss when the given input can be used to reconstruct the genome map of interest, and if we can discard a minimal set of encoded consecutivity information from the model to obtain an input which can be used to reconstruct this genome map. We also discuss the problem of ambiguity introduced by repeats, and introduce the concept of *repeat spanning intervals* in order to address them. We show that the problem of optimizing over the set of repeat spanning intervals is NP-hard in general, and give an algorithm when the intervals are small. In models without repeated elements, we discuss the problem of optimization by finding a solution that minimizes the distortion in the consecutivity information, by generalizing the concepts of bandwidth and minimum linear arrangement to hypergraphs. We design approximation algorithms for two versions of the latter problem, with an approximation ratio of $O\left(\sqrt{\log n}\log\log n\right)$.

Finally, we provide details of implementations of some of the methods developed for genome mapping and scaffolding on ancestral genomes. We include results on real data for the genome of the Black Death agent, and for ancestral *Anopheles* mosquitoes.

**Keywords:** consecutive ones property; genome reconstruction; repeat ambiguity; vertex ordering; approximation algorithms

## DEDICATION

*To Pāti and Pāpu.*
*I hope you would have been proud.*

## ACKNOWLEDGEMENTS

There is usually a set order or presentation in a dissertation where acknowledgements are concerned. The body of the document itself may be scrambled beyond recognition, but this section is held sacrosanct. So I shall resist my inner impulse to shock and awe, and respect the time-tested traditions of academics the world over.

In prime position would have to be my senior supervisor, Cedric, who made sure that I started and finished the work being presented in this document. All graduate students procrastinate; it is only the supervisor who holds the ability to make them work past it. I would also like to thank him for introducing me to mathematical problems in bioinformatics. His approach to these problems provided an orthogonal viewpoint to my own. Every research project requires a good foil, and an important part of my graduate studies was spent in his office bouncing ideas off each other. This, of course, is besides the academic and financial support that he was generous enough to provide me.

I could not have done half the work I have if it were not for my wonderfully patient collaborators. Brad, Eric, Jáno, João, Murray and Yann had to put up with what I suspect were frequent and frustrating mistakes on my part, not to mention paranoia about the validity of proofs, code and experiments. So I tender my apology for being a trying colleague.

In a similar vein, the other members of my supervisory committee, Willie Davidson, Bojan Mohar and Tamon Stephen, took an interest in the progress I made, and were available to discuss the topics I was working on. Between them, they cover a vast breadth of knowledge and experience, and provided useful suggestions during my time here.

I owe a lot to the Pacific Institute for the Mathematical Sciences and the Dean of Graduate Studies Office at SFU, who provided most of the funding through my time here. Thanks to them, I was able to focus on the research topics with little worry as to my financial position as a graduate student. I would also like to add to this list the Department of Mathematics, who have occasionally provided me with teaching positions, and the various course coordinators I have worked with: Petra, Justin, Brenda, Keshav and Elena.

Staying with the Department of Mathematics, a very sincere thanks to all the great people associated with it, faculty, staff and graduate students. I had a great time, and enjoyed working with them. Not to mention the free cookies and coffee at seminars, and the daily coffee routine with my office mates: Bamdad, Jeff, Lee and Will.

Going farther back in time, a very sincere thank you to the ‘Lords’, people who I know will mock me for it, needing no gratitude between friends. Their first words when I told them I am going into mathematics was to tell me I was crazy. Which were immediately followed by them telling me that its the best decision I had ever made.

Finally, to my family, Amma and Appa, Anna and Manni and little Surabhi, to whom I owe far more than I can ever give back, and always have my eternal love. Thanks for everything.

# Contents

# List of Tables

# List of Figures

# Notation

| | |
|---|---|
| $\mathbb{N}$ | Set of natural numbers |
| $\mathbb{Z}, \mathbb{Z}_{\geq 0}$ | Set of integers, and those which are greater than or equal to 0 |
| $\mathbb{R}, \mathbb{R}^+, \mathbb{R}_{\geq 0}$ | Set of all real numbers, those which are positive, and those which are greater than or equal to 0 |
| $[n]$ | Set $\{0, \ldots, n-1\}$ |
| $\lfloor x \rfloor$ | Largest integer smaller than $x$ (the *floor* of $x$) |
| $\lceil x \rceil$ | Smallest integer larger than $x$ (the *ceiling* of $x$) |
| $\|S\|$ | Size of a set $S$ |
| $\mathbf{x}, x_i$ | A vector, and the $i^{th}$ component of the vector |
| $\mathbb{1}_n$ | The all 1's vector in $\mathbb{R}^n$ |
| $M, m_{ij}$ | A matrix, and the entry in $M$ at the $i^{th}$ row and the $j^{th}$ column |
| $x \vee y$ | Logical **OR** operation between binary variables $x$ and $y$ |
| $x \wedge y$ | Logical **AND** operation between binary variables $x$ and $y$ |
| $\overline{x}, \neg x$ | Logical negation of the binary variable $x$ |
| $\Sigma^*$ | Set of all linear sequences over the alphabet $\Sigma$ |

All logarithms in the text are taken to base 2, unless stated otherwise.

# Part I

# Computational Genomics and Mathematics

# Chapter 1

# Introduction

Modern genetics has been characterized by fast and cheap acquisition of reasonably high quality data [46,139,166,192]. Compared to 20 years ago, the amount of data available to us is quite extraordinary. With an ever improving understanding of the working of genetics, such data has become an invaluable source of information in many fields in molecular biology[1].

From a computational point of view, it is important to have better, faster frameworks in place to deal with the massive amounts of data being made available for each of these problems [93,169,187]. In order to interpret the data, there are many challenges to overcome: statistical modelling of the data, storing genomes that are billions of nucleotides long, developing algorithmic techniques etc.

Computational biology evolved to address the unique computational challenges posed by genetic data. The field seeks to model and infer quantitative driving factors which can explain biological, possibly qualitative data arising from the study of genomes. There are a number of questions that fall into its ambit.

- What factors drive evolution? Is there a mathematical model that explains the pattern of evolution which shaped today's genomes?
- What does the genome of an organism actually look like? Where are the genes and other functional elements on the genome, and how are they organized?
- What are the driving genetic factors for diseases?

These are but a few broad questions which have attracted the attention of numerous researchers in the field. They tie into the understanding of evolution, the explanation of disease susceptibility and pathogeneticity, and other major topics in biology that have deep roots in genetics.

[1]We assume that the reader is familiar with the basic concepts in genetics: that of the DNA molecule, genes and the genome.

## 1.1 A quick overview of computational biology

Computational biology as a field has been involved in more than merely genetic data. It refers to the broad range of computational tools used in almost any sub field of biology, such as modelling biological systems [153], or studying brain function [136]. Our interest lies in the application of computational methods to genomics.

The last three decades have been particularly significant in two respects: the large-scale availability of a genetic data, and the computational resources at hand to analyze them. The development of BLAST [5] may be considered to be a turning point in the way computational genomics was approached. This was one of the first readily available tools that could handle genome sequence data with relative efficiency.

Since then, a lot of effort has been put into developing time and memory efficient software for many problems in genomics [110, 196, 209, 214]. Side-by-side, many concepts from theoretical computer science have made their way into the genomics community, and the rich exchange of ideas has benefited both fields [2, 18, 19, 20, 184].

It is provident that computer science became an established research area around the same time as the new developments in genetics. As a consequence, researchers can now call on massive computational resources in order to analyze and interpret data [16, 103, 126]. However, such analysis is limited by theoretical limits on computability, and effective limits on the computational power available. In order to analyze data, we need a mathematical construct, a *model* for the data. But operations within the model may not allow efficient solutions, and researchers are forced to sacrifice at many levels: the accuracy of the eventual output of their models and the processes, the computational costs in terms of time and memory used, or even the biological relevance of the computational output. Indeed, what the mathematical model predicts as the optimal solution to a formal problem may be far removed from the biologically correct solution. At the same time, it is necessary to know which problems are computationally hard: the hardness of a problem can point to the direction needed to be taken to address it, via heuristics, approximations or computational power. It also isolates the complexity in the model, and allows us to draw a correspondence between the biological features being modelled and their effect on problem complexity.

The aim, then, is a delicate balance of model accuracy and practicality. It is a balancing act that has been mastered to some effect [37, 51, 145], but with growing computational resources and a burgeoning repository of data, every model has to be updated at an ever increasing pace. The dissertation may be seen as an exercise in this task: the development, analysis and application of a model for a specific purpose in computational genomics.

## 1.2 The basics of genome structure and organization

One of the main motivations for the work in this dissertation is the organization of genetic material in genomes. Knowledge of the structure of the genome carries a lot of value. It is predicted to be an important indicator of various properties of an organism, such as pathogeneticity in microorganisms [32], the robustness of crops to droughts [213], and neurological and cellular functions in organisms [194].

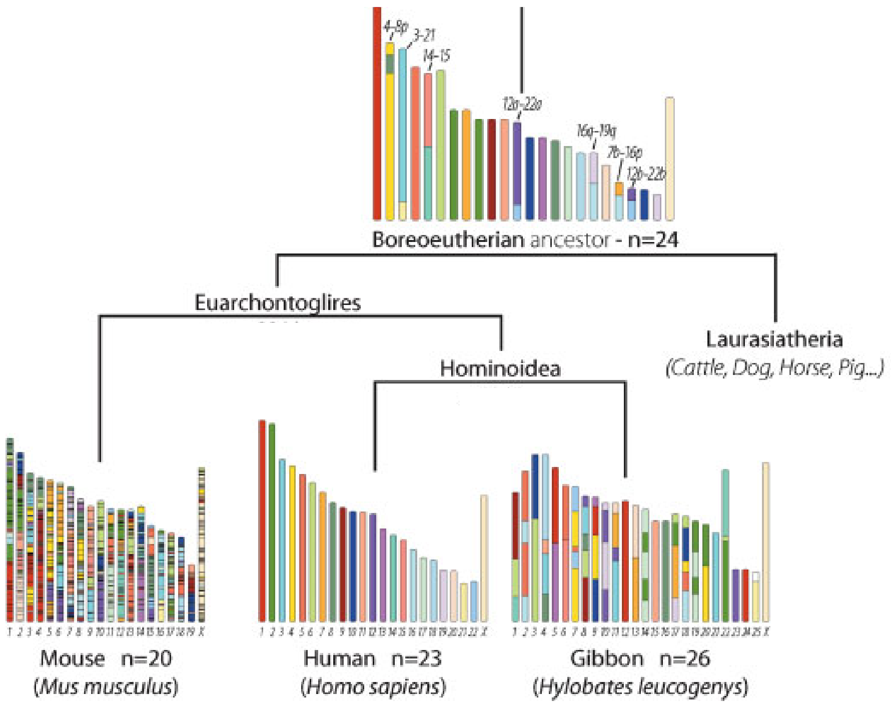
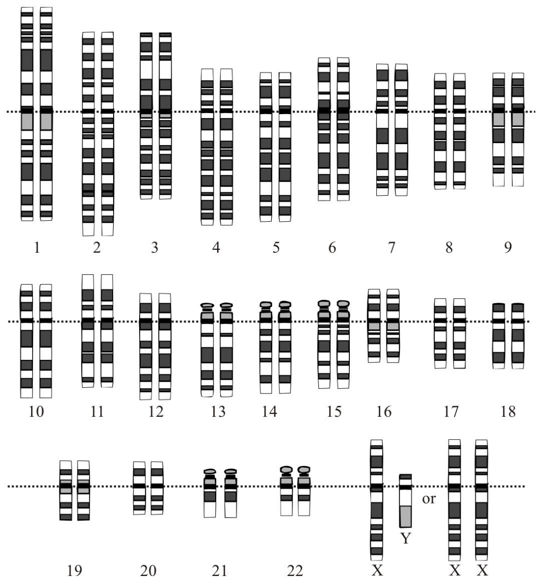
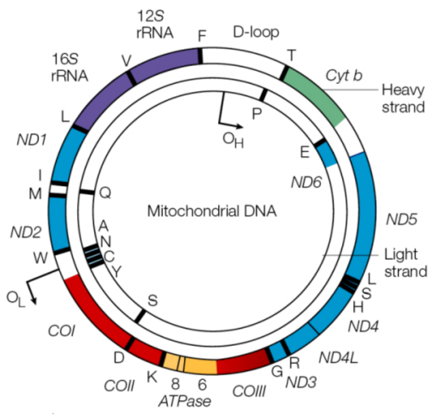


Figure 1.1: An illustration of how genome organization evolves in certain mammals [151]. Each linear segment indicates a chromosome at that species in the tree. The human chromosomes are coloured with a single colour each. Genome segments in non-human chromosomes are coloured according to the human chromosome in which they have a homolog, i.e. there is a genome segment in the human chromosome which is the same as the one indicated in the other species, up to small variations. The degree of multicoloured chromosomes in a species gives a rough idea of how different the genome organization is compared to the human genome.

A second motivating factor for the dissertation is the understanding of the evolution of genomes [63]. Genome evolution is a complex, incompletely understood process. At the nucleotide level, evolution can take place by changing, deleting or inserting a nucleotide in a genome sequence. But, on a large scale, evolution occurs through change in genomic segment organization. Large parts of the genome may be deleted from one part of the genome, only to be reinserted elsewhere. Such segments may be duplicated through evolution, or deleted altogether. Genomic segments may even be reversed during evolution. Figure 1.1 provides


(a) The nuclear human genome.



(b) The human mitochondrial DNA.

Figure 1.2: A representation of the chromosomes in the human genome. Figure 1.2a represents all 46 linear chromosomes in the main human genome [162]. Of these, there are two copies of 22 chromosomes. Figure 1.2b, taken from Chial and Craig [47], shows the human mitochondrial DNA, an example of a circular DNA molecule. Note the coloured segments on the molecule; these are *markers* that have been recognized to correspond to genes.

an illustration of genome organization evolution in some mammals. The operations involved in genome evolution are complex, and there are various models which are used to explain how a genome may evolve [10,19,79]. What is certain is that understanding genome organization is as important to understanding evolution as single nucleotide substitutions. To that effect, we now briefly discuss the main details of genetic material organization.

The genome of an organism can be visualized as a sequential organization of genomic segments. This presupposes that we have well-defined genomic regions, which we will call *markers*. Markers are linear DNA sequences which are inferred to occur in the genome. One may think of markers as a proxy for genes in the genome. The exact definition of a marker is usually made on a case by case basis, as we shall see in the following sections. The major difference in each case is a question of scale: sometimes markers are defined as very short genome segments, while other times they may be large DNA sequences composed of many of the short segments.

Markers in a genome are organized into a set of sequences, which are called *chromosomes*. Chromosomes may be a set of linear sequences, as in most multicellular organisms including humans, or may be a single circular sequence, as in many bacteria. Figure 1.2a shows the set of linear chromosomes found in humans.

Apart from chromosomes, there may be small circular DNA sequences present in certain parts of the cell. When these sequences are present in *mitochondria*, locations in eukaryotic cells that convert chemical energy from food into usable energy, the DNA sequence is called the *mitochondrial DNA* or mDNA. An example is provided in Figure 1.2b. In bacteria, however, there may be other circular DNA sequences present in the cell. These sequences are called *plasmids*.

Genome organization evolution, once markers are suitably defined, is characterized by certain operations on genomes. A few such basic operations are illustrated below.

1. Markers in an ancestor may be duplicated, inserted or deleted in the course of evolution.

2. Contiguous sequences of markers in the ancestor may be moved to another location within the genome sequence. This includes movement from one chromosome to another.

3. A contiguous sequence of markers may also be reversed within a single chromosome.

The process by which these events occur is hard to determine. For example, the movement of a contiguous sequence of markers may also be due to the sequential deletion of the markers from one location followed by sequential insertion of the same markers into the location at which the contiguous sequence is observed to have moved. In the absence of precise details of evolution, computational biologists often fix a model according to which the genome evolves, and base their inferences on the properties of this model [21, 82, 184, 217]. However, the mechanism of evolution will not be an important topic of discussion in this manuscript.

## 1.3 Genome maps and mapping problems

The title of this document mentions the term 'genomics'. The problems and methods presented here are primarily aimed at taking an algorithmic viewpoint to the basic question of visualizing genome organization, given limited data. Briefly put, the aim is to take a set of DNA sequence elements, which form the markers, and find their ordering and orientation along chromosomes of interest. These genomic segments may be genes, or physical locations defined either experimentally or through some computational processes. The order constructed by this procedure is called a *genome map*.

Constructing a genome map is an iterative procedure. Once we find a genome map of a certain genomic region, we can combine this with other maps on the same genome in order to create a larger, more complete map. This procedure of combining limited map information into larger maps is called *genome mapping*[2].

[2]Note that 'genome mapping', in a modern context, often refers to the process of aligning reads or contigs to a genome sequence. Our use of the term here is motivated by historical reasons, recalling the concept of physical genome maps.

At a fundamental level, all the problems we discuss can be thought of as extensions of the classical genome mapping problem. But they are usually classified by the scale at which the map construction is taking place, and the nature of the available data. Each problem can be broken into a two-part pipeline: data acquisition, and the mapping procedure.

**Data acquisition.** In order for us to get into the algorithmic questions at hand, we first need data which can be suitably represented in a mathematical model. The first piece of information is a set of well-defined *genomic markers*, which, as stated before, are linear segments of DNA which are inferred to appear on the genome of interest. There are a few different sources for finding genome segments that can be arranged into a genome map, differing by the exact mapping problem being encountered. In each case, the data given either defines, or is processed in order to define genomic markers. It is the organization of these markers on the genome of interest that we wish to reconstruct.

A common occurrence in genomes is the presence of duplicated sequences of nucleotides. If there is a large duplicated sequence, it is likely that the duplication was part of an evolutionary process, rather than *de novo* creation of the same sequence twice in the genome. This sequence might be inferred as a marker, and the number of occurrences of the marker on the genome is called its *copy number*. A marker with a copy number greater than 1 is often called a *repeat*. In order to get a complete picture of the genome organization, we need to associate each marker with a copy number, and find a genome map in which every marker appears at most as many times as its copy number. Repeats, we will see, pose unique challenges to genome mapping problems.

Obtaining the copy number of a marker is not an easy problem. Since we do not have the genome sequence at hand, the only way we can get an idea of the copy number is through the process of defining the markers itself. But it is hard to differentiate between 2 different occurrences of the same marker on a genome, because we usually do not know the locations of the DNA sequences defining the markers on the genome. We will discuss how to infer copy numbers for markers in palaeogenomics later in this chapter.

The other important piece of information which is needed in a mapping problem captures how the markers are arranged on the genome of interest. This is represented as sets of markers, which are understood to have been present consecutively on the genomic sequence. This *synteny information*, representing the colocalization of markers on the genome, serves a guiding window. If we are able to see the immediate neighbourhood of a marker, or have a set of mutually overlapping markers, we may have an idea of where the marker should be placed in the genome. The principle is that we should be able to piece together the entire genome map using such information.

There are many ways of finding which markers are colocalized on the genome of interest, i.e. we have methods to determine the neighbourhood of the markers. Such information can be inferred through experimental techniques (eg. hybridization experiments [167]), as well as by employing computational techniques (eg. identifying ‘common intervals’ [43,110]). We elaborate on some of these techniques, and the import of the data, later in this chapter, with examples.

**Mapping procedure.** From the point of view of this manuscript, the mapping procedure is the main topic of discussion. Data can be extracted using both experimental and algorithmic techniques, depending on the source and the paradigm employed. But in order to efficiently use this data to obtain a genome map, one has to take into account possible errors and missing data. The methods we discuss concern two problems: how do we check if the data is error-free, and how do we best optimize the data if it contains errors, while making sure that it conforms to a number of constraints imposed by the genome structure.

We highlight here various types of mapping problems commonly encountered in computational genomics, and the data available as input to each of them. We also briefly discuss some techniques to acquire such data in order to give the reader a feel for the general scope of the techniques presented in the manuscript.

### 1.3.1 Physical mapping

In physical mapping, the genomic segments are defined as *probes* on a genome, short DNA sequences that are extracted from the genome. Apart from this, we are given *clones*, longer genome segments which might contain many probes expected to occur close together on the genome. It is possible to know if a probe is contained in a clone using experimental techniques. The problem is to find the ordering and orientation of the probes on the source genome, while making sure that the order of the probes is consistent with their order within each clone containing them. The order reconstructed is called a *physical map*. Early approaches to investigate the chromosomal organization of genomes relied on the computation of such genome maps [2,144], and such maps are still being constructed [61].

In physical mapping, the markers are defined as probes extracted from the genome [119], and the clones define synteny information between the markers. Following this, we wish to find a genome map in which every probe appears, possibly with some copy number restriction. A further restriction is that the neighbourhood of a probe in the map must match the neighbourhood defined by the clones. Synteny information, which is clones containing a set of probes, in this case can be obtained through hybridization experiments [2]. Such experiments also allow the definition of the probes as particularly short sequences obtained through the experiments.

### 1.3.2 Genome assembly

Modern sequencing techniques on genomes focus on finding many short DNA segments called *reads*. Since these reads are often very short, they may occur hundreds of times on the genome being sequenced, and they may overlap. These reads are combined, using the overlap information, into larger, contiguous DNA sequences called *contigs* [214]. The process of combining reads into contigs is called *genome assembly.* Ideally, the goal of genome assembly is to find a single large contig, representing an entire chromosome. In practice, it is more common to find a set of contigs, which often overlap, and are subjected to further processing.

The availability of fully assembled genomes is key to understanding the functional organization of genomes, as well as the processes that govern their evolution. With the development of modern sequencing technologies, we are flooded with a large amount of data [193], and the challenge has been to develop genome assembly frameworks. Such frameworks, like the de Bruijn graph [145, 173] and the string graph [158], form the basis for most modern assemblers [160, 174]. In turn, assembled genomes prove to be useful in furthering more assembly, mapping and scaffolding projects [94]. This cascade of information is highly beneficial to the genomics community at large.

There are two major paths taken in genome assembly. Sometimes, reads extracted directly from the genome of interest must be assembled into contigs. If this is the first time that such reads have been made available, or if previous assemblies are judged to be very poor, there is no template which tells us what the genome looks like, and the reads must be assembled *de novo.* On the other hand, if there is a usable template present, one can use this as a guide for how the reads must be placed. This forms the basis for *reference guided assembly* algorithms. Both *de novo* assembly [121, 125, 126] and reference guided assembly [189] using short reads are important topics of research in modern genomics.

For assembly problems, markers can be defined as short reads, which need to be assembled into contigs [116, 145, 159]. For example, they may represent reads in an overlap graph [157], or $k$-mers in the de Bruijn graph approach [108, 214]. Synteny information can be extracted from long read information [57,121,143,196], or, in the case of reference guided assembly, through alignments onto the reference genomes [214]. However, algorithms used for assembly are often based on different theoretical concepts from those we will discuss.

### 1.3.3 Genome scaffolding

We stated that *contigs* are the end products of genome assembly. These are relatively large DNA sequences that are inferred to lie on the genome of interest. Contigs can be linked to obtain even larger DNA sequences, perhaps even an entire chromosome. Such linking can be done using data such as paired-end sequence information. Paired-end reads consist of two sequences of similar length sequences from different ends of the same DNA molecule, the sequences being separated by some bases in between. Information like this allows us to infer which contigs occur next to each other on the genome, which of them overlap etc. The process of orienting and ordering contigs on the genome is called *scaffolding*, and every continuous order of contigs obtained is called a *genome scaffold*. One would also like to predict the sequence between two consecutive contigs in a scaffold, and if these contigs overlap.

For scaffolding problems, the markers are used to represent contigs [107], which we wish to organize into a large genome scaffold. Synteny information is obtained by using mate-pair libraries [12,33,66,89,107,180,183], existing genome maps to which we can map the contigs [127], or comparison with one or several closely related genomes [94,106,118]. The last method is one which we will encounter again, albeit for a different problem.

### 1.3.4 Palaeogenomics

Another challenge that computational genomics is faced with is the prediction of the genome maps of extinct organisms. The problem here is that DNA decays rapidly, and ancient DNA sequencing is accompanied by high error rates, and is often highly fragmented [25]. One way around this is to use the genomes of the extant descendants of the organism under consideration [170]. This is the idea behind the distinction between *de novo* techniques and *comparative* techniques. The field of comparative genomics seeks to find information about a genomes, or many genomes, by studying genomes that are closely related to it. These genomes are used as references against which the data can be compared, and which can be used to construct the genome map. The principle behind the application of this field in palaeogenomics is an elementary hypothesis in evolution: two genomes belonging to organisms which are sufficiently close from an evolutionary perspective must have conserved features. Comparative methods play a vital role due to the absence of good quality data, which renders most *de novo* techniques moot (for a rare exception, see [62,67]).

The reconstruction of the organization of ancestral genomes using extant genomes has received great interest during the last few years, due to the increasing number of sequenced and assembled genomes and to major methodological advances. Most reconstruction techniques follow one of two paradigms.

**Global parsimony.** The principle of global parsimony is to assume that genome evolution occurs through a set of evolutionary events. For these purposes, a genome is considered to be a string of markers. The objective is to find a set of ancestral gene or marker orders that minimizes the total number of such events required to obtain the observed extant genomes. This defines a combinatorial model of evolution, which admits a computational analysis of genome evolution.

The class of events, which act on a given genome to yield a new genome, are generally called *genome rearrangements*. Such events were originally based on real-life evolutionary events such as *chromosome fission and fusion*, but the repertoire of events has grown to include, among other events, *reversal*, *transposition*, *translocation*, *duplication*, *loss*, *insertion*, *the double-cut-and-join* [82] and the *single-cut-or-join* [79]. These need not be true evolutionary events, but they are used as proxies to determine the evolutionary distance between genomes. Some of these events do not change the content of the genomes, i.e. the input genome and the genome it evolves into vary only in the marker order and orientations. However, since genome content may indeed vary through evolution, events like insertion, loss and duplication are regarded as important additions to a model [36].

The distance from one genome to another depends on the events allowed in a given model. After fixing the combinatorial model of evolution, we can define the following problems.

1. *Genomic distance*: Given two genomes, compute the minimum number of operations needed to transform one genome into the other.

2. *Genome median*: Given three different genomes, find a fourth genome such that the sum of the genomic distances from this genome to the other three is minimized.

3. *Small parsimony*: Assume we are given a phylogeny (a binary tree), where each leaf is labelled with an extant genome. Compute a labelling of the internal nodes with genomes such that the sum over all edges of the distance between two genomes at adjacent nodes is minimized.

4. *Large parsimony*: Given a set of genomes, compute a phylogeny (a binary tree) with the input genomes at the leafs such that the small parsimony score is minimized.

Global parsimony has been studied for a long time, and there are many results on the tractability of such methods [75, 79, 184], as well as intractability results [31, 198]. In particular, problems other than genomic distance are intractable for most models of global parsimony [82]. Furthermore, even distance problems become intractable when events which change genomic content are included [27, 36, 210]. Recently, tractability results have been extended to include whole-genome duplication events [109, 185].

Beside the mathematical study of such techniques, there has been considerable work done using these techniques in order to reconstruct ancestral genomes [21, 26, 88], including those for the rosid phylogeny [217], and for coffee [56]. The main drawback of these methods is that they generally do not consider the reconstruction of ancient genomes with repeated markers. As stated before, very few results exist on the inclusion of duplications, insertions and deletions in the underlying models outside of whole genome duplications [109], other than a few heuristic approaches [73], and limited results [27].

We do not study global parsimony based approaches in this document, though it is important to keep them in mind as an alternative approach to ancestral genome reconstruction.

**Local parsimony.** The alternative to global parsimony is the model-free, local parsimony approach to ancestral reconstruction [135]. In this method, we try to reconstruct *a single* *ancestor of interest*, while keeping the exact model of evolution hidden and employing no optimization criterion based on evolution at the level of the data set. This approach has gained a lot of support, since it allows comparatively efficient reconstruction algorithms for ancestral maps [1, 17, 38, 43, 135, 152]. It has recently been used on a number of datasets for reconstructing ancestral genomes [155, 164].

Palaeogenomics using the local parsimony approach forms the main motivation for the problems we address in the manuscript, and we will expand on the approach we take to it, and how we obtain the data in the next section.

## 1.4 A framework for reconstructing palaeogenomes

The framework we use to address questions in palaeogenomics is a comparative, local, model-free approach. This approach [19, 135], focusses on the reconstruction of a single ancestor in a phylogeny using the genome sequences of several closely related extant species. Furthermore, it does not rely on a model for the evolution of the ancestral genome into the reconstruction procedure. Techniques that include such a model are often aimed at reconstructing more than one ancestral genome, minimizing some well-defined genomic distance over the phylogeny [217]. On the other hand, the model-free approach operates at a local scale, without defining a genomic distance measure. Such a technique has been modified and adapted for various purposes, especially ancestral genome mapping [17, 43] and scaffolding [177].

The process, as before, is to first get the data, and then proceed with the mapping procedure. In this case, the preferred method for extracting data is to use the *comparative approach*. This assumes that we already have some processed input data- the genomes of extant species related to the palaeogenome we wish to study, and the phylogeny of these species, including the ancestor, in the form of a species tree. In the context of our work, the species tree is usually obtained through sequence comparison techniques [150]. We do not consider the actual problem of obtaining phylogenetic information, but make note of the fact that this information is independent of the syntenic information we seek to use to reconstruct the ancestor.

Using this input, we will explain how to obtain the marker and synteny data in the following subsections. This is followed by a short description of the mapping procedure.

### 1.4.1 Obtaining data: markers and copy numbers

The genome mapping problem, we stated, was to find an ordering of genomic regions along the chromosome(s) of a genome of interested. In the case of ancestral genome reconstruction, they may be defined as evolutionarily conserved regions in closely related extant genomes, including descendants. Such regions may be obtained through whole genome alignment of

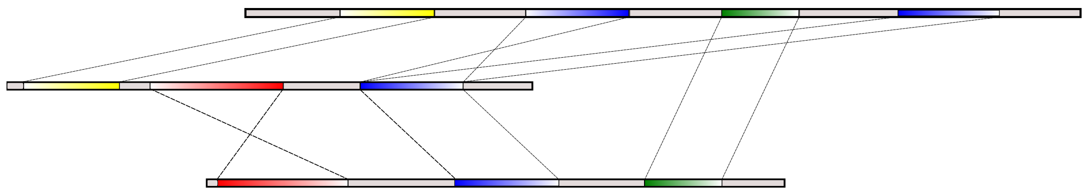


Figure 1.3: Inferring genomic markers through multiple sequence alignment. The linear sequences shown represent chromosomes of different but closely related species. Coloured regions represent loci which have been inferred to be homologous to loci in other species. The gradient of the colour indicates directionality of the sequence found with respect to an agreed upon reference. This is an idealized setting; in real data, there may be partial alignments, overlapping alignments, and mismatches.

extant genomes [43, 135, 170], the analysis of gene families [14, 38], or by sequencing the ancestral genome [177].

**Example.** In evolutionary genomics, readily coming by high quality sequence-level information for ancient genomes is complicated by the high rate of DNA decay. As a result, any data we extract through direct sequencing is quite error prone. This problem is bypassed by using comparative methods for obtaining data. One of the many ways to define markers in ancestral genomics is via multiple sequence alignment of the extant genomes. This is a standard and particularly useful comparative technique, since it does not require information about the ancestral genome apart from its phylogeny. The alignment recognizes regions on the extant genomes that are similar to each other. Using suitable filtering steps, such as choosing alignments having high identity and long length, it is possible to define ancestral markers [60], as exhibited in Figure 1.3.

Ancestral copy numbers can be found by assuming a parsimonious evolutionary scenario. Figure 1.4 shows a classical way of inferring the copy numbers of a marker in an ancestor, once the markers are defined. The number of copies of a marker in the extant species is known through the sequence information available for them. For the ancestor, the copy numbers may be calculated by minimizing the number of evolutionary gain-loss events along the branches of the species tree [52].

### 1.4.2 Obtaining data: co-localization information

For ancestral genomes, phylogenetic information again provides the basis to define syntenies between markers. Using the extant genomic sequences, and the locations of markers on these sequences, it is possible to use parsimony or probabilistic methods to infer candidate sets of markers that are expected to appear consecutively on the ancestor [43, 110, 170, 177]. It

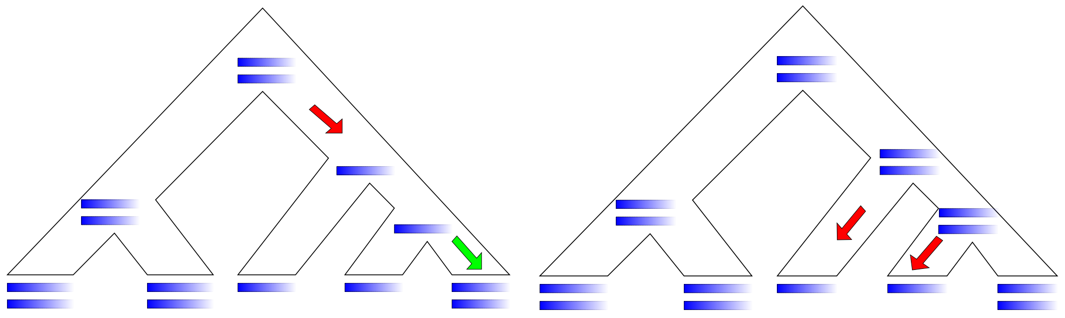


Figure 1.4: Inferring copy numbers using parsimony on a phylogeny. The number of copies of the blue marker in the extant species are labelled at the leaves of the phylogeny. The two trees show alternative evolutionary scenarios, one in which there are a single loss (red arrow) and gain (green arrow) of a marker copy, and one in which there are two losses (red arrows).

is even possible to use the phylogenetic information to include confidence measures for the inferred sets [135].

**Example.** For ancient genomes, we can use comparative methods in order to extract colocalization information from related extant genomes. We treat a possible adjacency between two markers as a binary variable. We know whether this adjacency does or does not occur in each of the extant species. Then, we use traditional parsimony to infer if the homologous ancestral markers were also adjacent in each of the ancestral genomes.

Larger syntenic information between markers can also be inferred through comparative techniques. Often, if we are considering a synteny consisting of three or more markers, instead of looking for a precise ordering of the markers in the synteny, we are interested if the same synteny is present in the ancestor up to internal variations in marker order. In this case, we treat the set of markers in the synteny as a binary variable of interest, rather than the exact ordered synteny itself.

In order to infer long range information, we need to use more complicated, but nevertheless still polynomial time algorithms on strings [18, 188]. Such algorithms vary between applications, and can also take into account the copy numbers of the markers on the extant species. This is followed by parsimonious inference of the binary syntenic characters [43].

The inference of ancestral syntenies is illustrated in Figure 1.5, in which the inferred syntenies at an ancestral species are marked in the tree. It is also possible to use probabilistic techniques to infer syntenies [42].

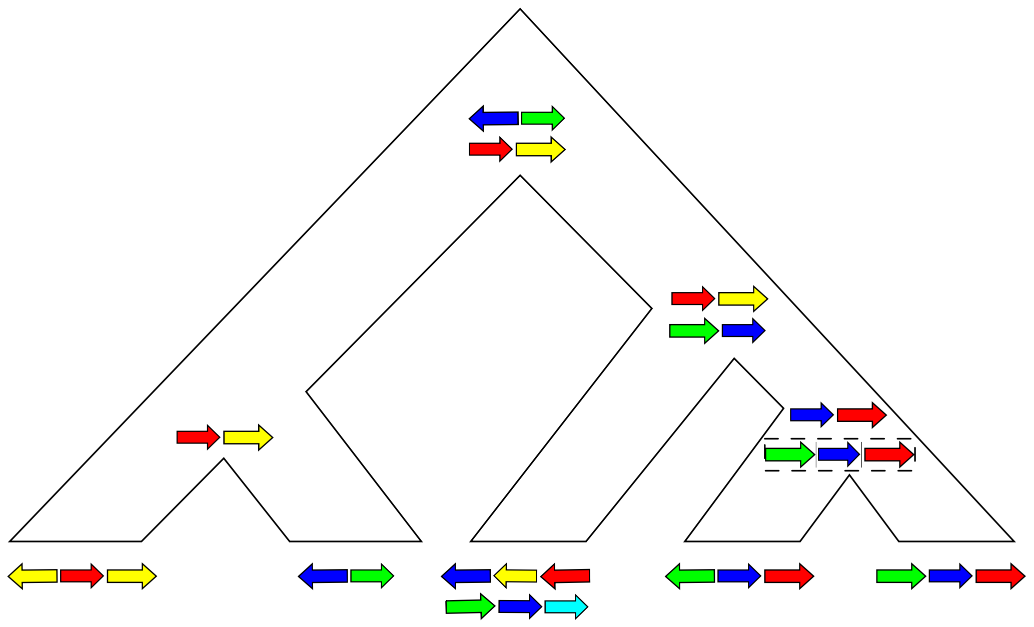


Figure 1.5: Inferring ancestral syntenies using Dollo parsimony. The coloured arrows are markers, with the direction of the arrow representing the orientation of the markers in the genomes. The immediate neighbourhood of the markers on the extant genomes is shown at the leaves. A synteny is inferred to be present at the ancestor if there are 2 extant species which contain the corresponding extant synteny such that the evolutionary path between them contains the ancestor. Sometimes, the synteny is preserved up to a change of order of the markers within it. Such syntenies are marked by dotted boxes, indicating that the order of the markers is not known.

### 1.4.3 The mapping procedure

Mapping ancestral genomes corresponds to the problem of finding a set of one or more sequences of the ancestral markers, with each marker appearing at least once and at most as many times as its copy number. These sequences should have the following properties.

1. The sequences should agree with the predicted genome structure. This means that if we expect the ancestral genome to be circular, as in the case of bacterial genomes, we wish to obtain a single circular sequence. On the other hand, in the case of linear genomes, such as mammalian genomes, we would expect the ancestor to be composed of linear chromosomes, and the same property is desired in the reconstructed genome map.

2. The genome map should be consistent with the syntenic information between markers. This means, in an ideal scenario, we should be able to find all ancestral syntenies that we predicted on the map we have reconstructed.

This problem is similar to reconstructing physical maps [2,48], and so there is a large corpus of work in computational biology to address such problems.

At a high level, the problem may be attacked using a combinatorial concept called the *consecutive ones property.* This concept was first introduced for other problems in biology [87], but has since been used for various problems in both physical mapping [133] and ancestral genome reconstruction [17, 38, 43, 170]. The concept is a mathematical encoding of syntenies between markers, and lends itself to algorithmic techniques that can be used to reconstruct *contiguous ancestral regions* (CARs), large sequences of markers that are consistent with the predicted ancestral syntenies.

One of the major drawbacks of using this approach is that the original concept did not have a mechanism to encode or handle repeated markers. Therefore, it was common to work with markers that are inferred to occur exactly once in the ancestral genome [38, 43, 170]. Later in the dissertation, we will discuss how such data can also be encoded, and the computational difficulties that arise on doing so.

## 1.5 Contributions in this dissertation

The theoretical contributions in this dissertation cover two main topics, both of which lie within the same mathematical framework of data representation. The first topic involves finding chromosomal organization in the presence of repeated genomic segments. We cover the difficulty of deciding the *realizability* of a given input as a valid genome map, which conforms to the constraints imposed within the input. We introduce the concept of *repeat spanning intervals* to aid with reconstructing the chromosomal organization, and prove tractability and intractability results concerning optimizing over sets of repeat spanning intervals in order to achieve realizability.

The second topic concerns the extension of *vertex ordering* problems in graphs to hypergraphs, and using them as natural optimization criteria to approximate chromosomal organization in the absence of repeated segments on the genome. We treat this as a theoretical problem and prove approximation bounds for one such extension. We also briefly discuss polynomial time solvability of these problems, and state a conjecture on the structure of the input in order to admit a polynomial time algorithm for optimal ordering.

We also discuss the application of some of the developed algorithms to real data from the field of ancestral genome reconstruction. We show the results of a scaffolding software for ancestral genomes on the genome of the ancestral Black Death agent. We also include the results of an ancestral genome map reconstruction software on ancestral *Anopheles* mosquito genomes.

## 1.6 A note to the readers

The results in this document are worded into formal mathematical statements. However, considering the community this manuscript is aimed at, it is essential that a person not acquainted with mathematics or theoretical computer science should be able to understand, or at least grasp the gist of the statements. Keeping this in mind, we present here a roadmap for reading the dissertation.

Chapter 2 presents the mathematical model that we will be using in this dissertation. It also serves to establish the connection between the mathematical objects we are dealing with, and the biological questions that are being addressed. Most importantly, this chapter presents an informal description of the mathematical problems we will address in this dissertation. Where suitable, we provide illustrations to elucidate our point.

The second part of the document is dedicated to the problem of dealing with repeats. The introductory chapter in the section lays out the problems that computational biologists face when having to deal with repeated genomic segments in genome mapping, and the algorithmic solutions, if any, for the same. These chapters combine a set of theoretical results, and explain the biological intuition behind each of them.

The third part of the document is a set of theoretical results, which, though interesting in their own right, do not yet have well defined biological applications. The reader is invited to go through them, though those uninterested in results of a more theoretical nature may skip them in a cursory reading.

The final part of the document is the most interesting from a biological standpoint. This compiles details about software developed with collaborators for ancestral genome mapping and scaffolding, and includes results on real data. We refer back to the theoretical results in the manuscript that we used to design the algorithms implemented in the software. So, in case the reader decided to skip the theoretically heavy sections, they may always refer to the relevant results on a need-to-know basis.

The curious reader may always turn to the appendices if they want a quick primer on some mathematical concepts that are used in the manuscript, or to get the pseudocode for some of the algorithms introduced in the text.

# Chapter 2

# Background and preliminaries

The aim of this dissertation is to introduce results concerning the reconstruction of genome maps. In order to do that, we first need to understand exactly what a genome map is within the formal constructs that we will be using. Let us recollect the types of data we are given.

1. We are given genomic *markers*.
2. We may have an upper bound on the number of times a specific marker appears on the genome of interest.
3. We have *synteny information*, represented by sets of markers that are conjectured to occur consecutively on the genome we wish to reconstruct.
4. These sets of markers may be accompanied by some confidence measure, a *weight* or probability that a particular set is truly present in the genome of interest.

There may be more intricate details we can use, but the current information is enough to formally specify a mathematical model. This is the objective of the chapter. We start with the mathematical model of a genome map, and move on to how the data available to us is encoded in a combinatorial construct, and how it relates to this mathematical model.

## 2.1 Representing genome maps

We can start with defining what a genome is. We define a genome map as follows.

**Definition 2.1.** Let $\Sigma$ be the alphabet of genomic markers. A *genome map* $M \subseteq \Sigma^*$ is a set of linear and/or circular sequences on the alphabet $\Sigma$.

To capture the notions of linear and circular chromosomes, as well as plasmids, we can define a genome model.

**Definition 2.2.** Given a genome map $M$ on an alphabet $\Sigma$ of markers, we say that

1. $M$ belongs to the *linear genome model* if there are no circular sequences on $\Sigma$ in $M$,

2. $M$ belongs to the *unichromosomal circular genome model* if there is exactly 1 circular sequence on $\Sigma$ in $M$, and there are no other sequences (linear or circular) in it,

3. $M$ belongs to the *multichromosomal circular genome model* if there are 1 or more circular sequences on $\Sigma$ in $M$,

4. $M$ belongs to the *mixed genome model* if there are 1 or more linear or circular sequences on $\Sigma$ in $M$.

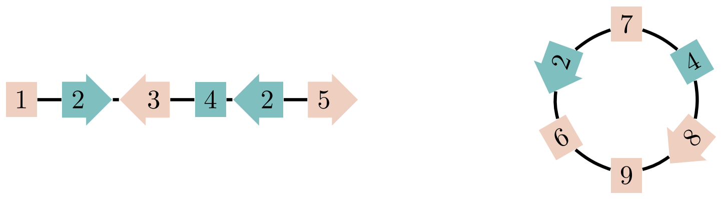
These two definitions capture the essence of a genome: it is a sequence, or a set of sequences (chromosomes and/or plasmids) of markers (genes). The structure of the sequences is specified by the genome model considered. This structure tells us whether the genome under consideration is linear or circular. The modifiers *unichromosomal* and *multichromosomal* specify the number of sequences, i.e. the number of chromosomes. It is important to note that the linear genome model is a special case of the mixed genome model. Furthermore, the unichromosomal circular genome model is a special case of the multichromosomal circular genome model, which itself is a special case of the mixed genome model. Formally, we have the following inclusions.

$$\text{Linear} \subset \text{Mixed},$$
$$\text{Unichromosomal circular} \subset \text{Multichromosomal circular} \subset \text{Mixed}.$$

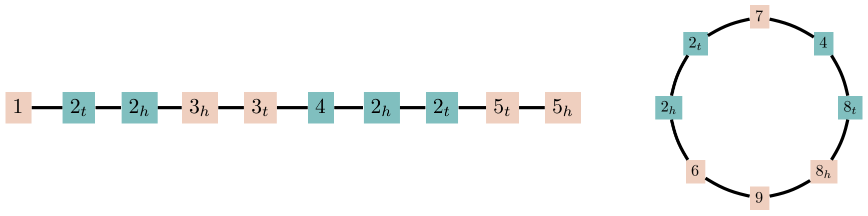
As a result, where multiple circular sequences are concerned, we will ignore multichromosomal circular models and only consider the mixed genome model. The linear and unichromosomal circular model will still prove to be useful, as very often we desire not only a certain structure on the genome to be reconstructed, but also a restriction on the number of sequences that the genome is composed of.

When two markers are found adjacent to each other in a genome map, they are said to form an *adjacency*. A set of three or more consecutive markers on a genome map is called an *interval*. Note that at the moment, we are not considering the exact order of the markers on the genome map; an interval is defined as a set of markers rather than a sequence. Later, we will discuss how to associate a sequential structure to intervals.

At this point, note that the only piece of information we are using is the set of markers. As a rule, we will treat this set as gospel for the algorithmic problems we discuss in this document. While there may be other data that contain errors, we will only use markers that we are confident to find in the genome being reconstructed.



(a) Genome map with undoubled markers.



(b) Genome map with doubled markers.

Figure 2.1: A genome map in the mixed genome model. Figure 2.1a shows a genome map with all markers undoubled, the direction of the oriented markers shown by arrows. Of the two sequences in this map, one is a linear sequence, while the other is a circular sequence. Markers that appear more than once in the map are depicted in green. Figure 2.1b shows the doubling of the same map. Every oriented marker has been separated into a head and tail component, indicated by the subscripts $h$ and $t$ respectively.

### 2.1.1 Repeats.

The definition of a genome map is very general; since we are simply asking for a sequence of markers, the only restriction placed on the map is the structure. In this context, we have the following definitions.

**Definition 2.3.** A *repeat* in a genome map $M$ is a marker $v \in \Sigma$ which appears at least twice over all sequences in $M$.

In other words, a repeat is simply a repeated genomic marker. As we shall see in the next chapter, the presence of repeated markers leads to theoretical obstacles while trying to reconstruct a genome map.

### 2.1.2 The orientation of markers

Markers in a genome are not point objects- they are sequences of nucleotides, which makes them linear objects. Marker orientations become important when we take into account the fact that DNA molecules are double stranded, and in biological processes, the direction in which a DNA strand is read is important. So, saying that one marker is adjacent to another does not present a complete picture, as this statement excludes information about which end of one marker is adjacent to which end of the other. This leads to the concept of the *orientation* of a marker occurrence in a map. This is usually indicated by a sign (either $+$ or $-$) associated to the occurrence with respect to some reference that is decided upon whilst defining the markers [82].

It is not necessary that the orientation is known for all markers in a map. The set of markers we consider will be assumed to be partitioned into two sets: the set of oriented markers $\Sigma^o$, and the set of unoriented markers $\Sigma^u$. In general, the algorithmic and complexity results in this document are not affected by the orientation of the markers. In the case that the orientation does play a key role, we explicitly make a note of it.

A common practice employed in order to treat oriented markers as point objects is to treat the start and the end of a marker as separate markers in themselves. Thus, given $v \in \Sigma^o$, we can replace it with a head marker $v_h$, and a tail marker $v_t$. An occurrence of $v$ in a genome map $M$ is interpreted as an occurrence of one of two consecutive substrings, $v_h.v_t$ or $v_t.v_h$, based on the orientation of that occurrence of $v$. The process of replacing all occurrences of a marker in an alphabet, and in a map $M$ defined on the alphabet, with the substrings corresponding to the orientation of that marker is called *doubling*.

The alphabet $\Sigma^o$ can thus be partitioned, and the alphabet $\Sigma$ is denoted as a partition into 3 sets, i.e. $\Sigma = \Sigma^h \cup \Sigma^t \cup \Sigma^u$, into a head marker set $\Sigma^h$, and a tail marker set $\Sigma^t$. There is a bijection between the markers in $\Sigma^h$ and those in $\Sigma^t$, defined as the correspondence between the head and the tail of the marker that has been doubled. We represent this relation as an injective involution $\psi\colon \Sigma \to \Sigma$, which has the following properties.

1. For $v \in \Sigma^h$, $\psi(v) \in \Sigma^t$, and no other $v \in \Sigma^h$ maps to $\psi(v)$.

2. For $v \in \Sigma^u$, $\psi(v) = v$.

3. For all $v \in \Sigma$, $\psi(\psi(v)) = v$.

From now, the set $\Sigma^o$ will be assumed to consist of head and tail marker sets, $\Sigma^h$ and $\Sigma^t$. The markers in these sets will still be referred to as *oriented*, as this implies that any genome map they appear in must have been doubled.

The concept of a genome map with repeats is illustrated in Figure 2.1. Figure 2.1a depicts a map in the mixed genome model with oriented and non-oriented markers, while Figure 2.1b shows the doubling of the same map.

## 2.2 Hypergraphs and binary matrices

We now have a mathematical model for a genome map . However, the genome itself is unavailable to us. We only have access to certain facets of the genome: the set of markers, the colocalization information, the copy numbers etc. How do we represent this data so that we can try to obtain a genome from it?

The model we use to do this is by representing the data as a binary matrix, or a hypergraph. A *hypergraph* is a 2-tuple $H = (V, E)$, where $V$ is a set of *vertices*, and $E \subseteq 2^V$ is a set of *hyperedges*, each hyperedge being a subset of $V$.

The *incidence matrix* of a hypergraph $H = (V, E)$, $|V| = n$ and $|E| = m$ is the binary $m \times n$ matrix $M$ where the columns are index by the vertices in $V$ and the rows are indexed by hyperedges in $E$, and $m_{ij} = 1$ if and only if the vertex associated with the column $j$ belongs to the hyperedge associated with the row $i$.

The basic idea of the model is this: every marker is represented by a vertex of the hypergraph. Since the colocalization information tells us which markers occur consecutively on the genome of interest, every piece of such information is encoded as a hyperedge. Then we look for a genome map representing the genome of interest, such that this encoding is in some sense consistent with it. Formally, we define an instance to the problems we deal with as the following object.

**Definition 2.4.** Let $\Sigma = \Sigma^h \cup \Sigma^t \cup \Sigma^u$ be a set of markers. An *instance* on $\Sigma$ is a triple $(H, \mu, w)$, with the following definitions.

1. $H = (V, E)$ is a hypergraph on the set of vertices $V$, which is partitioned into sets $V^h, V^t$ and $V^u$.

2. There is a bijective map $\chi\colon \Sigma \to V$, which can be partitioned into 3 different bijections, $\chi_h\colon \Sigma^h \to V^h$, $\chi_t\colon \Sigma^t \to V^t$ and $\chi_u\colon \Sigma^u \to V^u$.

3. $\mu\colon V \to \mathbb{N}$ is called a *multiplicity* function. For every marker $v \in \Sigma$, $\mu(\chi(v))$ is the upper bound of the copy number of $v$, and $\mu(\chi(v)) = \mu(\chi(\psi(v)))$.

4. $w\colon E \to \mathbb{R}^+$ is a non-negative weight function on the set of hyperedges.

The model we have presented seeks to represent markers by vertices, which are point objects. This is where the notion of doubling markers comes into play. In order to capture the orientation of markers into the model, we associate 2 vertices to the doubling of the marker instead: a *head* vertex, which is the image of its doubled head marker in $\chi_h$, and a *tail* vertex, the image of its doubled tail marker in $\chi_t$. The vertices associated to oriented markers are also said to be *oriented*, and conversely, vertices associated to unoriented markers are termed as *unoriented* vertices.

The involution $\psi$ defined over the doubled alphabet $\Sigma$ carries over to the set of vertices $V$. Given a vertex $v \in V$ associated to a marker $u \in \Sigma$, its *mate*, denoted by $\overline{v}$, is the vertex $\chi(\psi(u))$. Unoriented vertices are their own mate, and head oriented vertices are mated to their corresponding tail vertices and vice-versa. We also include edges $\{v, \overline{v}\}$ in the hyperedge set $E$, indicating that the two associated oriented markers should always occur together in a genome map. Since markers in $\Sigma$ are mapped bijectively with the set of vertices $V$ in an instance, a genome map may be interpreted as a set of sequences on the alphabet $V$. This is the convention we will adopt in the rest of the document.

The function $\mu$ specifies a natural partition of the vertex set of $H = (V, E)$. Formally, we define the following sets.

**Definition 2.5.** Given an instance $(H, \mu, w)$, $H = (V, E)$, the set of *repeats* is the set $V_R = \{v \in V : \mu(v) > 1\}$. The set of *non-repeats* is the set $V \setminus V_R$.

The set $V_R$ denotes the set of markers that we have inferred to possibly appear more than once in the genome of interest.

Often, we will assume that all the hyperedges in the set $E$ of an instance have equal weight. Alternately, it may be the case that the weights of the hyperedges do not play a role in designing a solution to the mathematical problem under consideration. In these cases, we specify an instance as the 2-tuple $(H, \mu)$, in order to make the notation less cumbersome. Sometimes, we will also deal with cases when the function $\mu$ maps all vertices to 1. In these cases, an instance is defined as the input $H = (V, E)$, and we shall specify the weight function $w$ as a separate input to avoid confusion. Given the incidence matrix of a hypergraph, the notions of the multiplicity function and the weight function can be naturally associated to the columns and the rows of the matrix respectively.

Since the vertices of the hypergraph are being used to represent markers, and the hyperedges represent sets of markers that are inferred to be colocalized, when a hyperedge is of size 2, it simply means that the two markers associated to the vertices in the hyperedge are inferred to occur together on the genome map. As such, they represent *adjacencies* in the genome. Similarly, when a hyperedge has more than 1 vertex, the markers corresponding to the vertices in the hyperedge are inferred to occur in a continuous *interval* on the genome map. In the rest of the document, the words *adjacency* and *interval* must be interpreted as explained above. The subset of all adjacencies in a hyperedge set $E$ is denoted by $E_A$. The subset of all intervals is denoted by $E_I$. The weight functions restricted to the sets $E_A$ and $E_I$ are denoted by $w_A$ and $w_I$ respectively. The instance $(H_A, \mu, w_A)$, where $H_A = (V, E_A)$, is called the *adjacency instance* underlying $(H, \mu, w)$.

Sometimes, we may have more information about intervals than can be captured by the current model. We may know, for example, the exact order in which the markers associated to the vertices in an interval appear on the genome. Based on this, we can partition the set of intervals as follows.

**Definition 2.6.** Given an instance $(H, \mu, w)$, where $H = (V, E)$, an interval $e \in E_I$ can be classified into one of the following classes.

1. $e \in E$ is an *unordered interval* over $k$ vertices if $e = \{v_0, \ldots, v_{k-1}\}$, where each $v_i \in V$ for all $i \in [k]$, $k > 2$.

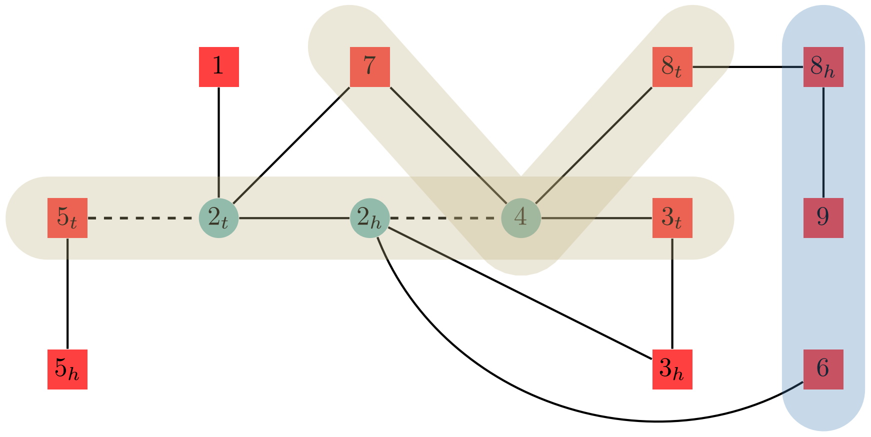


Figure 2.2: An example of an instance. The figure depicts a hypergraph, with edges representing adjacencies, and overlays representing intervals. In the instance, the circular, green coloured vertices are repeats. Of these, the vertex with label 4 has multiplicity 2, and the vertices with labels $2_h$ and $2_t$ have multiplicity 3. The square, red coloured vertices are non-repeats. Unordered intervals are shaded blue, and ordered intervals are shaded brown. The hyperedge $e = \{5_t, 2_h, 2_t, 4, 3_t\}$ is associated with the order $o(e) = 3_t.\ 4.\ 2_h.\ 2_t.\ 5_t$, and the hyperedge $e' = \{7, 4, 8_t\}$ is associated with the order $o(e') = 7.\ 4.\ 8_t$. Dotted lines are edges that are not present, but are implied by the order on the intervals.

2. $e$ is an *ordered interval* over $k$ vertices if $e = \{v_0, \ldots, v_{k-1}\}$ is associated to a sequence $o(e)$, in which each vertex $v \in e$ occurs at least once, and no other vertex in $V$ occurs in $o(e)$. An ordered interval $e \in E$ is said to have *content* $\left\{v_0^{i_0}, \ldots, v_{k-1}^{i_{k-1}}\right\}$ if each vertex $v_j$ occurs exactly $i_j$ times in $o(e)$, for $i_j \in \mathbb{N}$ for all $j \in [k]$. The sequence $o(e)$ is denoted by $x_0.x_1 \ldots x_{l-1}$, where $x_i \in e$ for all $i \in [l]$, and $l = \left(\sum_{j=0}^{k-1} i_j\right)$. The *reversal* of $o(e)$, denoted by $\overline{o}(e)$, is the string $x_{l-1}. \ldots .x_0$.

Given an ordered interval $e \in E_I$, an adjacency $\{u, v\}$ is said to be *compatible* with $e$ if either $u.v$ or $v.u$ occurs as a consecutive substring in $o(e)$.

Figure 2.2 shows an example of an instance. This example includes the concepts of adjacencies, intervals and multiplicities.

As in graphs, we can define a notion of a connected hypergraph.

**Definition 2.7.** Let $H = (V, E)$ be a hypergraph. $H$ is said to be *connected* if, for any two vertices $u, v \in V$, there is a sequence of distinct vertices $(v_0, \ldots, v_{k-1})$, where $v_i \in V \setminus \{u, v\}$, such that each of the pairs $\{u, v_0\}$, $\{v_{k-1}, v\}$, and each $\{v_i, v_{i+1}\}$ occur in a common hyperedge in $E$, for $i \in [k-1]$, for some $k \in \mathbb{Z}_{\geq 0}$.

We can also define an induced subgraph of a hypergraph as follows.

**Definition 2.8.** Let $H = (V, E)$ be a hypergraph, and let $U \subseteq V$ be an arbitrary subset of the vertices. The subhypergraph *induced* by the set $U$, denoted by $H[U]$, is the hypergraph with vertex set $U$ and hyperedge set $E_U = \{e \cap U : e \in E\}$.

It may be easier to visualize an induced subhypergraph in terms of the incidence matrix of the original hypergraph. Let $M$ be the incidence matrix of $H = (V, E)$. Given a subset $U$ of the vertices of $H$, $H[U]$ is the hypergraph whose incidence matrix is the submatrix of $M$ obtained by retaining columns associated to the vertices in $U$. Rows with no 1's or with only one 1 can be discarded as trivial, as they correspond to empty hyperedges (which contain no vertices in $U$) and to single element hyperedges (which are the same as vertices) respectively.

Using Definitions 2.7 and 2.8, it is possible to define a *connected component* of a hypergraph $H = (V, E)$. This is an induced subhypergraph on $U \subseteq V$ such that $H[U]$ is connected, but $\forall U'$, $U \subset U'$, $H[U']$ is not connected.

Given a set of vertices $U \subseteq V$ from an instance $(H, \mu, w)$, where $H = (V, E)$, the instance *induced* on $U$ specifies the instance $(H[U], \mu_U, w_U)$, where $\mu_U$ and $w_U$ are the restrictions of $\mu$ and $w$ to the set $U$ and $E_U$ respectively. In the case of $w_U$, all the hyperedges in $H[U]$ which were not present in $E$ inherit their weight from the hyperedges in $E$ that were truncated to obtain them.

## 2.3 Properties on binary matrices and hypergraphs

Given an instance $(H, \mu, w)$, we now want to see if the data encoded in this instance satisfies the constraints of being extracted from a genome map. We do this by defining the following terms.

**Definition 2.9.** Let $(H, \mu, w)$ be an instance, $H = (V, E)$, and let $M \in \mathcal{G}$ be a genome map in the genome model $\mathcal{G}$. We say that the map $M$ is *consistent* with the instance if, for all vertices $v \in V$, the number of occurrences of $v$ in the sequences specified by $M$ is at least 1 and at most $\mu(v)$.

A consistent genome map preserves the upper bound on the copy numbers of the markers represented by the instance. At the same time, in order to see if the colocalization information encoded by the instance is also correct, we will need the following concept.

**Definition 2.10.** Let $(H, \mu, w)$ be an instance, $H = (V, E)$, and let $M \in \mathcal{G}$ be a genome map in the genome model $\mathcal{G}$. A hyperedge $e \in E$ in the instance is said to be *compatible* with $M$ if

1. $e$ is an adjacency $\{v_0, v_1\}$, and there is a consecutive substring in $M$ composed of exactly the vertices $v_0, v_1$,

2. $e$ is an unordered interval $\{v_0, \dots, v_{k-1}\}$, and there is consecutive substring in $M$ in which each $v_i \in e$, $i \in [k]$, occurs at least once, and no other vertex in $V$ occurs in this substring, or

3. $e$ is an ordered interval, with associated order $o(e)$, and there is a consecutive substring in $M$ which is equal to $o(e)$, or the reversal $\overline{o}(e)$.

Finally, we can put these concepts together, to give a definition of what it means for a given instance to be 'good'.

**Definition 2.11.** Let $(H, \mu, w)$, where $H = (V, E)$, be an instance, and let $\mathcal{G}$ be a given genome model. The instance is said to be *realizable* in the genome model $\mathcal{G}$ if there exists a genome map $M \in \mathcal{G}$ such that

1. $M$ is consistent with $(V, \mu)$, and

2. every hyperedge $e \in E$ is compatible with $M$.

Let us take a demonstrable example here. The instance depicted in Figure 2.2 is realizable in the mixed genome model. In fact, Figure 2.1 is a possible realization for this instance. Note that no vertex appears in the genome map more often than its multiplicity, and that the adjacencies and intervals do indeed form consecutive substrings in the genome map. The reader will find that every adjacency or interval in the model is compatible with the map in Figure 2.1, and the vertices appear exactly as many times as their multiplicity.

The realizability of an instance $(H, \mu, w)$ may also be understood as the existence of a special set of *walks* on the vertices of the hypergraph. This set of walks must encounter each vertex $v \in V$ at least once and at most $\mu(v)$ times, and each hyperedge must be compatible with at least one of the sequences of vertices inscribed by these walks.

It is interesting to note here that an instance which is realizable in the linear genome model is also realizable in the unichromosomal circular, the multichromosomal circular and the mixed genome models. For example, given a realization in the linear genome model, if we concatenate the various sequences in the realization, and then add the adjacency between the first and the last vertex in the sequence, this will be a realization in the unichromosomal circular model. Similarly, an instance realizable in the unichromosomal circular model is realizable in the multichromosomal circular and the mixed genome models. This is because realizability does not specify that the only adjacencies and intervals which are compatible with the realization are the ones present in the instance. While constructing a realization of a given instance, we can add adjacencies and intervals that are not present in the input instance, subject to the consistency of the map, and not affect realizability.

Note that the multichromosomal circular genome model constraints the eventual resulting map to be a set of circular sequences. However, the instance given may not naturally admit a set of circular sequences, i.e. we have to add extraneous adjacencies between certain markers in order to get circular sequences. This can be avoided by using the mixed genome model. Since we are allowed to have linear sequences, we can bypass the need to add extraneous adjacencies, and restrict the constructed genome map to adjacencies and intervals that were already present in the instance. Thus, at this point we will completely ignore the multichromosomal circular model, and the term *circular genome model* should be interpreted to mean the unichromosomal circular model.

While these definitions may seem a bit exotic, this is merely a generalization of a long-known property of certain binary matrices.

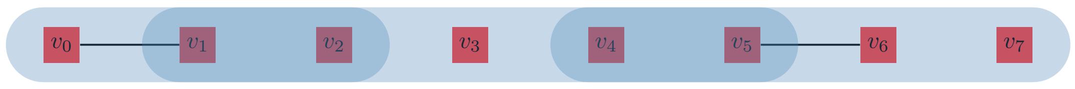
**Definition 2.12.** A binary matrix $M$ is said to have the *consecutive ones property* (C1P) if there exists a permutation of the columns of $M$ such that the 1's in every row of $M$ occur in a single consecutive block in the row. A permutation that imparts this property to a matrix $M$ is called a *consecutive ones ordering* of $M$ (or a C1P ordering of $M$).

A hypergraph $H = (V, E)$ has the *consecutive ones property* (C1P) if its associated incidence matrix has the consecutive ones property.

The consecutive ones property was first introduced by Benzer [15] for studying the fine structure of genes. Before the advent of modern sequencing techniques, it was used extensively for building physical maps [2, 97, 133]. It is closely related to various mathematical objects, such as interval graphs and clique decompositions. For us, it is the simplest case of more general definitions of consecutivity. Assume that the function $\mu$ in a given instance $(H, \mu, w)$ maps all vertices to 1. Then, this instance is realizable in the linear genome model if and only if the hypergraph $H$ has the C1P. The connection is almost obvious: given a C1P ordering for the incidence matrix $M$, we can lay out the vertices of $H$ in the specified order. Then, since every row is a hyperedge, and is stated to have a single consecutive block of 1's, the hyperedges in $E$ must occur as consecutive substrings in this order, which means they are all compatible with the order. Since every vertex appears exactly once in the order, and every row is compatible, the instance must indeed be realizable in the linear genome model.

As an example, take the instance shown in Figure 2.3a. All vertices are unoriented and have multiplicity 1, and intervals are depicted in the figure as overlays. Clearly, there is a natural map in the linear genome model for which this instance is realizable. The incidence matrix of the hypergraph we associate with the data is shown in Figure 2.3b (ignoring the weights on the hyperedges). The matrix associated with this data does not specify the order of the columns (i.e. markers) or of the rows (i.e. colocalization data).

The same matrix, with columns ordered as per the order of the vertices in a realization of the instance, is shown in Figure 2.3c. One can see that every row has a single consecutive block of 1's. Indeed, this is what we would expect, since the data tells us that every row is a synteny between markers that occur together in the genome, and no other marker occurs



(a) A hypergraph with only non-repeats.

| $v_0$ | $v_4$ | $v_3$ | $v_5$ | $v_6$ | $v_7$ | $v_1$ | $v_2$ |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

(b) The incidence matrix of the hypergraph.

| $v_0$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ | $v_7$ |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

(c) Reordered matrix.

Figure 2.3: An example of an instance which has the consecutive ones property.

between them.

A closely related property of binary matrices is the *circular ones property* [201].

**Definition 2.13.** A binary matrix $M$ is said to have the *circular ones property* (Ci1P) if there exists a permutation of the columns of $M$ such that, for every row, either the 1's in the row occur in a single consecutive block, or the 0's occur in a single consecutive block. A permutation that imparts this property to a matrix $M$ is called a *circular ones ordering* of $M$ (or a Ci1P ordering of $M$).

A hypergraph $H = (V, E)$ has the *circular ones property* (Ci1P) if its associated incidence matrix has the circular ones property.

This is equivalent to the realizability of an instance with all vertices having multiplicity 1 in the circular genome model.

A more general definition is required to include the concept of repeats, which is where the idea of realizability is useful. In literature, realizability in a given genome model are defined as generalizations of the C1P [208].

1. An instance which is realizable in the linear genome model is said to have the *consecutive ones property with multiplicity* (mC1P).

2. An instance which is realizable in the circular genome model is said to have the *circular ones property with multiplicity* (mCi1P).

3. An instance which is realizable in the mixed genome model is said to have the *component-wise circular ones property with multiplicity* (cmCi1P).

We will mostly use the notion of realizability in order to be able to specify the genome model of interest in different problems. If the problem being discussed is specific to the C1P, however, we will use the classical definition of the C1P.

## 2.4 Defining the problems

The ambit of the problems discussed in the document fall into two main categories. The first problem asks if, given data, we can determine if this data does indeed come from a genome of interest, or if there are errors in the data. In other words, given a set of syntenies, along with multiplicities of markers in these syntenies, can we reconstruct a genome map which contains all syntenies and at most as many copies of a marker as its multiplicity?

**Question 2.1.** Given an instance $(H, \mu)$, can we determine if it is realizable in a given genome model, i.e. does it have the mC1P, mCi1P or cmCi1P?

The second problem is somewhat more general. It asks how close the given data is to data that would have no errors. So, given the set of syntenies and markers with multiplicities, and knowing that this data has errors, is there any way we can quantify these errors?

**Question 2.2.** Given an instance $(H, \mu, w)$ which is known to not have a realization in a given genome model, can we quantify how 'far' it is from having a realization?

It is very possible that the answer to either of these problems depends on the structure of the instance, as well as the genome model we will be working with. Furthermore, there are various ways to quantify how 'close' an instance is to something ideal. The theoretical content of the document is intentionally split into near-independent parts, the first of which deals with cases with possible repeats, and the second generalizes well-known graph theoretic measures of linearity to instances when the multiplicity function maps all vertices to 1.

It is also necessary to note that even if an instance is realizable in a given genome model, there is usually no guarantee that the genome map associated to it is unique. Very often, there may be multiple maps that are consistent, and the hyperedges in the given instance may be compatible with all these maps. In this case, the onus is on us to choose the right map, the one that is closest to the real genome map. We shall see that this problem is a common by-product of having repeats in the instance.

The final section is devoted to applications of the methods developed before to problems in ancestral genome map reconstruction.

# Part II

# The Consecutive Ones Property with Multiplicity

# Chapter 3

# Genome maps with repeats

In the second chapter, we chalked out a broad idea of the type of problems that we deal with, and the mathematical model we use. We also mentioned, quite insistently, that repeats in the model give rise to computational problems. Here, we explain why repeats are such an obstacle to efficient computation of a genome map. We will formally define the problems we shall encounter over the next four chapters, and state the known results for the tractability of these problems.

## 3.1 The problem with repeats

Repeats are a fairly common occurrence in genomes, and one can immediately surmise that accounting for repeated markers in genome maps would certainly be a welcome addition to current mapping and assembly models [200]. But quite often, the results presented exclude repeated markers, and work solely on unique ones [30]. This is not an arbitrary decision- computing maps and assemblies with repeats is a major, and well studied, problem in bioinformatics. Algorithms usually either fail or become computationally costly when faced with repeated markers. This adds to the already weighty complexity issues that these methods have to work around.

Significantly, the presence of repeats introduces *ambiguity* in the construction of a genome map. Because a repeat is present in different parts of a genome, colocalization information taken from the neighbourhood of the repeat occurrences may differ wildly. At this stage, if the colocalization information is not long enough, it is hard to reconcile information from one neighbourhood with another, and it is hard to tell them apart. This means that we are faced with a choice to make as to which occurrence of the repeat is each neighbourhood associated with.

Furthermore, when many repeats are inferred to be colocalized, it is hard to make out the *internal organization* of the repeats at any location. This becomes a major problem when all colocalization information we have is unordered. For example, assume there are

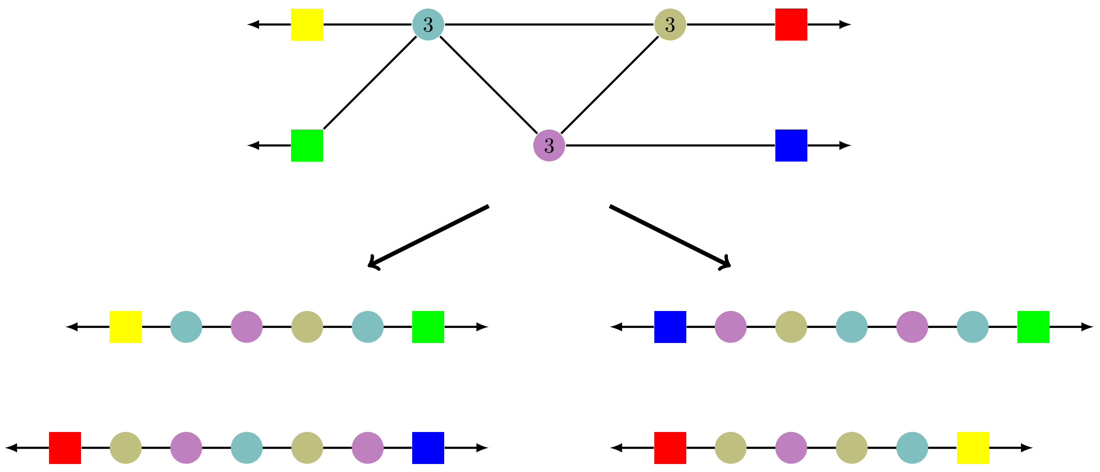


Figure 3.1: Ambiguity in the genome map and the internal repeat organization due to repeats. The square vertices are non-repeats, and the rest are repeats with multiplicity 3. We show two possible ways to construct a map which is consistent and compatible with the adjacencies shown (assuming that the rest of the instance is also realizable in the given genome model). In the two cases, the non-repeats and the repeats are organized differently.

two repeated marker $A$ and $B$, and the subsequence corresponding to an occurrence of $B$ on the genome of interest is $A.B.A$. If we only have unordered intervals, then the only information we can infer regarding the colocalization of $A$ and $B$ is that they are adjacent. So, there are multiple possibilities available for organizing $A$ and $B$ on a genome map, such as $A.A.B$, $A.B.B$, $B.A$, $A.B.A.B$ etc. , limited only by the copy numbers of $A$ and $B$.

Each of these situations is an obvious problem when the only data we have is the adjacencies between markers. Then, since the neighbourhood information available is so localized, the location of a repeated marker, which can be adjacent to many other markers, becomes ambiguous. When many repeats occur next to each other, then this ambiguity magnifies to the problem of finding which repeat occurrences appear where on the genome. Figure 3.1 shows a classic example of both problems due to repeats. The instance here has a set of repeats adjacent to each other, and each adjacent to certain non-repeats, but there are no intervals. Assume that the entire instance is realizable in some genome model. However, there are at least 2 possible ways to traverse the set of adjacent repeats from one non-repeat to another. This is a consequence of the various possible adjacencies with the repeats, a problem which is encountered when there is not enough information available in the neighbourhood of the various occurrences of the repeats. At the same time, within the set of repeats itself, it is not possible to find out what the exact order of these repeats are. Thus, the internal organization of the repeats among themselves is also unclear.

However, if we do have long-range information, which encodes more than just the immediate neighbourhood of a repeated marker, it is possible to narrow down the set of realizations. Such sets fix the order of the vertices in a map, which means the internal organization and the ambiguities get resolved. The problem is that in the presence of such sets, the computational problems we deal with become intractable.

## 3.2 Repeat clusters and repeat spanning intervals

Recall that an instance to a problem is a 3-tuple $(H, \mu, w)$, where $H = (V, E)$ is a hypergraph, $\mu$ is the multiplicity of the vertices in the hypergraph, and $w$ is a weight function on the hyperedges. Also recall that $V_R$ is the set of all repeats in $V$. This partition of the set of vertices gives rise to a way to isolate 'problem areas' in the model, by defining connected components in the hypergraph induced by the repeats.

**Definition 3.1.** Given an instance $(H, \mu, w)$, $H = (V, E)$, a *repeat cluster* $R$ of this instance is a maximal connected component in the induced subhypergraph $H[V_R]$.

The *frontier* of a repeat cluster $R$, denoted by $F(R)$, is the set of non-repeats in $V$ which were in the same hyperedge as a vertex in $R$. The *extension* of $R$, denoted by $\overline{R}$ is obtained by augmenting the hyperedge set of $R$ by a set of hyperedges $E'$ defined as follows.

$$E' = \left\{e' : \exists e \in E, e' = e \setminus (V \setminus (R \cup F(R)))\right\}.$$

The notion of a repeat cluster localizes zones in the model which will present ambiguities and computational problems. Given a repeat cluster $R$, we use the notation $v \in R$ to denote that the vertex $v$ is in the vertex set of $R$.

A final object, which shall play a major role later in this manuscript, is the repeat spanning interval.

**Definition 3.2.** Let $(H, \mu, w)$ be a given instance, $H = (V, E)$, and let $R$ be a repeat cluster in the instance. A *repeat spanning interval* over $R$ is a hyperedge $e \in E$ with the following properties.

1. $e$ is an ordered interval with content $\left\{u^1, v^1, v_0^{i_0}, \ldots, v_{k-1}^{i_{k-1}}\right\}$, and associated sequence $o(e)$, where each $v_i$ is a repeat in $R$, for $i \in [k]$, and $u, v$ are non-repeats in $F(R)$.
2. The sequence $o(e)$ is of the form $u.r_0 \ldots r_{\ell-1}.v$, where all $r_i \in e \setminus \{u, v\}$, and $\ell = \left(\sum_{j=0}^{k-1} i_j\right)$.

The two non-repeats at the extremities of a repeat spanning interval are said to *frame* it. A *minimal repeat spanning interval* is a repeat spanning interval which spans over a single unoriented repeat, or two oriented repeats in $R$.

The repeat spanning interval is basically an encoding of long range information that is highly specific. In this respect, it is very similar to information from long reads in genome assembly, which may span many repeated genomic regions. It tells us exactly what the

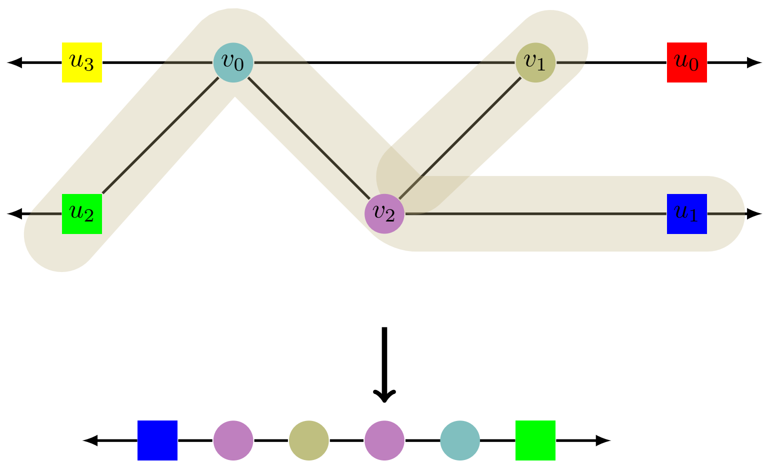


Figure 3.2: An example of how repeat spanning intervals resolve ambiguities. The overlay here is a repeat spanning interval associated with the order $u_2$. $v_0$. $v_2$. $v_1$. $v_2$. $u_1$, spanning over the repeat cluster consisting of the repeats $v_0, v_1, v_2$. In the absence of this information, Figure 3.1 showed an example of the problem encountered in the presence of repeats. However, with this information, a single map is fixed, as depicted, with the internal order within the repeats partially resolved.

order of markers is in a consecutive subsequence of the genome map, when we know that the subsequence starts and ends at unique markers, but spans repeats. For example, take the instance given in Figure 3.1. If we had no intervals, but only adjacencies, there are two possible realizations for this instance in the linear genome model. On the other hand, if we are given repeat spanning intervals, it may be possible to disambiguate the possible genome maps, as well as specify the correct repeat organization within a cluster. This case is illustrated in Figure 3.2.

In the next few chapters, we will see that repeat spanning intervals are much better behaved than general intervals. They will be an important addition to the model.

## 3.3 Deciding the existence of a genome map with repeats

Consider the following problem.

**Problem 3.1.** Let $(,\mu)$ be an instance, $H = (V, E)$ and let $\mathcal{G}$ be a genome model. Does there exist a realization $M$ of the instance such that $M \in \mathcal{G}$?

This is the first and most basic of the problems of interest to us, and the formal statement for Question 2.1. To put it in a more applied framework, it asks if, given the data regarding the copy number and colocalization of markers, there is a genome from which this data could have originated.

Note here that the problem says nothing about getting a unique realization. Indeed, it is very possible that the data provided to us is insufficient to determine a unique realization, as explained in Section 3.1. That being said, the problem is interesting since we are seldom able to obtain data without errors. This means that, before we subject the data to possibly resource hungry techniques of optimization, we might want to first see if the data is already optimal. An algorithm to solve the decision problem can also function as a greedy heuristic, a proxy in the absence of an efficient optimization algorithm.

### 3.3.1 Known results

The best known decision result for Problem 3.1 is when the multiplicity function $\mu$ maps all vertices to 1. Then, assuming that the genome model $\mathcal{G}$ is the linear model, the problem degenerates into the problem of deciding if the hypergraph in the instance has the C1P. Testing if a hypergraph has the C1P, which is the same as testing whether its incidence matrix has the C1P, can be done in polynomial time and space, as shown by Fulkerson and Gross [87]. Since then, many algorithms have been designed to decide if a hypergraph/binary matrix has the C1P in time and space linear in the size of the input [99,142,175], the first of which can be attributed to Booth and Leuker [24].

**Theorem 3.1.** *[24] Given $(H, \mu)$, $H = (V, E)$, $|V| = n$, $|E| = m$, where $\mu$ maps all vertices to 1 and all intervals are unordered, it is possible to decide if the hypergraph $H$ has the C1P in $O\left(n + m + \sum_{e \in E} |e|\right)$ time and space.*

A by-product of this theorem is that, assuming $\mu$ still maps vertices to 1, the decision problem is tractable in linear time and space for all genome models. Realizability in the circular genome model is equivalent to the circular ones property (Ci1P) [201]. This can be tested by complementing those rows in the incidence matrix that have a 1 in their first column in the initial ordering, and then testing for the C1P. Realizability in the mixed genome model is equivalent to the cmCi1P, and since each vertex can only occur once, this is the same as testing if every connected component in the hypergraph has the Ci1P. It is also worth noting that in the case when certain intervals are ordered, it is relatively easy to decide if the resulting instance is realizable. This can be done by constructing a *PQ-tree*, a data structure which represents all possible realizations [24]. The tree consists of 2 types of nodes, $P$-nodes, whose children can be permuted in any manner, and $Q$-nodes, on whose children a linear order is imposed, but the order may be reversed. Once we have the tree for the instance in which all intervals are unordered, we can check if the given set of ordered intervals is compatible with the orders encoded in the tree.

But as stated before, the scenario completely changes when we introduce repeats into the model. Deciding if an instance is realizable in the linear genome model is then only possible in a few, restricted cases, a result that was first proved by Batzoglou and Istrail [13], and later tightened by Wittler et al. [208].

**Theorem 3.2.** *[208] Let $(H, \mu)$ be an instance, $H = (V, E)$, and let $\mathcal{G}$ be a given genome model.*

1. *It is possible to decide the realizability of the instance in $\mathcal{G}$ in polynomial time if every hyperedge in $E$ is an adjacency.*

2. *If $\max_{v \in V} \mu(v) \geq 2$ and $\max_{e \in E} |e| \geq 3$, then deciding the realizability of the instance in the model $\mathcal{G}$ is generally NP-complete.*

The proof idea for the tractability result is simple. The hypergraph $H$ in this case is a graph. Add a new vertex $v_0$ to the instance, having unbounded multiplicity. From every vertex $v \in V$

1. if $v$ is an unoriented vertex with degree $deg(v) \leq 2\mu(v)$, add $2\mu(v) - deg(v)$ edges from $v$ to $v_0$, or

2. if $v$ is an oriented marker with degree $deg(v) \leq \mu(v) + 1$, add $\mu(v) - deg(v)$ edges from $v$ to $v_0$. Add multiedges between $v$ and $\overline{v}$ after performing this operation for all oriented vertices, so that their degree is $2\mu(v)$.

An Eulerian tour in the resulting graph will give a walk on the vertices in which every edge is traversed exactly once. This specifies a genome map. In order to restrict the resulting genome map to a specific genome model, extra constraints need to be placed on the total degree of the connected components in the graph, but this can also be done in polynomial time. On the other hand, if there is an unoriented vertex $v$ whose degree is greater than $2\mu(v)$, or an oriented vertex $v$ whose degree is greater than $\mu(v) + 1$ after collapsing multiedges between mates, or if the extra constraints due to the genome models are violated, the instance cannot admit a valid Eulerian tour.

The hardness result is the tightest possible result for Problem 3.1. As soon as we constraint the multiplicity to be 1, Theorem 3.1 can be applied, and an algorithm for deciding the realizability of the instance instance. On the other hand, the tractability result states that if we are only dealing with adjacencies, the problem is equivalent to checking for Eulerian tours in a graph. But if both constraints are removed, the problem takes a leap in computational complexity.

It is possible to get around the negative result by imposing further constraints on the instance structure. This is the principle behind the following theorem.

**Theorem 3.3.** *[40] Let $(H, \mu)$ be an instance, with hypergraph $H = (V, E)$ having the following properties for every $e \in E$.*

1. *$e$ is an adjacency.*

2. *$e$ is an interval containing exactly $1$ repeat $v \in V$, and $e' = e \setminus \{v\}$ is a hyperedge in $E$.*

*Then, it is possible to decide if $(H, \mu)$ is realizable in the linear genome model in polynomial time and space.*

This result is achieved by using the structure of the intervals to restrict the position of the repeats, and then working around these positions by using Theorem 3.1. The result again relies on the PQ-tree data structure to impose these restrictions, and pares down the possible orders in the tree using the intervals.

### 3.3.2 New results

The concept of repeat spanning intervals opens new possibilities in extending the tractability of Problem 3.1. Each repeat spanning interval restricts the space of possible realizations for a given instance. We will show that, in the case that all intervals containing repeats are repeat spanning, the decision problem is tractable for all genome models.

We will also present a parameterized algorithm for the general decision problem. The exponential complexity of this algorithm will depend on the multiplicity, the number of repeats and the maximum size of a hyperedge, rather than the number of vertices, hyperedges, or the total size of the hyperedges.

## 3.4 Optimizing to get a genome map with repeats

While decision problems are the natural starting point for exploring the possibilities behind the model, they are quite limited. Most data we obtain has at least some degree of errors or inconsistencies, and ideally, we would like to be able to filter these errors before trying to use the model to get a realization. This motivates the slew of optimization problems we have, which can be captured by the following general problem.

**Problem 3.2.** Let $(H, \mu, w)$ be an instance, $H = (V, E)$, and let $\mathcal{G}$ be a genome model. Find a minimum weight set $S \subseteq E$ such that the instance $(H', \mu, w')$, where $H = (V, E \setminus S)'$, is realizable in $\mathcal{G}$, where $w'$ is the restriction of $w$ to the set $E \setminus S$.

There is a stark dichotomy to be seen between the tractability of decision problems, and the natural optimization problems that arise from them. This is exhibited in the results known for Problem 3.2.

### 3.4.1 Known results

Perhaps the best example of how quickly the scenario changes in the optimization world is when Problem 3.2 is restricted to the linear genome model, with the function $\mu$ mapping all vertices to 1. This is the equivalent of finding a minimum weight set of rows in a binary matrix to delete in order to get a submatrix that has the C1P. We refer the reader to Dom [64] for a survey of optimization problems related to the C1P.

**Theorem 3.4.** *Problem 3.2 is NP-hard for the linear and circular genome models, even when the function $\mu$ maps all vertices to* 1*.*

The hardness of these problems means it is usually inefficient to optimize over the colocalization information if we want to find a subset of such information that best fits a linear or circular genome. There are methods that can be used to approximate the problem [65], as well as algorithms that run reasonably fast, assuming that certain parameters in the given instance are small [65, 161]. When the multiplicity function maps all vertices to 1, we have the following result.

**Theorem 3.5.** *[161] Problem 3.2 is solvable in time $O^*\left(10^k\right)$ for unweighted instances in the linear genome model if $\mu(v) = 1$ for all $v \in V$, where $k$ is the number of hyperedges that need to be deleted in order to obtain an instance which is realizable in the linear genome model.*

The notation $O^*(\cdot)$ hides factors that are polynomial in the size of the instance. The algorithm uses the fact that the interval deletion problem, where one needs to delete the smallest number of edges in a graph to get an interval graph, is fixed parameter tractable in the number of edges to delete [141]. The exponent base of 10 comes from the upper bound of at most 10 recursive calls being made in the the branching step of the FPT algorithm. Since we expect the data to be at least somewhat close to an optimal instance, the number of hyperedges needed to be deleted is usually low. However, the constant is still quite large for practical purposes.

Alternately, there are approximation algorithms for this problem for the linear genome model when the multiplicity of all vertices is 1.

**Theorem 3.6.** *[65] There is a polynomial time algorithm for Problem 3.2 for unweighted instances in the linear genome model if $\mu(v) = 1$ for all $v \in V$, which outputs a solution within a factor of at most $\Delta_e + 4$, where $\Delta_e$ is the maximum size of a hyperedge.*

The algorithm for this result is a clever use of a forbidden submatrix characterization of the C1P [202], which is equivalent to the linear genome model in this case. The idea is to find such submatrices, and destroy them by deleting a row.

If we try to solve Problem 3.2 for the mixed genome model, we find the limit of tractability for the problem.

**Theorem 3.7.** *[138] Let $(H, \mu, w)$ be an instance, with hypergraph $H = (V, E)$.*

1. *Problem 3.2 can be solved for the mixed genome model if all hyperedges in $E$ are adjacencies.*

2. *If $\max_{e \in E} |e| \geq 3$, then Problem 3.2 is NP-hard for the mixed genome model, even if $\mu(v) = 1$ for all $v \in V$.*

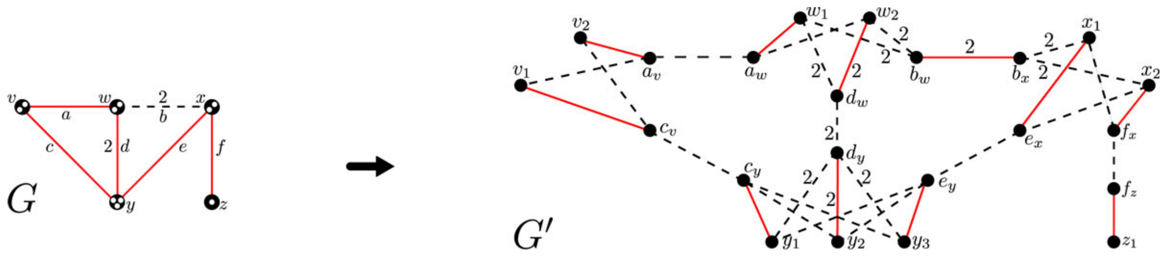


Figure 3.3: An example of the $b$-matching algorithm. The dots within a vertex $v$ is the associated value of $b(v)$, and the labels on the edges are edge weights. This figure was taken from Maňuch et al. [138].

Strikingly, the hardness result does not depend on the multiplicity function at all. The tractability result, however, is interesting, and provides a way to find a maximum weight set of adjacencies that is realizable as a mixed genome. The key to this result is the algorithm for maximum *b-matching*. First published by Dessmark et al. [58], this has become a staple in various methods for ancestral genome reconstruction [185, 198].

**Definition 3.3.** Let $G = (V, E)$ be a graph, and let $b\colon V \to \mathbb{N}$ be a function that maps the vertices to natural numbers. A *b-matching* of $G$ is a set of edges $S \subset E$ such that every vertex $v \in V$ is adjacent to at most $b(v)$ edges in $S$.

This concept can be exploited by using the function $\mu$ to limit the number of edges a vertex can be adjacent to in the mapping. Given an instance $(G, \mu, w)$, $G = (V, E)$, with only adjacencies and all vertices unoriented, the algorithm of Maňuch et al. makes the following construction [138].

1. For every vertex $v$, introduce $2\mu(v)$ vertices $\left\{v_0, \ldots, v_{2\mu(v)-1}\right\}$.
2. For every edge $e = \{u, v\}$, introduce 2 vertices $e_u$ and $e_v$, and the edge $\{e_u, e_v\}$. Give this edge the weight $w(e)$.
3. Add edges from $e_u$ to each $u_i$, $i \in [2\mu(u)]$, and given all these edges weight $w(e)$.

Once this construction is done, as depicted in Figure 3.3, the classical maximum matching algorithm [146] is run on it. If, for an edge $e = \{u, v\}$, $e_u$ is matched to $e_v$, the edge $e$ is discarded. Otherwise, it is included into the optimal set of edges. Since there are $2\mu(v)$ copies of each vertex $v$, each $v$ can be adjacent to at most $2\mu(v)$ edges. Once the algorithm is finished, the resulting graph can be augmented by adding a vertex $t$ with unbounded multiplicity, and edges from each $v$ to $t$ such that $v$ has degree $2\mu(v)$. An Eulerian tour on the resulting graph contains each edge in the chosen set, and each vertex $v$ appears at most $2\mu(v)$ times. By disconnecting the tour at all occurrences of $t$, we obtain a set of linear and circular sequences, proving realizability in the mixed genome model. From the maximum matching algorithm, this is guaranteed to be the maximum weight set of such edges. In order to adapt the algorithm to oriented vertices, we introduce $\mu(v)$ copies for an oriented vertex, with $\mu(v)$ edges between $v$ and $\overline{v}$.

### 3.4.2 New results

Since the inclusion of intervals in an instance makes optimization hard, but optimizing a set of adjacencies is doable in polynomial time (as long as we are working with mixed genomes), the direction which we take is to first obtain a realizable instance consisting only of adjacencies, and then using a set of repeat spanning intervals to disambiguate repeats. The goal, then, is to discard a minimum weight set of repeat spanning intervals such that the remaining instance is still realizable.

This 2-stage approach, which we call *partial optimization*, is formally defined and discussed in Chapters 5 and 6. The idea is that if we have an instance composed only of adjacencies which is realizable, we can assume that some realization of this instance must be the realization we are looking for. Following this, the repeat spanning intervals are introduced in order to filter the list of possible realizations.

We prove that this approach is NP-hard in general. We also give a polynomial time algorithm for the case when the repeat spanning intervals in the instance are all minimal. Also, motivated by the fact that repeat clusters and multiplicities are often small, we give a dynamic programming scheme that exactly computes an optimal solution to the problem.

# Chapter 4

# The existence of genome maps with repeats

In the current chapter, we shall discuss 2 new results regarding Problem 3.1. Both are tractability results, but in different senses. The first is a decision result for Problem 3.1 when the repeat vertices in the instance are only contained in either repeat spanning intervals or adjacencies. The result is in part inspired by Theorem 3.3, in that we use similar techniques to isolate 'paths' of non-repeats, which we then test for realizability in the linear genome model.

The second result is more of theoretical interest, and presents a fixed parameter tractable algorithm for deciding realizability in the linear genome model, parameterized by the number of repeats, the maximum multiplicity, the maximum size of a hyperedge, and the maximum degree of a vertex.

Since decision problems do not optimize over the set of edges, the weight function in an instance $(H, \mu, w)$ is unused. In order to reduce clutter in the notation, we provide the instance as a 2-tuple instead, $(H, \mu)$.

The results presented in this chapter were joint work with Cedric Chauve and Murray Patterson [41].

## 4.1 Including repeat spanning intervals

The result we present here states that if the repeats in an instance are localized to adjacencies or repeat spanning intervals, then the decision problem on this instance can be solved in polynomial time and space.

**Theorem 4.1.** *Problem 3.1 can be solved for all genome models on an instance $(H, \mu)$, $H = (V, E)$, $|V| = n$, $|E| = m$, if every interval is either repeat-free or a repeat-spanning interval in time $O(n + m + \bar{\mu}\rho + \xi)$ and $O(n + m + \bar{\mu}\rho + \xi)$ space, where $\xi = \sum_{e \in E} |e|$.*

The idea behind this result is that every repeat spanning interval can be simulated by a path in an augmented instance, which we will construct. This construction shall be used frequently in the next couple of chapters, so we will formalize the notion and prove a lemma concerning it. This lemma will be a key ingredient in the intractability result in Chapter 6.

**Overview of applications.** Theorem 4.1 is particularly important from the point of view of applicability. It tells us that if repeats are only contained in ordered intervals which are framed by unique markers, it is possible to decide the realizability of the instance. Since the general decision problem is intractable [208], the class of polynomial time decidable instances is not very large, which makes this result helpful in many respects. In the absence of good optimization algorithms, it also provides a simple heuristic for finding a realizable instance. Assume the realizability of the instance can be decided by Theorem 4.1, and it is not realizable. We can discard the minimum weight adjacency or interval and decide realizability for the remaining instance instead. This process can be carried out till a realizable instance is found. If the data encoded in the instance is of reasonably good quality, the resulting realizable instance will be close to the optimum solution.

This is especially useful since, at least in the context of reconstructing ancestral genome maps, it is relatively easy to detect repeat spanning intervals. This can be done via a parsimonious scheme over the phylogeny of the ancestor under consideration, by finding extant genomes in which the markers associated to the vertices in the interval occur in a given order [177]. A repeat spanning interval inferred to have occurred in the ancestor also merits a certain degree of confidence: it means that a relatively long segment of the genome has not only been conserved in content, but also in the order that this content is arranged in.

### 4.1.1 The equivalence of repeat spanning intervals and paths

Consider an instance $(H, \mu)$, $H = (V, E)$ in which the set of repeats contained in repeat spanning intervals are not contained in any other intervals. Let $E_{rsi} \subseteq E$ be the set of repeat spanning intervals specified in the instance. This instance is more general than the hypothesis for Theorem 4.1, which does not have unordered intervals containing repeats. Results on the more general instance, thus, also hold for instances of the form given in Theorem 4.1.

We will construct a new instance $(H', \mu')$ from $(H, \mu)$, where $H' = (V', E')$, and the set $E'$ of hyperedges does not contain any repeat spanning interval.

#### Construction

Initialize $V' = V$, $E' = E \setminus E_{rsi}$ and $\mu' = \mu$. Iterate over every repeat spanning interval $e \in E_{rsi}$, such that $o(e) = u.r_0.r_1.\dots.r_{k-1}.v$, where $u, v$ are non-repeats, and all the $r_i$ are

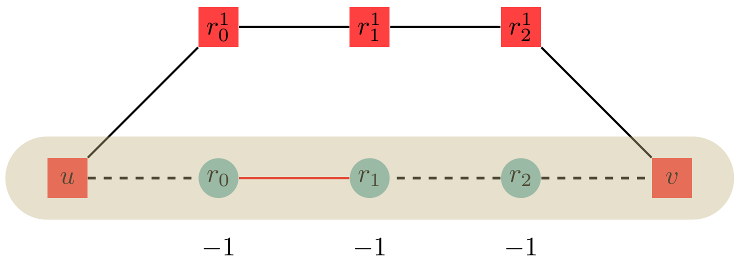


Figure 4.1: Replacing repeat spanning intervals by paths. In this example, the interval $\{u_0, v_0, v_1, v_2, u_2\}$ is a repeat spanning interval associated with the order $u_0$. $v_0$. $v_1$. $v_2$. $u_1$, and can be replaced by a path as shown. The multiplicity of all repeats used in the order is decreased by the number of times they occur in the order. In this case, each repeat has its multiplicity decreased by 1. The red adjacency is deleted once either of $r_0$ or $r_1$ has degree greater than a constant multiple of its multiplicity, depending on whether they are oriented or unoriented.

(not necessarily distinct) repeats, and make the following construction. Let $S = \emptyset$ be the set of repeat spanning intervals in $E_{rsi}$ that have been iterated over.

1. Add $e$ to $S$.

2. For each $r_i$ in $o(e)$, add a vertex $r_{e,i}$ to $V'$, and set $\mu'(r_{e,i}) = 1$.

3. Decrement $\mu'(r_i)$ by 1 for all $i \in [k]$.

4. For $i \in \{0, 1, \ldots, k-2\}$, add an adjacency from $r_{e,i}$ to $r_{e,i+1}$ to $E'$.

5. Add adjacencies $\{u, r_{e,0}\}$ and $\{r_{e,k-1}, v\}$ to $E'$.

6. Delete edges $\{u, r_0\}$ and $\{r_{k-1}, v\}$.

7. If the degree of an unoriented repeat $r_i$ in $H'$ exceeds $2\mu'(r_i)$, delete all adjacencies to it which are compatible with a repeat spanning interval in $S$. This can be tracked by a simple boolean hash table over the set of adjacencies, and does not require the explicit construction of the set $S$.

8. If the degree of an oriented repeat $r_i$ in $H'$ exceeds $\mu'(r_i) + 1$, delete all adjacencies to it which are compatible with a repeat spanning interval in $S$, except for the adjacency with its mate $\bar{r}_i$.

An example of this construction is given in Figure 4.1.

The new instance $(H', \mu')$ can be seen to have no repeat spanning intervals, since none were added to $E'$ at any step of the construction. In the specific case of Theorem 4.1, this instance does not have a repeat contained in an interval, since all repeats were themselves only contained in adjacencies or repeat spanning intervals. This is not a requirement of the following result.

**Lemma 4.1.** *The instance $(H, \mu)$, in which repeats contained in repeat spanning intervals are not contained in other intervals, is realizable in the genome model $\mathcal{G}$ if and only if the instance $(H', \mu')$, $H = (V', E')$, is realizable in $\mathcal{G}$ and $\mu'(v) \geq 0$ for all vertices $v \in V'$.*

This lemma shows a simple but powerful result: if we can decide the realizability of $(H', \mu')$, an instance without repeat spanning intervals, then we can decide the realizability of $(H, \mu)$. In the case of Theorem 4.1, there is a strong structural restriction on the instance. We know that when all hyperedges are adjacencies, the decision problem is tractable (c.f. Theorem 3.2), and the only difference is that we are allowing intervals with non-repeats.

The proof of the lemma is relatively simple. One can use the sequence associated to a repeat spanning interval to restrict the possible genome maps that can serve as realizations of the instance. Since these sequences can be interpreted as paths, this serves to establish a correspondence between the realizations of the instance $(H, \mu)$, and those of $(H', \mu')$.

*Proof.* Assume $(H', \mu',)$ is realizable in $\mathcal{G}$ and that $\mu'(v) \geq 0$ for all $v \in V'$. Then there exists a set of linear/circular sequences on the alphabet of vertices such that every adjacency $e \in E'$ occurs in a sequence, and no vertex $v$ appears more than $\mu'(v)$ times, i.e. a realization for the instance. Let this realization be $M'$.

Every repeat $r \in V_R$ maps to a subset of $V'$, consisting of $r$ and the vertices added to $V'$ whenever $r$ appeared in a repeat spanning interval. Let $\phi\colon V \to 2^{V'}$ be this map (mapping non-repeats in $V$ to their corresponding copies in $V'$, and mapping repeats to the corresponding subset of vertices introduced in $V'$, except for the original copy of the repeat itself), and let $\phi^{-1}$ be the inverse map from vertices in $V'$ to $V$.

The construction specifies that for every $e \in E_I$, one introduces a path in $H'$ that corresponds to $o(e)$. Let this path describe the string $\mathbf{p} = v_0. \ \ldots \ v_{k-1}$. Take a substring $s \in M'$ such that $s = \mathbf{p}$. This substring is guaranteed to exist in $M'$, since each vertex in the path is a non-repeat, and every edge in the path must be compatible with the realization $M'$. Looking at the vertices in this path and applying the inverse map $\phi^{-1}$ to each element of $s$ yields a sequence of vertices in $V$ which corresponds exactly to $o(e)$. All the other hyperedges $e \in E'$ are already contained in $E$. This translates the linear walks in $H'$ which are used to define $M'$, into a set of linear walks on $H$, say $M$. Thus, every hyperedge in $E'$ corresponds to a hyperedge in $E$, or it is an adjacency which is compatible with a repeat spanning interval in $E$.

Finally, we need to check the multiplicities of the vertices. Assume, for every repeat $r$ in $V'$, $\mu'(r) \geq 0$, and the instance $(H', \mu')$ is realizable in $\mathcal{G}$. For a vertex $v \in V$, the cardinality of the set $\phi(v)$ cannot exceed $\mu(v)$, since otherwise, we would have had $\mu'(v) = \mu(v) - |\phi(v)| < 0$. Thus, the number of occurrences of a vertex $r \in V_R$ in $M$ is at most $\mu(r)$. This, combined with the fact that every hyperedge in $E$ is compatible with $M$, proves that $(H, \mu)$ is realizable in $\mathcal{G}$.

In the other direction, if $(H, \mu)$ is realizable in $\mathcal{G}$, we can find a realization such that for every repeat spanning interval $e \in E_I$, the order $o(e)$ appears in this realization. Replace the repeats in this genome map by new copies having multiplicity 1, as during the algorithm. This corresponds to a realization of $(H', \mu')$, which gives a realization in $\mathcal{G}$, with $\mu(v) = \mu'(v) + |\phi(v)|$ as defined before. This ensures that $\mu'(v) \geq 0$, since we know that $|\phi(v)| \leq \mu(v)$, a fact that is obvious since we can only have at most $\mu(v)$ copies of every vertex $v$ in $M$, and we are only relabelling them. $\square$

Note again that this result is independent of the structure of the instance $(H \setminus E_{rsi}, \mu)$. This implies that Lemma 4.1 works even if the original instance does not conform to the structural restrictions of Theorem 4.1.

### 4.1.2 Decomposing the instance

Let us get back to the original instance of the problem $(H, \mu)$, in which we know that repeats are only present in adjacencies or repeat spanning intervals. Using Lemma 4.1, we will convert it to an instance in which repeats are only contained in adjacencies. Let this modified instance be $(H', \mu')$, where $H' = (V', E')$.

The next lemma shows that if the instance $(H', \mu')$ which we constructed is realizable, all instances defined on connected subhypergraphs induced by non-repeats must be realizable in a linear or circular genome model. In order to prove this, we start from the following construction. Let $(H_u[V_u], \mu_u)$, where $H_u(V_u) = (V_u, E_u)$, be the instance induced on the vertex set $V_u = \{v \in V' : \mu'(v) = 1\}$, where $\mu_u$ is the restriction of $\mu'$ to $V_u$. This instance consists of a set of connected components, with only non-repeats. For each such connected component $C = (V'_c, E'_c)$, with restriction $\mu'_c$ of $\mu_u$ to $V'_c$, define an instance $(H_c, \mu_c)$, where $H_c = (V_c, E_c)$ and a map $\phi_c$ as follows.

1. Initialize $V_c = V'_c$, $E_c = E'_c$ and $\mu_c = \mu'_c$.
2. For every repeat $r$ in $(H', \mu')$ adjacent to a vertex $u$ in $V'_c$, add a vertex $r_{c,u}$ in $V_c$, $\mu_c(r_{c,u}) = 1$. Define $\phi_c(r_{c,u}) = r$.
3. Add adjacencies $\{r_{c,u}, u\}$ to $E_c$ for all adjacencies $\{r, u\} \in E'$, and vertices $u$ in $V'_c$.

The function $\mu_c$ maps all vertices in $V_c$ to 1. Once this procedure is completed for every connected component $C$, we get a family of instances which only have non-repeats, and a set of maps from the repeats in $V'$ to the vertices in each instance of this family. Let us call this family $\mathcal{C}$. We can also define $\phi_c^{-1}$ as a map that takes a repeat $r$ and a non-repeat $u$, and maps to $r_{c,u}$ if the vertex was introduced during the construction, and to $\emptyset$ otherwise. Then, having defined the family of instances $\mathcal{C}$, we have the following result on it.

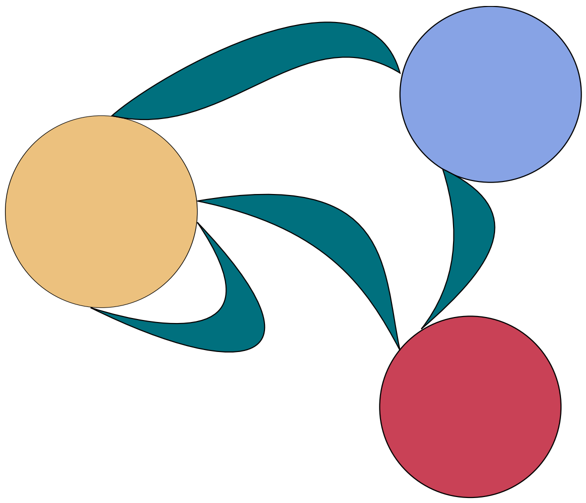


Figure 4.2: Decomposition of an instance. Each of the circles represents a repeat cluster, and the connections between them represent non-repeats connecting these clusters. Lemma 4.2 claims that if the entire instance is realizable in some genome model, each of the connecting non-repeat components form an instance which is realizable in the linear genome model.

**Lemma 4.2.** *If the instance $(H', \mu')$ is realizable in a genome model $\mathcal{G}$, then every instance $(H_c, \mu_c)$, $H_c = (V_c, E_c)$ in the family $\mathcal{C}$ must be realizable in the linear genome model.*

The claim made in this lemma can be visualized in Figure 4.2.

*Proof.* Let $M$ be a genome map in $\mathcal{G}$ such that $M$ is a realization of $(H', \mu')$. Assume without loss of generality that every repeat in this map appears at least twice. Construct a set of linear or circular sequences $\mathcal{M}'$ as follows.

1. Delete all adjacencies in $M$ which do not involve a non-repeat.
2. If there is an occurrence of an undoubled repeat $r$ which is adjacent to two non-repeat in $M$, select an arbitrary doubling of $r$ and delete the adjacency between the new oriented vertices.
3. Delete repeats except those which are adjacent to a non-repeat.

This gives us a set of linear sequences, framed by occurrences of repeats if any. We claim that for every $(H_c, \mu_c)$ in $\mathcal{C}$, there exists a realization in $M' \in \mathcal{M}'$ which can be obtained by relabelling using $\phi_c^{-1}$.

First, notice that every vertex in $V_c$ which is not in the domain of the map $\phi_c$ must be a non-repeat in $V'$. This set of vertices, which we call $U$, must be contained in a consecutive subsequence in $\mathcal{M}'$. By definition, the vertices in $U$ form a maximal connected component in the instance $(H[V_u], \mu_u)$. If the vertices in $U$ are not present as a consecutive subsequence, there was a repeat $r \in V' \setminus U$ such that $\{r, u\}$ was an adjacency compatible with $M$ which broke up a consecutive subsequence. But in that case, there is some interval or adjacency containing $u$ and another vertex $v \in U$ which is not compatible with $M$. This contradicts the fact that $M$ is a realization of $(H', \mu')$, which proves that such a consecutive subsequence must exist.

Since every adjacency/interval in $E_c$ induced by $U$ is also present in $E'$, they must also be compatible with $M'$. Now, for the vertices in the domain of $\phi_c$, every vertex $r_{c,u}$, which is adjacent to $u \in U$, is mapped to a repeat $\phi(r_{c,u}) \in V'$. Since $M$ is a realization for the instance $(H', \mu')$, applying the inverse map $\phi_c^{-1}(r, u)$ to every occurrence of such a repeat in the sequence $M'$ will result in a sequence, say $\phi_c^{-1}(M')$, such that the adjacency $\{u, r_{c,u}\}$ is compatible with it. This proves that all adjacencies and intervals in $E_c$ are compatible with $\phi_c^{-1}(M')$.

Another conclusion from this is that there can be at most 2 repeats from $V'$ with corresponding copies in $(H_c, \mu_c)$. Otherwise, there would have to be at least 3 repeat copies $r_{c,u}, r'_{c,v}, r''_{c,w}$ adjacent to 3 vertices, say $u, v, w$, in $U$. Since repeats in $E'$ only occurred in adjacencies, there is no interval in which all of $u$,$v$ $w$ and any of the repeat copies are contained. However, the structure of $\phi_c^{-1}(M')$ states that $u$, $v$ and $w$ occur within a consecutive subsequence of non-repeats, framed by two repeats. An adjacency to a third repeat from a vertex in $U$ would not be compatible with $M$.

Clearly, since $\phi^{-1}(\mathcal{M}')$ only consists of linear sequences, the instance $(H_c, \mu_c)$ is compatible with a map in the linear genome model. To see that $\phi_c^{-1}(M')$ is consistent with $(V_c, \mu_c)$, note that $V_c$ only has non-repeats, and $\phi_c^{-1}(M')$ has exactly 1 copy of a vertex. So, $\phi_c^{-1}(M')$ is be a realization of $(H_c, \mu_c)$ in the linear genome model. This proves the lemma.

□

Lemma 4.2 can be used to check if the given instance is not realizable in $\mathcal{G}$.

### 4.1.3 Putting the pieces together

After checking if each instance in the family $\mathcal{C}$ constructed in the previous section is realizable in the linear genome model, we transform $(H', \mu')$ into a new instance $(H'', \mu'')$, where $H'' = (V'', E'')$, as follows. Recall the set $\mathcal{C}$, containing instances of the form $(H_c, \mu_c)$ as constructed leading up to Lemma 4.2.

1. Set $V''$ to the union of the repeats and the set of frontier vertices in $V'$.

2. Set $\mu''$ to be the restriction of $\mu'$ on $V''$.

3. Set $E''$ to the set of all adjacencies in $E'$ which only contain vertices in $V''$.

4. For all remaining non-repeats $u, v \in V''$, if both $u$ and $v$ are found in some instance $(H_c, \mu_c)$ in $\mathcal{C}$, merge $u$ and $v$ into a single vertex $v_c$, and set $\mu''(v_c) = 1$. If $u$ or $v$ were adjacent to any vertex $w \in V''$, add the adjacency $\{v_c, w\}$ to $E''$.

Note that since the intervals in $E'$ only involved non-repeats, $E''$ only consists of adjacencies. This means that we can check for an Eulerian tour[1] in this instance to decide if it is realizable in $\mathcal{G}$ using Theorem 3.2. The exact routine for doing this varies with the genome model $\mathcal{G}$, but the principle remains essentially the same.

### 4.1.4 The algorithm

The algorithm can now be specified in very simple terms. Let the input instance be $(H, \mu)$, $H = (V, E)$ in which repeats are only involved in adjacencies or repeat spanning intervals, and assume we are testing for realizability in the genome model $\mathcal{G}$.

1. From $(H, \mu)$, use Lemma 4.1 to create an instance $(H', \mu')$ without repeat spanning intervals. If $\mu'(v) < 0$ for any vertex $v \in V'$, return No.
2. In $(H', \mu')$, test if every derived instance in the set $\mathcal{C}$, as defined in Lemma 4.2, is realizable in the linear genome model. If not, return No. This can be done in polynomial (linear) time and space using Theorem 3.1, since the vertices in all the instances have multiplicity 1.
3. Contract the instances in $\mathcal{C}$ into non-repeats, as specified in the construction preceding Lemma 4.3, to get an instance $(H'', \mu'')$ consisting only of adjacencies. Use Theorem 3.2 to check if this instance is realizable in $\mathcal{G}$. If so, return Yes. Else, return No.

The detailed pseudocode for the algorithm, which specifies the details of all the steps, is given in Appendix A.

### 4.1.5 Analysis

#### Algorithm correctness

The first step of the algorithm follows from Lemma 4.1, which states that a map of the instances $(H, \mu)$ is also a map of $(H', \mu')$ and vice-versa. In the second step, if every instance $(H_c, \mu_c)$ in the set $\mathcal{C}$ is realizable in the linear genome model, as stated in Lemma 4.2, then we can contract these instances into non-repeats, laying the ground for the next step.

[1]The term 'Eulerian tour' may be misleading, because we actually search for an Eulerian tour in an auxiliary graph which can be constructed in linear time, depending on the genome model being used. An example was provided in Theorem 3.2.

The final lemma specifies a necessary and sufficient condition for the instance $(H', \mu')$ to be realizable in genome model $\mathcal{G}$, given $(H'', \mu'')$ and the results of checking if every instance $(H_c, \mu_c)$, as defined for Lemma 4.2, is realizable.

**Lemma 4.3.** *$(H', \mu')$ is realizable in genome model $\mathcal{G}$ if and only if $(H'', \mu'')$ is realizable in genome model $\mathcal{G}$ and each instance $(H_c, \mu_c)$ is realizable in the linear genome model.*

*Proof.* Consider a map $M$ for $(H', \mu')$. In this map, Lemma 4.2 proves that each instance $(H_c, \mu_c)$ must be realizable in the linear genome model. So, we can find a consecutive substring of non-repeats corresponding to a realization of $(H_c, \mu_c)$ in $M$. Contract this substring into a single vertex $v_c$. This gives a realization of $(H'', \mu'')$ in the genome model $\mathcal{G}$, since it contains all the adjacencies in $E''$, and the number of vertex occurrences does not increase.

In the other direction, consider a realization $M''$ of $(H'', \mu'')$ in the genome model $\mathcal{G}$. A realization $M_c$ of every $(H_c = (V_c, E_c), \mu_c)$ is framed by copies of vertices corresponding to repeats in $V'$ under the map $\phi_c$. Remove these framing vertices to get a linear sequence $M'_c$, and replace the occurrence of $v_c$ in $M''$ by the sequence $M'_c$.

This gives a map $M$ which is consistent with $(V', \mu')$, since each repeat $v$ occurs at most $\mu''(v) = \mu'(v)$ times, and each non-repeat occurs exactly once. Furthermore, every adjacency which was present in $E''$ was also present in $E'$, and is compatible with the new map. The rest of the adjacencies and intervals were present in $E_c$, and are also compatible with the map. Thus, the new map is a realization of $(H', \mu')$.

□

Using Lemma 4.3, if all the instances in $\mathcal{C}$ and the instance $(H'', \mu'')$ have been verified to be realizable, the instance $(H', \mu')$ is proved to be realizable in the model $\mathcal{G}$, which in turn implies that $(H, \mu)$ is realizable in $\mathcal{G}$.

## Algorithm complexity

Assuming we have $|V_R| = \rho$ and $\bar{\mu} = \max_{v \in V_R} \mu(v)$, we can observe that we add at most $\rho\bar{\mu}$ new vertices to get the hypergraph $H' = (V', E')$ in the first step of the algorithm. We also add 2 edges per new vertex. So, we can conclude that $|V'| = n + \bar{\mu}\rho$, and $|E'| = m + 2\bar{\mu}\rho$, where $n, m$ are the number of vertices and hyperedges in the input. In general, $\bar{\mu} << n, m$, so the instance can be constructed in time and space linear in its size. Since we need only iterate through the repeat spanning intervals, and check their content, the time taken can be at most $O(m + 3\bar{\mu}\rho)$.

The second step, which checks if all instances $(H_c, \mu_c)$, as defined in Lemma 4.2, are realizable in the linear genome model, can be performed in time and space linear in the size of the instance. This is because the total number of vertices in these instances is of the order of $|V'|$, and no adjacencies or intervals are added. Then, we can use Theorem 3.1 to check if each of them is realizable or not, and this routine runs in linear time and space [24], having a time complexity of at most $O(n+m+\xi-\rho)$, where $\xi = \sum_{e\in E}|e|$. The contraction of the non-repeats and the intervals can be performed in time linear in the number of instances in the set $\mathcal{C}$, which can be at most the number of non-repeats in the instance, and we do this for all such instances. So, the instance $(H'', \mu'')$ can be constructed in time $O(n-\rho)$.

We finally have to check if there is an Eulerian cycle in the instance $H'', \mu''$. This can be done in linear time and space, i.e. $O(|V'|+|E'|)$ (since $|V''| \leq |V'|$ and $|E''| \leq |E'|$) [84]. So, one can check for Eulerian cycles, in $(H'', \mu'')$, in time $O((n+\bar{\mu}\rho)+(m+2\bar{\mu}\rho))$, i.e. polynomial in the size of the input. So, the total time complexity is $O(3n+3m+6\bar{\mu}\rho+\xi-2\rho)$, or, dropping the negative term and absorbing the constants, $O(n+m+\bar{\mu}\rho+\xi)$.

The space complexity can be similarly bounded. We already know that the number of vertices and edges in $H' = (V', E')$ is $n+\bar{\mu}\rho$ and $m+2\bar{\mu}\rho$ respectively, while the cumulative size of the instances $(H_c, \mu_c)$ is guaranteed to be at most $n+m+\xi$. The final instance $(H'', \mu'')$ also has at most $n+\bar{\mu}\rho$ vertices and at most $m+2\bar{\mu}\rho$ edges. So, assuming that we do not reuse any space and by absorbing the constants in the notation, the total space complexity is $O(n+m+\bar{\mu}\rho+\xi)$. Note that this is a gross overestimate, since ideally the instance $(H'', \mu'')$ will be much smaller than $(H, \mu)$, and each $(H_c, \mu_c)$ can be collapsed once it is checked if the instance is realizable in the linear genome model.

## 4.2 Fixed parameter tractability of realizability

The next result is of a more theoretical nature. It provides an algorithm that decides if an instance is realizable in a given genome model, with the runtime complexity being polynomial provided that the number of repeats and the maximum multiplicity are bounded.

**Theorem 4.2.** *Let $(H, \mu)$ be an instance in which $\bar{\mu} = \max_{v\in V}\mu(v)$, $\rho$ is the total number of repeats in the instance, $\Delta_e$ is size of the largest hyperedge, and $\Delta_v$ is the maximum degree of a vertex. Let $\xi = \sum_{e\in E}|e|$ be the sum of the sizes of all hyperedges. It is possible to decide if the instance $(H, \mu)$ is realizable in the linear genome model with time complexity $O\left((\Delta_v(\Delta_e+\rho\bar{\mu}))^{2\rho\bar{\mu}}(\Delta_e+\bar{\mu}\rho)(n+m(1+\bar{\mu}\rho)+3\rho\bar{\mu}+\xi)\right)$ and space complexity $O(n+m+\bar{\mu}\rho(m+3)+\xi)$.*

The complexity is exponential, as expected. However, the exponent itself only depends on the number of repeats and the multiplicity, and not on the size of the hyperedges. This is expected behaviour; after all, in the absence of repeats, the problem is solvable in polynomial time. From a practical point of view, though, this means that as long as the number of repeats is small, and the multiplicity function is bounded, we can use a reasonably naive algorithm to decide Problem 3.1 on the input instance. Notably, this result holds for the linear genome model, and since both the circular and the mixed genome models are weaker,

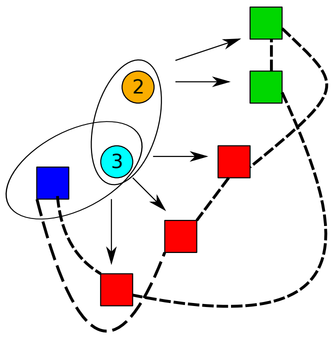


Figure 4.3: Constructing an instance without repeats. Each repeat, represented by circular vertices, is replaced by as many non-repeats as its multiplicity, as in the case of the green and red vertices added here. For each such new vertex, 2 'neighbours' are chosen, as per a mapping, defined in (4.1). This defines a set of new edges, which we call $f$-edges. Here, the $f$-edges are depicted as dashed edges. In this particular case, the resulting hypergraph forms a cycle, and is thus not realizable in the linear genome model. However, there may be another set of $f$-edges for which the resulting instance is realizable.

they too can be decided by the following algorithm, as can the realizability of an instance with ordered intervals.

The main idea of the algorithm is to transform the input instance into an instance where the multiplicity function maps vertices to 1. Then, one can use Theorem 3.1 to check if the modified instance is realizable in the linear model (i.e. checking if it has the C1P). Here is where the notion of repeat clusters proves to be useful again. If we had no idea of where the repeats will occur, we would have to check all possible sequences that can be created using $\mu(v)$ copies of a vertex $v$. But repeat clusters localize the concentration of repeats, and this lets us restrict the number of sequences to generate, while the non-repeats somehow make the remaining instance 'rigid'.

### 4.2.1 Constructing repeat-free instances

Let $(H, \mu)$, $H = (V, E)$ be the input instance. Note that for every repeat $v \in V_R$ in this instance, if there exists a realization in the genome model $\mathcal{G}$ which contains $\lambda_v < \mu(v)$ occurrences of $v$, then there is a realization which contains $\mu(v)$ occurrences in $\mathcal{G}$ as well. This is a trivial observation- take the original realization with $\lambda_v$ occurrences of $v$, and simply concatenate $\mu(v) - \lambda_v$ more copies of $v$ to any one of the original occurrences. For oriented vertices, concatenate $\mu(v) - \lambda_v$ occurrences of the string $v.\bar{v}$ and insert it at the original occurrence position instead.

The point of the above observation is that it is sufficient for us to find a realization which has $\mu(v)$ occurrences of every vertex $v \in V$. We shall create a set of instances $(H_f, \mu_f)$, where $H_f = (V_f, E_f)$, in which the function $\mu_f$ maps all vertices in the set $V_f$ to 1. $V_f$ is defined as follows.

$$V_f = (V \setminus V_R) \cup \bigcup_{v \in V_R} R'(v),$$

where the set $R'(v)$ is defined for a repeat $v \in V_R$ as $\{v_i : v \in V_R, 0 \leq i < \mu(v)\}$. Let $R' = \bigcup_{v \in V_R} R'(v)$.

The *open neighbourhood* of a vertex $v \in V$, denoted by $N(v)$, is the set of all vertices in $V$, excluding $v$, which appear in some hyperedge in $E$ which also contains $v$. The *modified neighbourhood* of $v \in V$ is the following set.

$$N'(v) = \{u \in V \setminus V_R : u \in N(v)\} \cup \bigcup_{w \in (V_R \cap N_v) \cup \{v\}} R'(w).$$

In a genome map which contains all vertices in $V_f$ exactly once, every vertex must be adjacent to at least 1 and at most 2 other vertices in the map. On the basis of this, we can allocate 'neighbours' to each vertex, their predecessor and successor in the sequence corresponding to the map. For every repeat vertex $v \in V_R$, we can define this relation as the following mapping.

$$f_v \colon R'(v) \to S_v, \tag{4.1}$$

where $S_v = \{\{u_0, u_1\} : u_0, u_1 \in N'(v) \cup \{\emptyset\}\}$, which associates the vertices in $R'(v)$ to their adjacent vertices in the sequence. We use $f \colon V_R \to S'$ to denote the map obtained by taking the union $\bigcup_{v \in V_R} f_v$, where $S' = \bigcup_{v \in V_R} S_v$. The map can be represented as a set of edges between the vertices in $R'(v)$ and those in $N(v)$ for all $v \in V_R$, as shown in Figure 4.3.

### 4.2.2 The algorithm

We now describe the main algorithm. The algorithm includes the complete description of the hypergraphs $H_f = (V_f, E_f)$ for all $f$, and specifies the choice of edges.

1. Define $V_f = (V \setminus V_R) \cup \bigcup_{v \in V_R} R'(v)$. Set $\mu_f(v) = 1$ for all $v \in V_f$.
2. For each repeat $v \in V_R$, choose a mapping $f_v \colon R'(v) \to S_v$ as defined in (4.1). This defines a map $f \colon V_R \to S'$.
3. For a given $v_i \in R'(v)$ for a repeat $v \in V_R$, if $f(v_i) = \{u_0, u_1\}$ (where one or both of $u_0$ and $u_1$ may be empty vertices), add the edges $\{v_i, u_0\}$ and $\{v_i, u_1\}$ to $E_f$. Call the set of edges added $f$-*edges*.
4. For each edge $e \in E$, add an edge $e' = e \setminus V_R$ to $E_f$.
5. For each vertex $u \in V_f \setminus R'$ which is adjacent to a vertex $v^0 \in R'$, there exists a unique path $u.v^0.\ldots.v^{k-1}.w$, where all $v^i \in R'$ for all $i \in [k]$, $u, w \in V_f \setminus R'$, and each of $\{u, v^0\}$, $\left\{v^{k-1}, w\right\}$ and $\{v^i, v^{i+1}\}$, $i \in [k-1]$ are $f$-edges. Add all $v^i$ in this path to each $e \in E_f$ such that $u \in e$. By symmetry, each of the $v^i$ are also contained in $e \in E_f$ such that $w \in e$.
6. For each $e \in E$, check if the corresponding edge $e'$ which was added to $E_f$ has vertices from $R'(v)$, where $v \in V_R \cap e$. If not, continue to the next mapping $f$.
7. Use Theorem 3.1 on $(H_f, \mu_f)$ to check for realizability in the desired genome model. If the algorithm returns Yes, output Yes and exit.
8. Iterate over all possible sets of neighbours in Step 2.
9. Output No if there is no instance $(H_f, \mu_f)$ which is realizable in the genome model.

### 4.2.3 Analysis

#### Algorithm correctness

We base the correctness of the algorithm on the following claim.

**Claim.** *$(H, \mu)$ has a realization in the genome model $\mathcal{G}$ if and only if there exists $f$ such that $(H_f, \mu_f)$ has a realization in $\mathcal{G}$.*

In order to prove this claim, let us first assume that the instance $(H, \mu)$ is realizable in $\mathcal{G}$, and that there exists a realization $M$ of the same. In this realization, we may assume that every vertex $v \in V$ appears exactly $\mu(v)$ times. Otherwise, as stated before, we can add tandem copies of a vertex next to a known occurrence of it in $M$ without destroying any adjacencies or intervals. Alternately, we could just add these copies as a separate sequence, without modifying $M$. Once this is done, replace the occurrences of a repeat vertex $v \in V_R$

by copies $v_i \in R'(v)$, where $R'(v) = \{v_i : 0 \leq i < \mu(v)\}$. This yields a new map, $M'$. Clearly, the vertices occurring in $M'$ are exactly the vertices in the vertex set $V_f$. Note that this holds irrespective of the choice of $f$.

We will show that $M'$ is the realization of a particular instance $(H_f, \mu_f)$ which we construct in the algorithm. In $M'$, each vertex in the set $R'$ is adjacent to either 1 or 2 vertices. Thus, for every $v_i \in R'$, there is a map $v_i \mapsto \{u', u''\}$, where $u', u'' \in V_f \cup \{\emptyset\}$. This defines the map $f$, which can be partitioned into $f_v$ according to the repeats $v \in V_R$. The map $f$ also uniquely defines the instance $(H_f, \mu_f)$ which we shall prove is realizable in $\mathcal{G}$ with realization $M'$. First note that $M'$ is certainly consistent with $(V_f, \mu_f)$, since every vertex in $V_f$ occurs exactly once in $M'$, the necessary condition enforced by the all-ones multiplicity function $\mu_f$.

It only remains to see that the set of edges $E_f$ is compatible with $M'$.

1. The $f$-edges in the instance must be compatible with $M'$, since they are precisely the edges defined by the map $f$.

2. All hyperedges $e \in E_f$ which only contain vertices from $V_f \setminus R'$ must be compatible, as they correspond to hyperedges in $(H, \mu)$ which did not contain any repeats.

3. For the rest of the edges, which were constructed in Step 5 of the algorithm, the vertices in these edges are precisely the copies of repeats which were present in the analogous hyperedges in $(H, \mu)$, along with vertices that form chains of $f$-edges. Otherwise, they are discarded in Step 6 since some vertex in the edge is missing. Since the original hyperedges were compatible with $M$, and each $f$-edge is also compatible with $M$, these hyperedges must be compatible with $M'$.

This proves that $(H_f, \mu_f)$ is realizable in $\mathcal{G}$, which concludes the first part of the proof.

In the other direction, let us assume that there is an instance $(H_f, \mu_f)$ which is realizable in $\mathcal{G}$, and has realization $M'$. This realization contains all vertices in $V \setminus V_R$, and vertices in $R'(v)$ for repeats $v \in V_R$. Replace every occurrence of $v_i \in R'(v)$ in $M'$ by $v$ for every $v \in V_R$ to get the map $M$. Since there were at most $\mu(v)$ vertices in $R'(v)$, the vertex $v$ does not appear more often than $\mu(v)$ times in $M$. To check that all edges in $E$ are compatible with $M$, we observe the following.

1. Every edge $e \in E$ which consists only of vertices from $V \setminus V_R$ is compatible with $M$, since it was compatible with $M'$.

2. For every other edge $e \in E$, if there exists a repeat $v \in e$, then there exists an edge $e' \in E_f$ such that there is a copy $v_i \in R'(v)$ which is in this edge. This is assured by Steps 5 of the algorithm, since every $v_i$ which is in a path of $f$-edges with one end in $e'$ is also contained in $e'$. Since the neighbours of $v_i$ are assured to be in $N'(v)$, this means that the compatibility of $e'$ translates into the compatibility of $e$ with $M$. Otherwise, the map $f$ is skipped, and we do not check it, as we can see in Step 6.

This proves that $(H, \mu)$ is realizable in $\mathcal{G}$, which finishes the claim and the proof of correctness.

### Algorithm complexity

The main complexity follows from the number of possible mappings $f$ that can be used to construct the auxiliary instances $(H, \mu_f)$. These mappings denote choices of neighbours for each of the vertices in the set $R'$. If $\bar{\mu}$ is the maximum value attained by the multiplicity function, we get a total of $\bar{\mu}\rho$ vertices in $R'$. If each vertex is contained in at most $\Delta_v$ edges, and each edge contains at most $\Delta_e$ vertices, the number of mappings $f$ is at most $O\left(\binom{\Delta_v(\Delta_e+\bar{\mu}\rho-1)}{2}^{\bar{\mu}\rho}\right)$. We can cycle through these mappings by iterating through the list $N'(v)$ for every repeat $v \in V_R$ and allocating 2 neighbours (or 1 neighbour and a null element) to each vertex in $R'(v)$, which is where the exponential complexity arises from. There are many ways to enumerate the set of mappings $f$, including probabilistic methods which can be derandomized [4, 165].

One non-probabilistic way to implement it is to construct a graph consisting of all vertices in $V_f$, and edges from each vertex in $R'(v)$ for a repeat $v$, to all vertices in $S_v$. A maximal 2-matching in this graph, using the notation of Dessmark et al. [58], corresponds to a single mapping $f$, since it associates every vertex to 2 adjacent vertices. The problem of listing all such mappings is the same as that of iterating through the maximal matchings in this graph. There are algorithms that can do this in time proportional to the number of matchings times the maximum degree of the graph [123, 203], which is the same as the number of mappings times the maximum number of vertices in an edge in the constructed instance, which is $\Delta_e + \bar{\mu}\rho$. So, the set of mappings can be enumerated in time $O\left(\binom{\Delta_v(\Delta_e+\bar{\mu}\rho-1)}{2}^{\bar{\mu}\rho}(\Delta_e + \bar{\mu}\rho)\right)$.

Following this, the actual translation to the instance $(H_f, \mu_f)$ is relatively cost-efficient. The set $V_f$ only needs to be constructed once, and can be done in time $O(\bar{\mu}\rho)$. The construction of the set $E_f$ can be done by first iterating through the set of hyperedges in $E$, and then iterating through the function $f$ in order to add the $f$-edges and allocate the vertices in $v_i$ to the added edges. The number of vertices from $V'$ added in an edge $e \in E_f$ is at most $\bar{\mu}\rho$, and we do this for every edge in $E_f$ which is not an $f$-edge. Finally, using Theorem 3.1, the total time taken to test the constructed instance for realizability in the genome model $\mathcal{G}$ is in the order of $O(n + \rho(\bar{\mu} - 1) + m + 2\bar{\mu}\rho + \xi + m\bar{\mu}\rho)$. Iterating over all the $O\left(\binom{\Delta_v(\Delta_e+\bar{\mu}\rho-1)}{2}^{\bar{\mu}\rho}\right)$ possible maps $f$, and ignoring constants, we get a runtime in the order of $O\left((\Delta_v(\Delta_e + \rho\bar{\mu}))^{2\rho\bar{\mu}}(\Delta_e + \bar{\mu}\rho)(n + m(1 + \bar{\mu}\rho) + 3\rho\bar{\mu} + \xi)\right)$.

For space complexity, note that $V_f = n + \bar{\mu}\rho$ and $E_f = m + 2\bar{\mu}\rho$. Since the cumulative size of the hyperedges is at most $\xi + m\bar{\mu}\rho$, the space complexity for deciding the realizability of the instance $(H_f, \mu_f)$ is $O(n + m + \bar{\mu}\rho(m + 3) + \xi)$. In order to iterate over all mappings $f$, instead of listing them all, they can be constructed on the go, as stated above. So, the total space needed is $O(n + m + \bar{\mu}\rho(m + 3) + \xi)$.

## 4.3 Improving decision algorithms

Theorem 4.1 proves to be a useful algorithm in its own right, as we shall see in later chapters. However, it relies greatly on the structure of the repeats in the instance. In the absence of repeat spanning intervals, deciding realizability becomes much more problematic, since we need to resolve both the internal order of the markers in a map and the ambiguity in the instance. It would be useful to have a more relaxed definition of repeat spanning intervals, one which can incorporate a little uncertainty in the associated sequence and yet capture the essential information for allowing a decision algorithm.

Theorem 4.2 is a relatively simple algorithm for deciding the realizability of an instance in a given genome model, but it comes at the cost of exponential runtime. There are many techniques in the theory of fixed parameter and exponential algorithms which allow increased performance [165], but they do not seem to be trivially applicable here. A significant improvement in performance would be a welcome addition to the toolbox of algorithms for deciding realizability. Indeed, even if the runtime is exponential, biological data often presents structure which can be exploited to make sure the runtime is still bounded. An alternate approach would be to parameterize by arguments other than the number of repeats and maximum multiplicity. A natural parameterization is not immediately obvious, and would be another useful equipment in the toolbox.

# Chapter 5

# Partial optimization problems on genome maps

Theorem 3.7 tells us that optimizing over an instance which contains intervals is NP-hard, even when the multiplicity function maps all vertices to 1. This holds independently of the genome model being used as well. So do we have no chance of disambiguating repeats using intervals?

From a practical perspective, of course, there are other resorts to this problem. As we mentioned in the previous chapter, we can always use heuristics to resolve the problem, by discarding intervals until a good subset is found. Alternately, when the multiplicity function maps all vertices to 1, there are FPT algorithms that can be used [65].

This chapter and the next present results on a different take on the issue. We define a modified optimization problem, which we motivate through the fact that Theorem 3.7 presents a positive tractability result for optimization when an instance only consists of adjacencies [138]. We show that under certain strong but practically useful conditions on the intervals, this problem can be solved in polynomial time, but the general problem is NP-hard. The result in this chapter was proved in collaboration with Cedric Chauve and Murray Patterson [41]. The conditions we impose on the intervals were suggested by Eric Tannier.

## 5.1 Partial optimization

Our solution to this problem is to use the tractability result in Theorem 3.7 to first find a set of good adjacencies such that the resulting instance is realizable. We take this instance as a skeleton, in that we look for a genome map that is a realization of this instance. By doing this, we can restrict the search space of genome maps by a considerable amount. Then, we try to find a maximum weight set of repeat spanning intervals that is compatible with at least one of these maps.

### 5.1.1 Formal definition and motivation

We state the problem as follows.

**Problem 5.1.** Let $(H, \mu, w)$, where $H = (V, E)$, be an instance such that $(H_A, \mu, w_A)$ is realizable in a given genome model $\mathcal{G}$. Let $E_I \subseteq E$ be the set of intervals, such that each $e \in E_I$ is a repeat spanning interval. Find a maximum weight subset $S \subseteq E_I$ such that the instance $(H', \mu, w')$, where $H' = (V, E_A \cup S)$ is realizable in $\mathcal{G}$.

The problem is a workaround to Problem 3.2, based on the fact that Theorem 3.7 offers an easy to implement algorithm for optimizing on adjacencies to obtain realizability in the mixed genome model. The point of the problem is to accept that the adjacencies that result from the first optimization step do indeed appear in the genome map that we want, and to use the set of repeat spanning intervals to pinpoint a less ambiguous map. The obvious question we have now is: can this be done efficiently?

One filter for the repeat spanning intervals is obvious. We want the realizations of the output instance to be a subset of those of the input instance. This means that if there is a repeat spanning interval $e$ such that $o(e) = u.r_0.\dots.r_i.r_j.\dots.r_k.v$, and there is no adjacency between the vertices $r_i$ and $r_j$ in the original instance, then $e$ cannot be present in an optimal set $S$. Using this, we can immediately discard repeat spanning intervals which would lead to such missing adjacencies. In this chapter and the next, we will always assume that the set of repeat spanning intervals is already filtered as explained.

We do have another advantage over Problem 3.2. The underlying adjacency instance of the input is known to have a realization in a chosen genome model. This is a strong structural restriction, one which we wish to exploit, and indeed plays a key role in the algorithm we shall present in this chapter.

**Overview of applications.** Problem 3.2 is aimed at resolving the two main problems brought about by repeats: genome map ambiguity and repeat organization. Repeat spanning intervals provide very specific information regarding repeats. Since they are ordered intervals, they carry information regarding repeat organization in a genome map. At the same time, being framed by non-repeats means that they fix the order in which two non-repeats appear. In effect, they point to an 'adjacency' between the non-repeats, or in other words, they predict that the unique markers corresponding to these vertices, when appearing on a genome map, can only have repeats from a single repeat cluster between them on the map, and no other markers.

Since repeat spanning intervals are highly structured, ideally one would like to be able to optimize over them to get a realizable instance in a given genome model. Then, a realization for the instance would be a genome map in which we have at least some organization in the order of the repeats, and hopefully most of the ambiguities can be removed. This is particularly helpful when we can infer repeat spanning intervals efficiently, as is the case in certain problem in palaeogenomics [177].

### 5.1.2 Multidimensional knapsack problems

Problem 5.1 is closely related to the multidimensional $(0,1)$–knapsack problem, which can be stated as follows.

**Problem 5.2.** Let $\mathcal{X} = \{\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}\}$ be a set of $n$ vectors in $\mathbb{Z}^d_{\geq 0}$, and let $w \colon \mathcal{X} \to \mathbb{R}^+$ be a weight function on the vectors. Then, given a vector $\mathbf{y} \in \mathbb{Z}_{\geq 0}$, find a solution to the following integer linear program.

$$\max_{\{b_0,\ldots,b_{n-1}\}} \sum_{i\in[n]} b_i w\left(\mathbf{x}_i\right)$$

subject to

$$\begin{aligned} b_i \in \{0,1\} & \quad \forall i \in [n]\,, \\ \sum_i b_i x_{i,j} \leq y_j & \quad \forall j \in [d]\,. \end{aligned}$$

This problem is NP-hard in general [113].

Problem 5.1, at a first glance, seems to be closely related to this problem. Each repeat spanning interval can be associated with a vector indexed by the vertices in the repeat cluster it spans. The vertices in this repeat cluster are the only ones that can be contained in a repeat spanning interval spanning it. The entry indexed by a vertex $v$ in this vector will be the number of times $v$ occurs in the order associated with that repeat spanning interval. Then, it almost seems a matter of formality to use this structure and conclude that Problem 5.1 is NP-hard as well. But there are two complications. First, the hardness for the multidimensional knapsack problem does not include structure that may arise from oriented vertices. In fact, we will exploit this structure to get a tractability result later in this chapter. Furthermore, we will see in Chapter 6 that there is a subtle technicality in Problem 5.1, arising out of the fact that we need to preserve all the adjacencies that were already present in the instance. We shall elaborate on this technicality in the next chapter, when we design a fixed parameter algorithm for the problem.

## 5.2 Tractability

Problem 5.1 can be solved in polynomial space and time as long as the repeat spanning intervals are minimal, and all frontier vertices correspond to oriented markers. Recall that minimal repeat spanning intervals are those whose associated sequence contains a single occurrence of an unoriented repeat, or a single occurrence of two oriented repeats which are each other's mates, framed by two non-repeats. This is the smallest possible type of repeat spanning interval, since no other repeat occurrences are possible within it.

**Theorem 5.1.** *Problem 5.1 can be solved in polynomial time and space for the mixed genome model if all the repeat spanning intervals are minimal and are framed by oriented non-repeats.*

Why is this result useful? While it is true that the immediate scope of the result seems small, we have to remember that repeats give rise to two different issues: genome map ambiguity and uncertainty in internal repeat organization. In the absence of very long syntenic information containing repeats, neither problem can be resolved.

However, it is often easier to get unordered interval data, and the instance may contain a number of 'partly ordered' intervals, in which we know that repeats from a specific repeat cluster in an adjacency instance are present. For example, we may always see a set of unique markers which are separated in the genome by a set of repeat markers which change only in order, or in which there are minor changes in content. We can infer that, barring some insertions, deletions and substitutions within the set of repeated markers, the sequence in this region is mostly conserved.

Knowing this, we can collapse the repeat cluster consisting of the vertices corresponding to the repeats we always find between the non-repeats into a single unoriented 'repeat' vertex. This allows us to define repeat spanning intervals that span this repeat. Optimizing over the set of repeat spanning intervals now implies that we are fixing an order for traversing this repeat cluster, which in turn refers to removing ambiguity in the genome map. So at least one of the problems due to repeats can be addressed in this manner. Such techniques have been used in genome assembly frameworks to remove ambiguity in the construction of assemblies, but have used a less formal framework [209, 214].

The proof for Theorem 5.1 proceeds in two stages. In the first stage, we prove a simple lemma which restricts the number of repeat spanning interval a repeat can occur in, depending on the number of vertices it is adjacent to. The second stage uses the structure of the instance to pose a strong condition on the number of repeat spanning intervals a vertex can occur in, assuming that the conditions of Problem 5.1 are satisfied. This key result uses the realizability of the underlying instance on adjacencies in order to understand the structure of the set $S$ of repeat spanning intervals that must be retained.

A corollary to Theorem 5.1 is the following statement.

**Corollary 5.1.** *Problem 5.1 can be solved in polynomial time and space for the mixed genome model if all non-repeats are oriented, and the maximum number of vertices in a repeat cluster is* 1 *if the repeat is unoriented, and* 2 *if the repeats in it are oriented.*

This corollary follows immediately from the fact that if the repeat cluster has only 1 unoriented vertex, or 2 oriented vertices, every repeat spanning must be minimal.

### 5.2.1 Constraints due to non-involved adjacencies

Consider a repeat $v$ which is adjacent to a vertex $x$ in the given instance. An important constraint in Problem 5.1 is that the adjacency $\{v, x\}$ must be compatible with a realization of the output instance. This serves as a cap on the number of repeat spanning intervals involving a given repeat, and is the basis for the following result.

**Lemma 5.1.** *Let $(H, \mu, w)$, $H = (V, E_A \cup E_I)$ be a realizable instance in the mixed genome model, such that every interval in $E_I$ is a repeat spanning interval. If $v \in V_R$ is adjacent to a vertex set $X$, such that for every $x \in X$, $x$ and $v$ do not appear together in any repeat spanning interval in $E_I$, then*

1. *if $v$ is an oriented repeat, it can appear in at most $\mu(v) - |X|$ repeat spanning intervals in $E_I$, and*

2. *if $v$ is a non-oriented repeat, it can appear in at most $\mu(v) - \lceil |X|/2 \rceil$ repeat spanning intervals in $E_I$.*

*Proof.* First assume that $v$ is an oriented repeat. Let $X$ be the set of vertices adjacent to $v$, and let $x \in X$ be one such vertex. There are two possible cases.

1. $x$ is a repeat. If so, either $x = \overline{v}$ or it is not the mate of $v$. Unless it is the mate of $v$, it cannot be in a minimal repeat spanning interval which contains $v$, and since there is no repeat spanning interval containing both $x$ and $v$, we can rule out that $x = \overline{v}$.

2. $x$ is a non-repeat.

Now, note that the edge $\{x, v\}$ does not appear in a repeat spanning interval, since, by the hypothesis, there is no repeat spanning interval containing both $x$ and $v$. Thus, in order for this edge to be compatible with a realization of the instance $(H', \mu, w')$, where $H' = (V, E_A \cup S)$ and $S \subseteq E_I$, it must be contained in $E_A$, as it cannot be compatible with any repeat spanning interval in $S$. In order to maintain realizability, there must be an occurrence of $v$ in the realization which is dedicated solely to make sure $\{x, v\}$ is compatible. This occurrence must also be adjacent to $\overline{v}$, since $v$ is an oriented repeat. So, it is not possible that this occurrence of $v$ forms part of a consecutive substring verifying the compatibility of other adjacencies or repeat spanning intervals. This is shown visually in Figure 5.1a. Thus, the number of remaining copies of $v$ which can be used up in repeat spanning intervals is $\mu(v) - 1$. Performing this analysis for all vertices $x \in X$, we can conclude that we need to allocate $|X|$ copies of $v$ for making sure that the adjacencies to vertices in $X$ are all compatible. So, only $\mu(v) - |X|$ copies remain which can correspond to occurrences of $v$ in repeat spanning intervals. This proves the first part of the lemma.

If $v$ is an unoriented repeat, the $\{v, x\}$ must still be compatible with a realization of the instance $(H', \mu, w')$, where $H' = (V, E_A \cup S)$ and $S \subseteq E_I$ is an optimal set of repeat

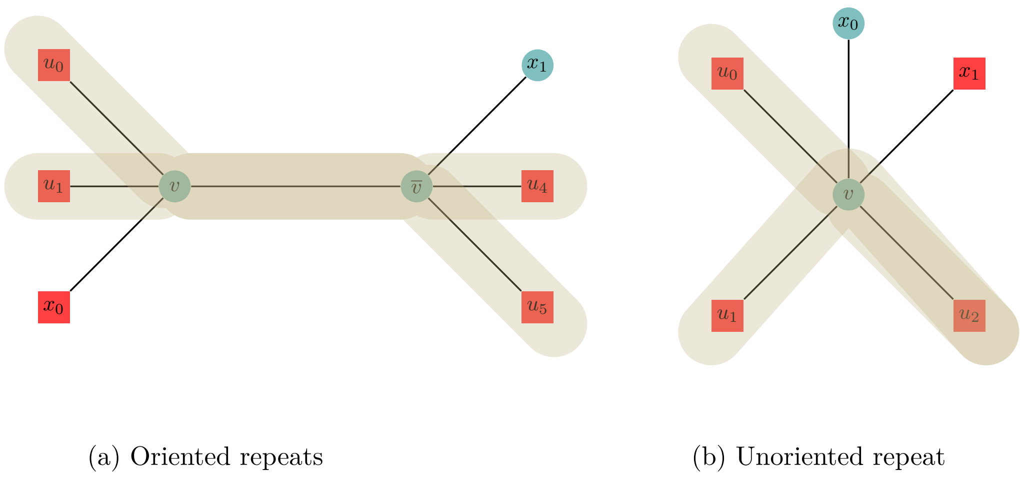


Figure 5.1: If a repeat is adjacent to a vertex which is not in a repeat spanning interval, a copy of that repeat must be 'reserved' in order to make sure this adjacency is compatible. In the first figure, such a situation is shown for oriented repeats. In this case, the adjacencies $\{v, x_0\}$ and $\{\overline{v}, x_1\}$ can be removed after decreasing the multiplicity of $v$ and $\overline{v}$ by 1. In the second figure, $\{v, x_0\}, \{v, x_1\}$ can both be removed after reducing the multiplicity of $v$ by 1.

spanning intervals. Since $\{v, x\}$ is not compatible with any repeat spanning interval in $E_I$, an occurrence of $v$ must be dedicated to making sure that $\{v, x\}$ is compatible with a realization. This occurrence cannot form part of a repeat spanning interval, since otherwise the vertex $x$ would have to be in the interval. Unlike the oriented case, however, it is possible that this occurrence also forms part of another adjacency. This is because $v$ is unoriented, and is not constrained to be adjacent to its mate in the realization, as shown in Figure 5.1b. This vertex too cannot be part of a repeat spanning interval containing $v$. Thus, the occurrence is shared across two adjacencies, and we need to 'reserve' only 1/2 copies of $v$ per adjacent vertex in $X$. This lets us deduce that at most $\mu(v) - \lceil |X|/2 \rceil$ copies of $v$ can occur in repeat spanning intervals.

□

A consequence of this result is that we can modify the instance $(H, \mu, w)$ as follows.

1. For every repeat $v$, find the set of vertices $X$ that it is adjacent to, and which do not occur with $v$ in repeat spanning intervals.

2. For every oriented repeat, reduce the multiplicity of $v$ to $\mu(v) - |X|$, and delete all edges from $v$ to $X$.

3. For every unoriented repeat $v$, reduce the multiplicity of $v$ to $\mu(v) - \lceil |X|/2 \rceil$, and delete all edges from $v$ to $X$.

In the rest of the chapter, we will always assume that the instance has been modified as stated above. That is, no instance to the problem has a repeat $v$ which is adjacent to a non-repeat which is not contained in a repeat spanning interval or to another repeat apart from, possibly, $\overline{v}$.

### 5.2.2 Bounding the number of repeat spanning intervals through repeats

Assume we are given a set $S \subseteq E_I$ such that the instance $(H', \mu, w')$, with $H' = (V, E_A \cup S)$, is realizable in the mixed genome model. The next lemma shows that the structure of this set $S$ is controlled by the fact that the underlying adjacency instance is realizable, and further restricts the number of repeat spanning intervals involving a vertex $v \in V$. The naive bound for this is $\mu(v)$ in general, and actually 2 for unoriented non-repeat. But Theorem 5.1 assumes that there are no unoriented non-repeat framing repeat spanning intervals. This helps us get tighter bounds.

**Lemma 5.2.** *Let $(H, \mu, w)$ $H = (V, E)$ be an instance such that $(H_A, \mu, w_A)$ is realizable in the mixed genome model, and $E_I$ is a set of repeat spanning intervals with properties as stated in Theorem 5.1. Then, given $S \subseteq E_I$, the instance $(H', \mu, w')$, where $H' = (V, E_A \cup S)$ is realizable in the mixed genome model if and only if $S$ has the following structure.*

1. *A non-repeat $u \in V$ appears in at most 1 repeat spanning interval in $S$,*

2. *An oriented repeat $v \in V_R$ appears in at most $\min\{deg_{H_A}(v) - 1,\ deg_{H_A}(\overline{v}) - 1\}$ different repeat spanning intervals in $S$.*

3. *A non-oriented repeat $v \in V_R$ appears in at most $\mu(v)$ different repeat spanning intervals in $S$.*

*Proof.* Let us first determine the structure of $S$ by assuming that $(H', \mu, w')$ is realizable in the mixed genome model. We first prove the following claim, which is the first part of the lemma.

**Claim.** *There is at most 1 repeat spanning interval in $S$ containing a non-repeat $u$.*

This is a direct consequence of the fact that non-repeats involved in a repeat spanning interval are oriented. Consider one such vertex $u \in V$ which is contained in some repeat spanning interval. Being oriented, $u$ must also be adjacent to $\overline{u}$ in a realization of $(H_A, \mu, w_A)$. Since it is a non-repeat, $u$ can only be adjacent to 1 other vertex apart from $\overline{u}$. Let such a vertex exist, and be called $v$. Then, $u$ can frame at most one repeat spanning interval, which must contain $v$.

The other two results in the lemma are consequences of this bound on the number of repeat spanning intervals using a non-repeat, and the fact that the repeat spanning intervals are minimal.

**Claim.** *An oriented repeat $v \in V_R$, can appear in at most* $\min\left\{deg_{H_A}(v) - 1, \deg_{H_A}(\overline{v}) - 1\right\}$ *different repeat spanning intervals in $S$.*

The instance $(H_A, \mu, w_A)$ is realizable in the mixed genome model. This means that both $v$ and $\overline{v}$ can only be adjacent to at most $\mu(v)$ vertices in $H_A$, apart from each other.

In the set $S$, any repeat spanning interval containing $v$ must also contain $\overline{v}$. From the previous claim, the non-repeat neighbours of $v$ and $\overline{v}$ can each be contained in exactly 1 repeat spanning interval each. Since a repeat spanning interval containing $v$ and $\overline{v}$ must be framed by 1 neighbour of $v$ and one of $\overline{v}$, the number of repeat spanning intervals in $S$ containing $v$ must be limited by $\min\{deg_{H_A}(v) - 1, deg_{H_A}(\overline{v}) - 1\}$, where the $-1$ accounts for the edge $\{v, \overline{v}\}$. Let this quantity be called $\tau_v$.

**Claim.** *An unoriented repeat $v \in V_R$ can appear in at most $\mu(v)$ different repeat spanning intervals in $S$.*

Using the fact that $(H_A, m, w_A)$ is realizable in the mixed genome model, we can conclude that a non-oriented repeat $v$ can be adjacent to at most $2\mu(v)$ vertices. Since each non-repeat can be contained in at most 1 repeat spanning interval, we can have at most $\mu(v)$ repeat spanning intervals which contain $v$, by pairing non-repeats into framing vertices of repeat spanning intervals. Note that this is the only case where the worst-case bound of $\mu(v)$ repeat spanning intervals containing $v$ is not improved.

This concludes the forward direction of the lemma. In order to prove the opposite direction, we will explicitly construct a realization of $(H', \mu, w')$. To do this, we will use a variant of the construction specified in Lemma 4.1. Initialize the instance $(H'', \mu', w'')$, with $H'' = (V, E'')$ to be the same as the instance $(H_A, \mu, w_A)$. For every repeat spanning interval $e \in S$ which spans an oriented repeat, such that $o(e) = u.r.\overline{r}.v$, where $u, v$ are the non-repeats framing $r$ and $\overline{r}$, which are oriented repeats, perform the following operations.

1. Add the edge $\{u, v\}_e$, labelled by $e$, to $E''$.
2. Decrement $\mu'(r)$ and $\mu'(\overline{r})$ by 1.
3. Delete the edges $\{u, r\}$ and $\{v, \overline{r}\}$.

For every repeat spanning interval $e \in S$ which spans an unoriented repeat, such that $o(e) = u.r.v$, where $u, v$ are non-repeats framing the repeat $r$, perform the following operations.

1. Add the edge $\{u, v\}_e$, labelled by $e$, to $E''$.
2. Decrement $\mu'(r)$ by 1.
3. Delete the edges $\{u, r\}$ and $\{v, r\}$.

The output is the instance $(H'', \mu', w'')$, which has no repeat spanning intervals. Note that the weight function $w''$ does not play any role in the following proof, and so we can set it to map all edges in $E''$ to 1.

We make the following claim.

**Claim.** *The instance $(H', \mu, w')$ is realizable in the mixed genome model if and only if the instance $(H'', \mu', w'')$ is realizable in the mixed genome model.*

Let us first prove that $(H'', \mu', w'')$ is realizable in the mixed genome model. We examine the local structure around each vertex in this instance, and conclude that this structure is sufficient to admit a realization.

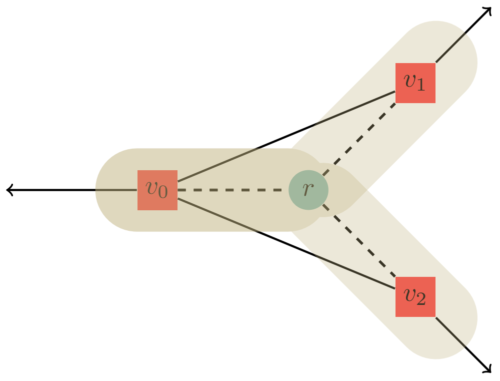
1. Every non-repeat $u \in V$ was contained in at most 1 repeat spanning interval in $S$. The degree of $u$ in $H''$ remains either unchanged, or it is incremented and decremented by 1 while considering the repeat spanning interval that $u$ is contained in. So, the degree of $u$ in $H''$ is still at most 2.

2. Assume that an oriented repeat $v \in V_R$ is contained in $\lambda_v \leq \tau_v \leq \mu(v)$ repeat spanning intervals in $S$, where $\tau_v = \min\{deg_{H_A}(v) - 1, deg_{H_A}(\overline{v}) - 1\}$. While constructing $(H'', \mu', w'')$, each of the $\lambda_v$ repeat spanning intervals was deleted, and the multiplicity $\mu'(v)$ was decremented to $\mu(v) - \lambda_v$. The degree of $v$ decreases to $deg_{H_A}(v) - \lambda_v \leq \mu(v) + 1 - \lambda_v$, since we delete all adjacencies to non-repeats that are in the same repeat spanning interval as $v$, and no such adjacency can be compatible with 2 repeat spanning intervals in $S$.

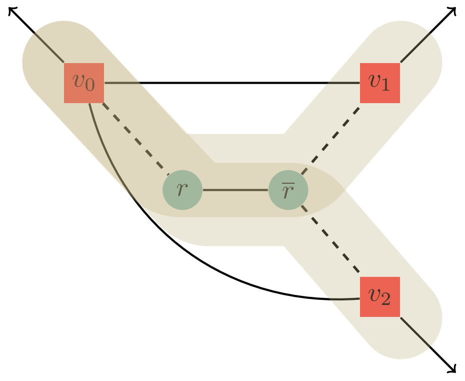
3. Assume that an unoriented repeat $v \in V_R$ is contained in $\lambda_v \leq \mu(v)$ repeat spanning intervals. As in the oriented case, the multiplicity $\mu'(v)$ is reduced to $\mu(v) - \lambda_v$. Since no non-repeat adjacent to $v$ occurs in 2 repeat spanning interval in $S$, the degree of $v$ in $H''$ is decreased to at most $deg_{H_A}(v) - 2\lambda_v \leq 2\mu(v) - 2\lambda_v$, where the extra factor of 2 accounts for the fact that $v$ is unoriented.

The degree of every non-repeat $v \in V$ is at most 2 in $H''$. The degree of every oriented repeat in $H''$, excluding its mate, is at most its multiplicity, while that of an unoriented repeat is at most twice its multiplicity. This means that in a $b$–matching, with the function $b$ being defined by $\mu'$, every edge is used at least once, and thus, the instance $(H'', \mu', w'')$ is realizable in the mixed genome model.

Now consider a realization $M'$ of the instance. In this realization, we have consecutive substrings of size 2 corresponding to a set of edges $\{u, v\}_e$, labelled by a repeat spanning interval $e \in S$. Replace the occurrence $u.v$ corresponding to such an edge with the string $o(e)$. Let the modified realization be called $M$. Since $o(e)$ occurs as a consecutive substring in $M$ for every $e$, each repeat spanning interval in $S$ is compatible with $M$.



(a) Unoriented repeat



(b) Oriented repeat

Figure 5.2: The constructions made during the algorithm for Theorem 5.1. In each case, for both oriented and unoriented repeats in the repeat spanning interval, the interval is replaced by an edge between its framing vertices, and any edges compatible with it are deleted. The multiplicity of the repeat/repeats in the interval is decreased by 1. Note that the framing vertices, which are non-repeats, are always oriented.

No non-repeats are introduced in $M$ during the construction, and they all appeared exactly once in $M'$, so their multiplicity is not violated. For a repeat $v \in V_R$, $M'$ originally contained $\mu'(v)$ occurrences of $v$. In $M$, we have added $\lambda_v$ occurrences of $v$, so we get a total of $\mu'(v) + \lambda_v$ occurrences. For both oriented and unoriented repeats, this value is at most $\mu(v)$, which means that $M$ is consistent with $(V, \mu)$.

Furthermore, all edges in $E_A$ are compatible with $M$. An edge $e \in E_A$ was deleted during the construction of $H''$ only if there was a repeat spanning interval $e' \in S$ with which it was compatible. Since each such interval is compatible with $M$, the deleted edges must also be compatible with $M$. The rest of the edges were already compatible with $M'$. This means that $M$ is a realization of $(H', \mu, w')$, which completes the proof. □

### 5.2.3 The algorithm

We can now describe the algorithm we use. The input is the instance $(H, \mu, w)$, with $H = (V, E_A \cup E_I)$, such that the underlying adjacency instance is realizable in the mixed genome model, and which has been suitably preprocessed using Lemma 5.1. The goal is to find a set $S \subseteq E_I$ of repeat spanning intervals of maximum weight which satisfies the conditions of Lemma 5.2.

1. Define an instance $(G', \mu', w'')$, $= (V, E')$, which only has adjacencies. Initialize $E = E_A$, $\mu' = \mu$. Also initialize $D = \{\emptyset\}$.

2. For every repeat spanning interval $e \in E_I$ spanning over an oriented repeat, carry out the following operations.

   (a) If $o(e) = u.r.\overline{r}.v$, where $u, v$ are non-repeats framing the repeat spanning interval, add the adjacency $\{u, v\}$ to $E'$, and associate it to the repeat spanning interval $e$.

   (b) Decrement $\mu'(r)$ and $\mu'(\overline{r})$ by 1.

   (c) Assign $w''(\{u, v\}) = w(e)$.

   (d) Add the edges $\{u, r\}$ and $\{\overline{r}, v\}$ to $D$.

3. For every repeat spanning intervals $e \in E_I$ spanning over an unoriented repeat, perform the following operations.

   (a) If $o(e) = u.r.v$, where $u, v$ are non-repeats framing the repeat spanning interval, add the adjacency $\{u, v\}$ to $E'$, and associate it to the repeat spanning interval $e$.

   (b) Decrement $\mu'(r)$ by 1.

   (c) Assign $w''(\{u, v\}) = w(e)$.

   (d) Add the edges $\{u, r\}$ and $\{r, v\}$ to $D$.

4. Delete the edges in $D$ from $E'$, and set all unassigned weights to $1 + \sum_{e \in E_I} w(e)$.

5. Using Theorem 3.7, optimize the instance $(G', \mu', w'')$ to obtain an instance $(G, \mu', w_Q)$, where $G = (V, Q)$ which is realizable in the mixed genome model, where $w_Q$ is the restriction of $w''$ to the edge set $Q$.

6. Output all repeat spanning intervals labelling edges in $Q$ as the desired set $S$.

Figure 5.2 shows the constructions made in the course of the algorithm for both the oriented and unoriented repeat cases. The detailed pseudocode is included in Appendix A for reference.

### 5.2.4 Analysis

#### Algorithm correctness

To check that the output of the algorithm, which is a set $S$ of repeat spanning intervals, is indeed the set of repeat spanning intervals that we desire, we note that the output set satisfies all the conditions of Lemma 5.2. Every non-repeat is contained in at most 1 repeat spanning interval in the set $S$. Otherwise, since each interval in $S$ corresponds to an edge in the set $Q$, there would have been a non-repeat adjacent to 2 edges in $Q$. But since the non-repeats framing repeat spanning intervals are oriented, this contradicts the fact that $(G, \mu', w_Q)$ is realizable in the mixed genome model.

This restriction on the non-repeats also means that an oriented repeat $v \in V_R$ appears in at most $\tau_v = \min\{deg_{H_A}(v) - 1, deg_{H_A}(\overline{v}) - 1\}$ repeat spanning intervals in $S$. Otherwise, the non-repeat neighbours of $v$ or $\overline{v}$ would be contained in more than 1 repeat spanning interval in $S$. Similarly, we can infer that an unoriented repeat $v \in V_R$ cannot appear in more than $\mu(v)$ intervals in $S$ This proves that the set $S$ is indeed compatible with a realization of $(H_A, \mu, w_A)$, and so, using Lemma 5.2, we conclude that the instance $(H', \mu, w')$ is realizable in the mixed genome model.

The set $S$ is also the maximum weight set of repeat spanning intervals having the desired property. Note that the maximum matching routine does not delete any edges in $e \in E_A \backslash D$, since they had weight $1+\sum_{e\in E_I} w(e)$. It would have been suboptimal to discard any of them when weighed against discarding the set of edges that were added during the construction of $G'$. The algorithm only deletes edges that were added in place of repeat spanning intervals. Since the maximum matching routine makes sure that $Q$ is the maximum weight set of edges that can be chosen, and each of these edges corresponds to a repeat spanning interval, the set $S$ which satisfies the conditions of Lemma 5.2.

#### Algorithm complexity

The instance $(G', \mu', w'')$ can be constructed from the original instance in time linear in $|E_I|$, i.e. the number of repeat spanning intervals. At the end, too, we can recover the set $S$ from the set $Q$ of retained labelled adjacencies in linear time. The computational bottleneck of the algorithm is in the maximum matching algorithm. There are 2 steps in this algorithm. The first consists of constructing an auxiliary graph for $b$–matching, as described by Dessmark et al. [58], and Maňuch et al. [138]. The number of vertices in $G'$ is $n$, and the number of edges is $O(m)$, where $n$ and $m$ are the number of vertices and hyperedges (adjacencies and intervals) in the input instance. Let $\bar{\mu}$ be the maximum multiplicity of a vertex. In the construction of the auxiliary graph, we introduce at most $2\bar{\mu}n + 2m$ vertices, and at most $(2\bar{\mu} + 1)m$ edges. In general, $\bar{\mu} << n, m$, and the multiplicity is usually taken to be a constant and hidden in the asymptotic notation. The maximum matching algorithm runs in time $O\left(|E|\sqrt{|V|}\right)$ for a graph with $|V|$ vertices and $|E|$ edges [146]. So, the runtime for the maximum matching routine in this instance is $O\left(m\sqrt{\bar{\mu}n+m}\right)$.

The space complexity for the algorithm is linear in the size of the input instance. In this case, this amounts to $O\left(\bar{\mu}n+m\right)$, so it remains linear in the size of the input instance, assuming $\bar{\mu} << n, m$.

### 5.2.5 The role of oriented vertices

Theorem 5.1 crucially relied on the fact that the frontier vertices of a repeat cluster were all oriented. One may ask if Problem 5.1 is solvable in polynomial time if all the vertices are oriented, and the problem is hard otherwise. The next chapter shows that this is not true. The orientation of the vertices allows us to use restrict the number of repeat spanning intervals through a repeat, but if the interval is not minimal, this is not a sufficient condition, as there may be other types of conflicts.

The result in this chapter also fails when we have to consider unoriented framing vertices in the repeat spanning intervals. It is useful to know if there is a condition on the repeat spanning intervals under which we can optimize over them even if they are framed by unoriented vertices.

# Chapter 6

# Optimization on repeat spanning intervals is hard

We introduced Problem 5.1 in the previous chapter, and showed in Theorem 5.1 that at least when the repeat spanning intervals involved are minimal, and when the frontier vertices are oriented, the problem is tractable. This is indeed a useful first step, as we discussed, but it definitely limits the scope of the problem. So one would wonder: can Theorem 5.1 be generalized to non-minimal repeat spanning intervals? In this chapter we rule out this possibility.

The reduction in this section was joint work with Cedric Chauve, Ján Maňuch and João Zanetti. The dynamic programming algorithm presented at the end was joint work with Cedric Chauve and João Zanetti.

## 6.1 Hardness of optimization

**Theorem 6.1.** *Problem 5.1 is NP-hard for the mixed genome model even when the maximum multiplicity of a repeat is* 2.

The proof given below assumes that all vertices in the instance are oriented, and so, they can be paired into heads and tails. This automatically enforces the condition that the frontier vertices to a repeat cluster are also oriented. For the purposes of the construction, we shall use the term *oriented pair* to refer to two such vertices, and this relation is denoted by $(v, \overline{v})$ (the order is unimportant). To *add* an oriented pair $(v, \overline{v})$ to an instance $(H, \mu, w)$, $H = (V, E)$, implies that we are adding the vertices $v, \overline{v}$ to $V$, and the adjacency $\{v, \overline{v}\}$ to $E$. Later, we give the corresponding gadgets when the vertices are not oriented, and the proof for that case follows the same basic principle.

### 6.1.1 3SAT(2,2)

In order to prove Theorem 6.1, we will provide a reduction from the following problem, which we call 3SAT(2,2). The details of the notation used here, and the hardness of this problem itself is proved through a reduction from classical 3SAT. This reduction is provided in Appendix B.

**Lemma 6.1.** *Let $\Phi$ be a boolean formula in CNF form, such that there are exactly $n$ variables, $m$ 3-clauses, and every variable has precisely 2 positive and 2 negative literals each. Then, the problem of deciding if $\Phi$ has a satisfying assignment is NP-complete.*

The choice of this problem is not arbitrary: the intractability result in Theorem 3.7 was proved through a reduction from (2,3)SAT(1,2), a related problem in which the CNF formula $\Phi$ has precisely 2 positive and 1 negative occurrence of each variable, and the negative occurrence and exactly one of the positive occurrences happen in 2-clauses. Similarly, the intractability result in Theorem 3.2 was also proved by reduction from another 3SAT variant. Let us assume that the instance of 3SAT(2,2) given is called $\Phi = (\mathcal{X}, \mathcal{C})$, where $\mathcal{X} = \left\{x_0, \ldots, x_{|\mathcal{X}|-1}\right\}$ is a set of boolean variables, each having 2 positive and 2 negative occurrences in $\Phi$, and $\mathcal{C} = \left\{c_0, \ldots, c_{|\mathcal{C}|-1}\right\}$ is a set of 3-clauses. Clearly, $4|\mathcal{X}| = 3|\mathcal{C}|$ by the constraints on the clause size and the number of variable occurrences.

### 6.1.2 Constructing the underlying adjacency instance

To define the reduction, we first need to define the underlying adjacency instance. Note that this instance needs to be realizable in the mixed genome model. We follow this up by defining a set of repeat spanning intervals, such that not all of them can be in an optimal set $S$. We call this instance $(H, \mu, w)$, where $H = (V, E)$, $w$ maps all hyperedges in $E$ to 1. Initialize $V = E = \emptyset$ and then begin the construction.

#### Variable gadget

For each variable $x_i \in \mathcal{X}$, $i \in [|\mathcal{X}|]$, we shall introduce a vertex gadget, constructed as follows.

1. Add an oriented pair $\left(X_i, \overline{X}_i\right)$ to the instance.
2. Add oriented pairs $\left(U_i, \overline{U}_i\right)$ and $\left(W_i, \overline{W}_i\right)$ to the instance. Add edges $\{U_i, X_i\}$ and $\{W_i, X_i\}$ to $E$.
3. Add oriented pairs $\left(P_i, \overline{P}_i\right)$ and $\left(Q_i, \overline{Q}_i\right)$ to the instance. Add edges $\left\{P_i, \overline{X}_i\right\}$ and $\left\{Q_i, \overline{X}_i\right\}$ to $E$.
4. Add oriented pairs $\left(P_i^0, \overline{P}_i^0\right)$ and $\left(P_i^1, \overline{P}_i^1\right)$, and edges $\left\{P_i^0, \overline{P}_i\right\}$ and $\left\{P_i^1, \overline{P}_i\right\}$ to $E$.

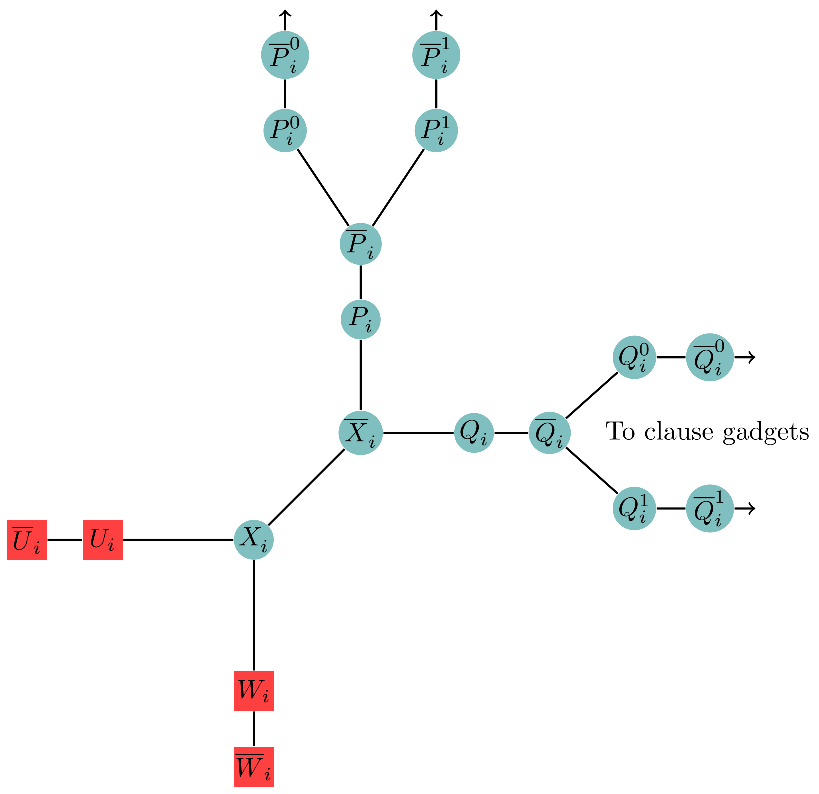


Figure 6.1: Variable gadget used in the reduction, for a variable $x_i$. Square vertices are non-repeats, while circular vertices are repeats.

5. Similarly, add oriented pairs $\left(Q_i^0, \overline{Q}_i^0\right)$ and $\left(Q_i^1, \overline{Q}_i^1\right)$, and edges $\left\{Q_i^0, \overline{Q}_i\right\}$ and $\left\{Q_i^1, \overline{Q}_i\right\}$ to $E$.

The gadget for $x_i \in \mathcal{X}$ is shown in Figure 6.1.

### Clause gadget

Consider a clause $c_p = (l_{p,0} \vee l_{p,1} \vee l_{p,2})$ in $\mathcal{C}$, where all the $l_{p,i}$ are are literals, possibly with $l_{p,i} = l_{p,j}$ for $i \neq j$. The next part of the instance construction consists of including the clause restrictions in $\Phi$ into the instance. This construction will take place in 3 stages. The first stage consists of making constructions for each literal $l_{p,i}$ in the clause.

1. Add an oriented pair $\left(L_{p,i}, \overline{L}_{p,i}\right)$ to the instance.
2. Add oriented pairs $\left(L_{p,i}^{i+1}, \overline{L}_{p,i}^{i+1}\right)$ and $\left(L_{p,i}^{i-1}, \overline{L}_{p,i}^{i-1}\right)$, where the additions in the indices is done modulo 3. Add the edge $\left\{\overline{L}_{p,i}^{i+1}, \overline{L}_{p,i}^{i-1}\right\}$ to $E$.
3. Add edges $\left\{L_{p,i}^{i+1}, \overline{L}_{p,i}\right\}$ and $\left\{L_{p,i}^{i-1}, \overline{L}_{p,i}\right\}$ to $E$.

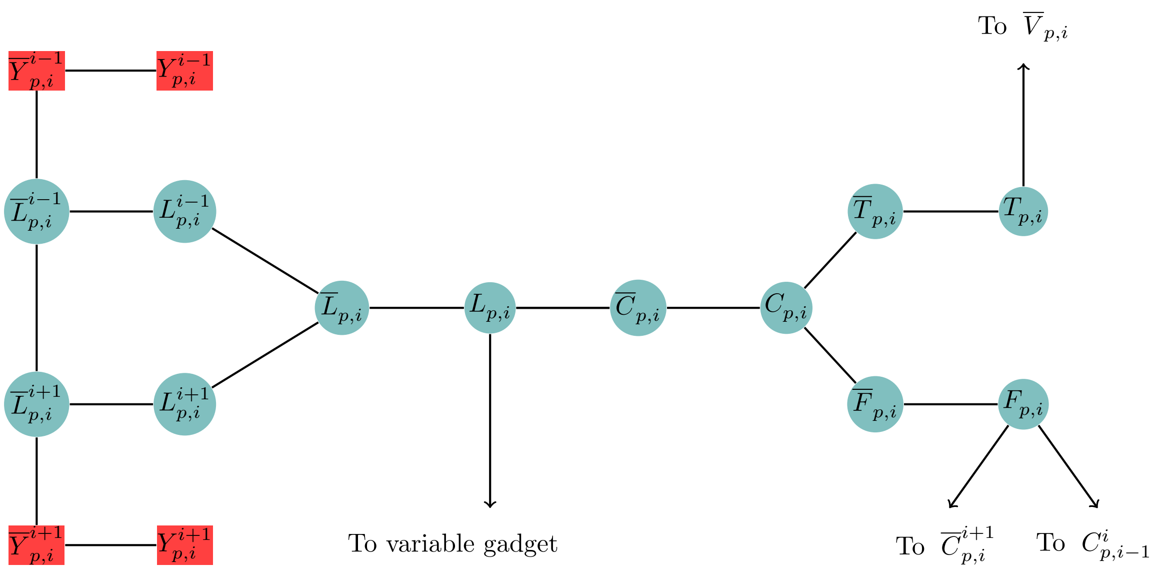


Figure 6.2: Part of the clause gadget constructed for a single literal $l_{p,i}$ of the clause $c_p = (l_{p,0} \vee l_{p,1} \vee l_{p,2})$.

4. Add an oriented pair $\left(Y^{i+1}_{p,i}, \overline{Y}^{i+1}_{p,i}\right)$, and an edge $\left\{\overline{Y}^{i+1}_{p,i}, \overline{L}^{i+1}_{p,i}\right\}$, additions being done modulo 3.
5. Similarly, add an oriented pair $\left(Y^{i-1}_{p,i}, \overline{Y}^{i-1}_{p,i}\right)$, and an edge $\left\{\overline{Y}^{i-1}_{p,i}, \overline{L}^{i-1}_{p,i}\right\}$.
6. Add an oriented pair $\left(C_{p,i}, \overline{C}_{p,i}\right)$, and add an edge $\left\{L_{p,i}, \overline{C}_{p,i}\right\}$ to $E$.
7. Add oriented pairs $\left(T_{p,i}, \overline{T}_{p,i}\right)$ and $\left(F_{p,i}, \overline{F}_{p,i}\right)$. Add edges $\left\{\overline{T}_{p,i}, C_{p,i}\right\}$ and $\left\{\overline{F}_{p,i}, C_{p,i}\right\}$ to $E$.

The construction of this part is given in Figure 6.2. Once this construction is completed for all three literals in $c_p$, we construct two cycles as follows. The first cycle will be involved in setting literals to 0, and we call it the *false cycle.*

1. Add oriented pairs $\left(C^{i+1}_{p,i}, \overline{C}^{i+1}_{p,i}\right)$ for all $i \in [3]$.
2. Add edges $\left\{C^{i}_{p,i-1}, F_{p,i}\right\}$, additions being done modulo 3.
3. Add edges $\left\{\overline{C}^{i+1}_{p,i}, F_{p,i}\right\}$.

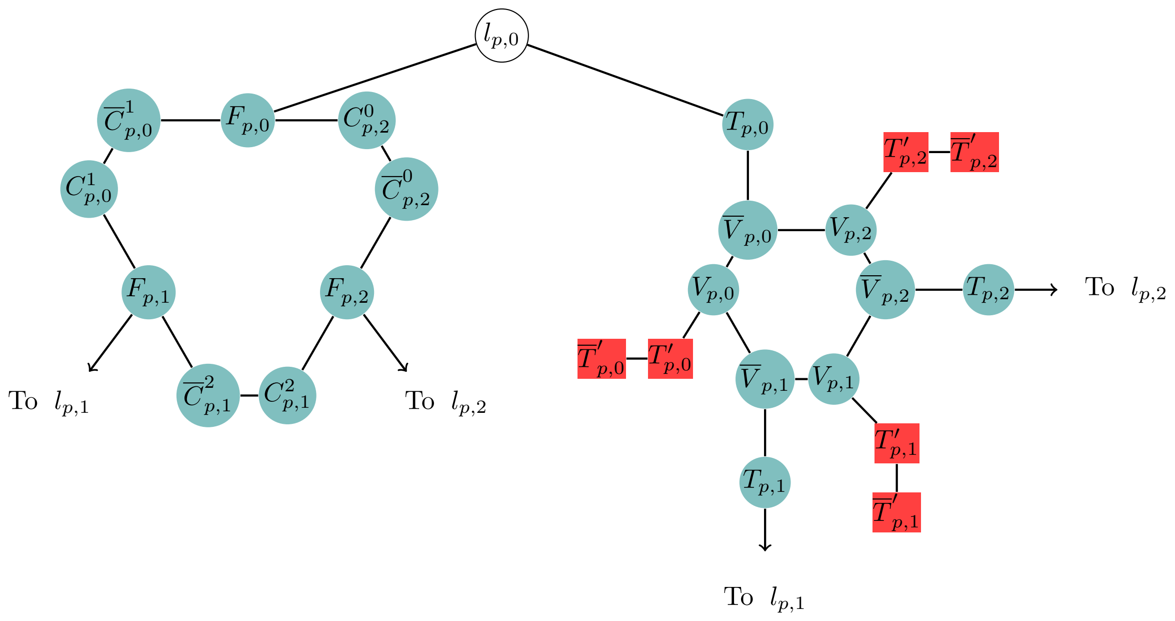


Figure 6.3: Cycles in a clause gadget connecting the components corresponding to different literals. Here, $l_{p,0}, l_{p,1}$ and $l_{p,2}$ are the constructions made in Figure 6.2 for every literal in the clause. The vertices $F_{p,i}$ and $T_{p,i}$ are identified with the corresponding vertices of the same name in Figure 6.2 for every literal $l_{p,i}$.

The second cycle, conversely, is involved in setting literals to 1, and so we call it the *true cycle*.

1. Add oriented pairs $\left(V_{p,i}, \overline{V}_{p,i}\right)$ for all $i \in [3]$.
2. Add edges $\left\{V_{p,i}, \overline{V}_{p,i+1}\right\}$ to $E$ for each $i$.
3. Add edges $\left\{T_{p,i}, \overline{V}_{p,i}\right\}$ to $E$ for each $i$.
4. Add oriented pairs $\left(T'_{p,i}, \overline{T}'_{p,i}\right)$ for $i \in [3]$.
5. Add edges $\left\{T'_{p,i}, V_{p,i}\right\}$ for all $i$.

The construction of the clause gadget, in total, is given in Figure 6.3.

Finally, the construction is completed by linking the variable gadgets to the clause gadgets. Consider a clause $c_p = (l_{p,0} \vee l_{p,1} \vee l_{p,2})$. Assume $l_{p,0} = x_i^\delta$, where $\delta \in \{0, 1\}$, depending on whether it is a positive ($\delta = 1$) or a negative literal ($\delta = 0$). If, during the construction, the vertex $\overline{P}_i^\delta$ has degree 1, add an edge $\left\{\overline{P}_i^\delta, L_{p,i}\right\}$. Otherwise, add an edge $\left\{\overline{Q}_i^\delta, L_{p,i}\right\}$. Since each $x_i$ has exactly 2 occurrences as a positive literal $x_i^1$, and two occurrences as a negative literal $x_i^0$, at least one of $\overline{P}_i^\delta$ and $\overline{Q}_i^\delta$ must have degree 1 before adding this edge.

The variable gadget for a single variable $x_i \in \mathcal{X}$ consists of 18 vertices and 17 edges. The clause gadget, when put together with all three literals and the involved cycles, consists of 60 vertices and 75 edges, including those which connect it to the variable gadgets. So, the total size of the vertex set of the instance $(H, \mu, w)$ is $n = 18|\mathcal{X}| + 60|\mathcal{C}|$, and the cardinality of the set of adjacencies $E_A \subseteq E$ is $m = 17|\mathcal{X}| + 75|\mathcal{C}|$. Clearly, this is polynomial in the size of the instance $\Phi$.

**Multiplicities and weights**

The multiplicity function $\mu$ is defined for the instance as follows.

1. In the variable gadgets, $\mu(U_i) = \mu(W_i) = 1$ for all $i \in [|\mathcal{X}|]$. By the definition of oriented vertices, we also get $\mu\left(\overline{U}_i\right) = \mu\left(\overline{W}_i\right) = 1$.
2. In the clause gadget, $\mu\left(T'_{p,i}\right) = \mu\left(Y^{i+1}_{p,i}\right) = \mu\left(Y^{i-1}_{p,i}\right) = 1$. As in the previous case, the respective mates in the oriented pairs involving these vertices also have multiplicity 1.
3. For all other vertices, $\mu$ evaluates to 2.

Note that there are no vertices of higher multiplicity than 2. For the weights, we define $w_A$, the restriction of $w$ to $E_A$, to be 1 for all $e \in E_A$. We now have to verify that adjacency instance satisfies the hypothesis under which we operate.

**Claim.** *The instance $(H_A, \mu, w_A)$ is realizable in the mixed genome model.*

To note this, consider the following construction. For every oriented pair $(v, \overline{v})$ in the adjacency instance $(H_A, \mu, w_A)$, $H_A = (V, E_A)$, collapse the edge $\{v, \overline{v}\}$ to get a single vertex adjacent to all vertices that each of $v$ and $\overline{v}$ were adjacent to. Add an unoriented vertex $V_0$ of unbounded multiplicity, and add edges (with multiple edges allowed) from $V_0$ to every vertex $v \in V$ which has degree less than $2\mu(v)$, till this limit is reached. Once this construction is complete, we get an instance in which each vertex has even degree. So, we can find an Eulerian tour in this graph. For every 3 vertex walk $u'.v'.w'$ in this tour, if $v'$ is the collapsed oriented pair $(v, \overline{v})$ such that $v$ was adjacent to a vertex in the oriented pair corresponding to $u'$, and $\overline{v}$ was adjacent to a vertex in the oriented pair corresponding to $w'$, expand this walk into $u'.v.\overline{v}.w'$. Doing so, we get a tour on the graph $H_A \cup \{V_0\}$. Deleting the vertex $V_0$, we now get a set of walks in which each vertex $v \in V$ appears at most $\mu(v)$ times, and every edge in $E_A$ is traversed at least once. This set of walks is the desired realization, and proves the claim.

Having verified this, we now proceed to define the repeat spanning intervals, which form the crux of the construction.

### 6.1.3 Constructing the set of repeat spanning intervals

There are 2 types of repeat spanning intervals that we will be adding to the instance. The first type, which we call *literal intervals*, are intervals that will be chosen when a literal is set to true. These intervals run from a variable gadget to a clause gadget. The second type of repeat spanning intervals are called *clause intervals*, and they run within a single clause gadget. They act to block us from setting a literal to true within a clause.

To specify a repeat spanning interval $e \in E$, we shall specify the order $o\left(e\right) \in V^*$ of the interval instead. In the proof, we make no distinction between the order of a repeat spanning interval and the interval itself. Since the order may be very long, we will use the following notation to condense their representation. For a clause $c_p = (l_{p,0} \vee l_{p,1} \vee l_{p,2})$, we define $\alpha_{p,i}$ for $i \in [3]$ as the sequence of vertices defined by the following unique walk in $H_A = (V, E_A)$.

$$\begin{aligned}\alpha_{p,i} =& L_{p,i}.\overline{L}_{p,i}.L_{p,i}^{i+1}.\overline{L}_{p,i}^{i+1}.\overline{L}_{p,i}^{i-1}.L_{p,i}^{i-1}.\ \overline{L}_{p,i}.L_{p,i}.\overline{C}_{p,i}.C_{p,i}.- \\ &- \overline{T}_{p,i}.T_{p,i}.\overline{V}_{p,i}.V_{p,i}.\overline{V}_{p,i+1}.V_{p,i+1}.\ \overline{V}_{p,i-1}.V_{p,i-1}.\overline{V}_{p,i}.V_{p,i}T'_{p,i},\end{aligned}$$

where the line break indicates a continuation of the walk.

#### Literal intervals

Consider a variable $x_k \in \mathcal{X}$, and assume that $\overline{P}_k^0$, $\overline{P}_k^1$, $\overline{Q}_k^0$ and $\overline{Q}_k^1$ are adjacent to $L_{p_0,i_0}$, $L_{p_1,i_1}$, $L_{p_2,i_2}$ and $L_{p_3,i_3}$ respectively. Define the set of literal intervals as the following set for all $k \in [|\mathcal{X}|]$.

$$\begin{aligned}&\overline{W_k}.\ X_k.\ \overline{X}_k.\ P_k.\ \overline{P}_k.\ P_k^0.\ \overline{P}_k^0.\ \alpha_{p_0,i_0},\\&\overline{U_k}.\ X_k.\ \overline{X}_k.\ P_k.\ \overline{P}_k.\ P_k^1.\ \overline{P}_k^1.\ \alpha_{p_1,i_1},\\&\overline{U_k}.\ X_k.\ \overline{X}_k.\ Q_k.\ \overline{Q}_k.\ Q_k^0.\ \overline{Q}_k^0.\ \alpha_{p_2,i_2},\\&\overline{W_k}.\ X_k.\ \overline{X_k}.\ Q_k.\ \overline{Q}_k.\ Q_k^1.\ \overline{Q}_k^1.\ \alpha_{p_3,i_3}.\end{aligned}$$

Note that no two intervals defined above pass through the same vertex $T_{p,i_j}$. So, we will use the notation $T_{p,i_j}$-interval to specify the repeat spanning interval passing through that vertex.

#### Clause intervals

The clause intervals for a clause $c_p = (l_{p,0} \vee l_{p,1} \vee l_{p,2})$ are defined for every pair of literals $\{l_{p,i}, l_{p,j}\}$, $i \neq j$, in the clause, as follows.

$$\overline{Y}^{i+1}_{p,i}.\overline{L}^{i+1}_{p,i}.L^{i+1}_{p,i}.\overline{L}_{p,i}.L_{p,i}.\ \overline{C}_{p,i}.C_{p,i}.\overline{F}_{p,i}.F_{p,i}.\overline{C}^{i+1}_{p,i}.C^{i+1}_{p,i}.-$$
$$-.F_{p,i+1}.\overline{F}_{p,i+1}.C_{p,i+1}.\overline{C}_{p,i+1}.L_{p,i+1}.\overline{L}_{p,i+1}.L^{i}_{p,i+1}.\overline{L}^{i}_{p,i+1}.\overline{Y}^{i}_{p,i+1},$$

where index addition is done modulo 3. We denote these repeat spanning intervals by the shorthand $\left[Y^{i+1}_{p,i}, Y^{i}_{p,i+1}\right]$ for all $p \in [|\mathcal{C}|], i \in [3]$.

In all, note that the set $E_I$ constructed consists of 4 literal repeat spanning intervals per variable, and 3 repeat spanning clause intervals per clause (by pairing up the 3 literals in it). So, the size of the set $E_I$ is $4|\mathcal{X}| + 3|\mathcal{C}| = 6|\mathcal{C}|$. Once the set of repeat spanning intervals $E_I$ is defined, we extend the weight function $w$ so that $w(e) = 1$ for all $e \in E_I$, i.e. all the repeat spanning intervals have the same weight.

### 6.1.4 The proof

We wish to prove the following claim.

**Claim** (Main claim)**.** *$\Phi$ has a satisfying assignment if and only if there is a subset $S \subseteq E_I$ of size at least $2|\mathcal{C}|$, such that $(H', \mu, w')$, where $H' = (V, E_A \cup S)$ is realizable in the mixed genome model.*

We prove the claim by demonstrating that, given a satisfying assignment for $\Phi$, we can construct a set $S \subseteq E_I$ of repeat spanning intervals having the given properties, and the converse. We prove the forward direction first.

**"⇒"** If $\Phi$ has a satisfying assignment, there exists a set $S \subseteq E_I$ of size at least $2|\mathcal{C}|$ such that $(H', \mu, w')$, $H' = (V, E_A \cup S)$ is realizable in the mixed genome model.

Using the given satisfying assignment, we are going to construct the set $S$, and show that the instance $(H', \mu, w')$ is realizable in the mixed genome model.

Assume that the satisfying assignment for $\Phi$ sets the literal $l_{p,0}$ of a literal $c_p = (l_{p,0} \vee l_{p,1} \vee l_{p,2})$ to 1. This can be assumed without loss of generality, since we can relabel the literals within a clause. Construct the set $S$ as follows. Initialize $S = \emptyset$.

1. For each clause $c_p \in \mathcal{C}$, add the $T_{p,0}$-interval to $S$.
2. For each clause $c_p \in \mathcal{C}$, add the $\left[Y^2_{p,1}, Y^1_{p,2}\right]$-interval to $S$.

The size of the set $S$ after this construction is exactly $2|\mathcal{C}|$, since we add precisely 2 repeat spanning intervals to it per clause. So, the required cardinality of the set, at least, is satisfied.

To show that the instance $(H', \mu, w')$ is realizable in the mixed genome model, we are going to use Lemma 4.1. Recall that this lemma states that an instance with repeat spanning intervals is realizable in a genome model if and only if an equivalent instance, in which repeat spanning intervals have been replaced by paths simulating the vertex order associated with the intervals, is realizable in the same model, without any vertices in this instance having negative multiplicity. We will construct this equivalent instance, and show that it is realizable in the mixed genome model.

Call the new instance $(G, \mu', w'')$, where $G = (V', E')$. Initialize $G = (V', E')$ to be the underlying adjacency graph $H_A = (V, E_A)$, and set $\mu' = \mu$ on this set of vertices. Let $S'$ be the set of repeat spanning intervals that have been iterated over. For every repeat spanning interval $e \in S$, with order $o(e) = u.r_0. \dots .r_{k-1}.v$, where it is possible that $r_i = r_j$ for $i \neq j$, perform the following operations.

1. Add $e$ to $S'$.
2. For every $r_i$ in the order, $i \in [k]$, add a vertex $r_{e,i}$ to $V'$, and set $\mu'(r_{e,i}) = 1$.
3. Decrement $\mu'(r_i)$ by 1 for all $i \in [k]$.
4. Delete edges $\{u, r_0\}$ and $\{r_{k-1}, v\}$.
5. Add edges $\{r_{e,i}, r_{e,i+1}\}$ for all $i \in [k-1]$ to $E'$.
6. Add edges $\{u, r_{e,0}\}$ and $\{r_{e,k-1}, v\}$ to $E'$.
7. If the degree of an oriented repeat $r_i$ exceeds $\mu'(r_i) + 1$, delete all adjacencies to it which are compatible with a repeat spanning interval in $S'$, except for the adjacency with its mate $\overline{r}_i$.

For the weight function $w''$, set $w''(e) = 1$ for all $e \in E'$. The instance $(G, \mu', w'')$ only consists of adjacencies, since the only intervals in the original instance were the repeat spanning intervals, and we replaced them by paths in the above construction. For this instance to be realizable in the mixed genome model, by Theorem 3.2, every vertex $v \in V$ must have degree at most $\mu'(v) + 1$.

First note that all non-repeats $u \in V'$ in the graph $G$ will have degree at most $\mu'(v)+1$. Every non-repeat $u$ was contained in at most 1 repeat spanning interval in $S$, by the construction of the set $S$. Otherwise, there would have been conflicting assignments of a variable in the satisfying assignment of $\Phi$. To see this, note that each positive literal of a variable $x_i$ is associated to a repeat spanning interval containing exactly one of the vertices $U_i$ and $W_i$, and so is each negative literal. Thus, the only way that $U_i$ occurs in 2 repeat spanning interval is if both $x_i^1$ and $x_i^0$ are set to 1, which is not possible. For the vertices $\overline{Y}_{p,i}^{j}$, only 1 repeat spanning interval containing them is chosen, if any (assuming $i, j \neq 0$). On replacing these intervals with a path, we deleted and added an edge to a non-repeat $u$. Since the degree of $u$ in $H_A$ was at most 2, its degree now is still the same. Also, the multiplicity $\mu'(v)$ does not change for non-repeats, so it is always positive, and the stated bound on the degree is satisfied.

For the repeats, if a repeat occurs once in a repeat spanning interval in $S$, its multiplicity is reduced by 1 and, if it has degree 3, an edge compatible with the interval it is contained in is deleted. So, its degree becomes 2. If it occurs twice, then 2 edges are removed. In either case, the degree is at most 2. To verify that this degree is bounded, we will show that the multiplicity of the vertices are accordingly decremented.

In every variable gadget, there can be at most 2 literal intervals chosen. This comes from the fact that either the corresponding variable is set to 0 or 1, and there are two literals of each type, of which 0, 1 or both of them can be leading literals in a clause (i.e. in a clause $c_p = (l_{p,0} \vee l_{p,1} \vee l_{p,2})$, they occur as $l_{p,0}$). By the construction, if a variable $x_i$ is set to $\delta \in \{0,1\}$, the repeat spanning intervals chosen (if any) will contain $P_i^\delta$ and $Q_i^\delta$, and these intervals only share the vertices $X_i$ and $\overline{X}_i$. If no intervals are chosen, the multiplicity of $X_i$ and $\overline{X}_i$ remains 2, and the degree remains 3. If exactly 1 interval is chosen, the multiplicity of $X_i$ and $\overline{X}_i$ is decremented by 1, and one of the 2 edges, $\{U_i, X_i\}$ or $\{W_i, X_i\}$ is deleted. Also, one of the edges $\left\{\overline{X}_i, P_i\right\}$ or $\left\{\overline{X}_i, Q_i\right\}$ is deleted, depending on which one is compatible with the chosen interval. So, the degree of $X_i$ and $\overline{X}_i$ becomes 2, including the edge $\left\{X_i, \overline{X}_i\right\}$. If both repeat spanning intervals are chosen, then all 4 edges stated before are deleted, and the multiplicity of both $X_i$ and $\overline{X}_i$ is decremented twice to 0. So, their degree is still at most $\mu'(X_i) + 1$.

For the rest of the vertices in the variable gadget, the only ones we need to worry about are $\overline{P}_i$ and $\overline{Q}_i$, which have degree 3. But each of these vertices can only be contained in at most 1 repeat spanning interval in $S$, and on deleting the edge adjacent to them which is compatible with the chosen repeat spanning intervals, their degree becomes 2, while the multiplicity is decremented to 1, which is still under the bound required. The rest of the vertices all have degree 2 and are contained in at most 1 repeat spanning interval.

In the clause gadget, first note that the two chosen repeat spanning intervals, which are walks within the graph, are vertex disjoint. No vertex is used more than 2 times in the literal interval chosen, and for a clause $c_p$ in which the $T_{p,0}$-interval is chosen, the only vertices that occur more than once in the order are $L_{p,i}$ and $\overline{L}_{p,i}$. Since their multiplicity is decremented to 0, on deleting all edges compatible with chosen repeat spanning intervals, their degree becomes 1, which is the edge between $L_{p,i}$ and $\overline{L}_{p,i}$. So, the bound is met. Following this same procedure, we can verify that on deleting compatible edges, which is all the edges that define the walk associated to the order of the repeat spanning interval, the degrees of the remaining vertices is bounded by 1 exceeding the multiplicity.

For the chosen clause interval, say the $\left[Y_{p,1}^2, Y_{p,2}^1\right]$-interval, no vertex occurs twice in the associated order. On decrementing the multiplicity of the vertices by 1, and deleting compatible edges, the remaining edges adjacent to a vertex $v$ in the interval are again at most $1 + \mu'(v)$. So, the bound is verified for all vertices.

Finally, since none of the vertices is assigned a negative multiplicity in this process, using Theorem 3.2, we can conclude that the instance $(G, \mu', w'')$ is realizable in the mixed genome model. Using Lemma 4.1, we can conclude that $(H', \mu, w')$, $H' = (V, E_A \cup S)$, is realizable in the mixed genome model as well.

The proof for the reverse direction seeks to establish that any compatible set of repeat spanning intervals $S$ such that $(H', \mu, w')$ is realizable in the mixed genome model must have the structure of the set that we constructed in the forward direction.

"$\Leftarrow$" If there exists a set $S \subseteq E_I$ of size at least $2|\mathcal{C}|$ such that $(H', \mu, w')$, $H' = (V, E_A \cup S)$ is realizable in the mixed genome model, then there exists a satisfying assignment for $\Phi$.

For the purposes of the proof, assume that $(G, \mu', w'')$, $G = (V', E')$, is the instance constructed by replacing repeat spanning intervals by paths, as in the previous case. By Lemma 4.1, this instance must also be realizable in the mixed genome model, since $(H', \mu, w')$ is realizable in it.

We claim that the set $S$, which we will restrict to have size exactly $2|\mathcal{C}|$, must contain exactly 1 literal interval per clause gadget. We prove this argument below.

1. If one chooses 2 clause intervals per clause gadget, say the $\left[Y_{p,1}^2, Y_{p,2}^1\right]$-interval and the $\left[Y_{p,2}^0, Y_{p,0}^2\right]$-interval for a clause $c_p$, the vertex $L_{p,2}$ occurs 2 times in total over the orders associated to these intervals. So, during the construction of the graph $(G, \mu', w'')$ using the set $S$ as a template, the multiplicity of $L_{p,2}$ will be set to 0. But $L_{p,2}$ is also adjacent to a vertex from a variable gadget, say $P_i^\delta$ (which means that the third literal in the clause is $x_i^\delta$). This edge cannot be compatible with any realization of the instance, since $L_{p,2}$ cannot occur a third time to satisfy the demand imposed by the compatibility constraint. So, at most 1 clause interval can be chosen per clause gadget.

2. If a clause gadget contains 2 literal intervals, say the $T_{p,0}$-interval and the $T_{p,1}$-interval for a clause $c_p$, then each of these repeat spanning intervals uses the vertices $V_{p,2}$ and $\overline{V}_{p,2}$ once each, setting their multiplicity to 0. Since $V_{p,2}$ is adjacent to $T'_{p,2}$, and $\mu'(V_{p,i}) = 0$ by construction, this contradicts the realizability of $(G, \mu', w'')$, and by extension, that of $(H', \mu, w')$. So, at most 1 literal interval can be in $S$ for each clause gadget.

Since the set $S$ consists of exactly $2|\mathcal{C}|$ intervals, this means that there are exactly $|\mathcal{C}|$ literal intervals, and the rest are all clause intervals.

Now we prove that there are at most 2 literal intervals per variable gadget, and that if one contains $P_i^\delta$, the other must contain $Q_i^\delta$ and vice-versa, where $\delta \in \{0, 1\}$. Assume we choose the literal intervals which contain $P_i^\delta$ and $Q_i^{1-\delta}$. Then both these repeat spanning intervals contain the non-repeat $W_i$. This means $W_i$ occurs twice in a realization of $(H', \mu, w')$, but since $\mu(W_i) = 1$, it contradicts the realizability of this instance. Similarly, if the literal intervals chosen contain both $P_i^\delta$ and $P_i^{1-\delta}$, the vertex $\overline{X}_i$ is used once in each of the chosen intervals, which means the edge $\left\{Q_i, \overline{X}_i\right\}$ cannot be compatible with a genome map, which again contradicts the realizability. So, these cases cannot occur, and proves that if one of the chosen literal intervals contains $P_i^\delta$, the other interval, if chosen, must contain $Q_i^\delta$ and vice-versa, where $\delta \in \{0, 1\}$.

Finally, we construct a satisfying assignment for $\Phi$ as follows. For each clause gadget, consider the corresponding literal interval contained in $S$. If this interval contains either $P_i^0$ or $Q_i^0$ for some $i \in [\mathcal{X}]$, set the variable $x_i \in \mathcal{X}$ to 0. Otherwise, if it contains $P_i^1$ or $Q_i^1$, set the variable to 1. Since we cannot pick 2 literal intervals such that one contains $P_i^\delta$ and the other contains either $P_i^{1-\delta}$ or $Q_i^{1-\delta}$, $\delta \in \{0, 1\}$, it is not possible to have conflicting assignments to a variable. Since a literal being set to 1 is present in a clause, and every clause has a literal set to 1, $\Phi$ is proved to be satisfiable. This completes the proof.

### 6.1.5 The gadgets for non-oriented vertices

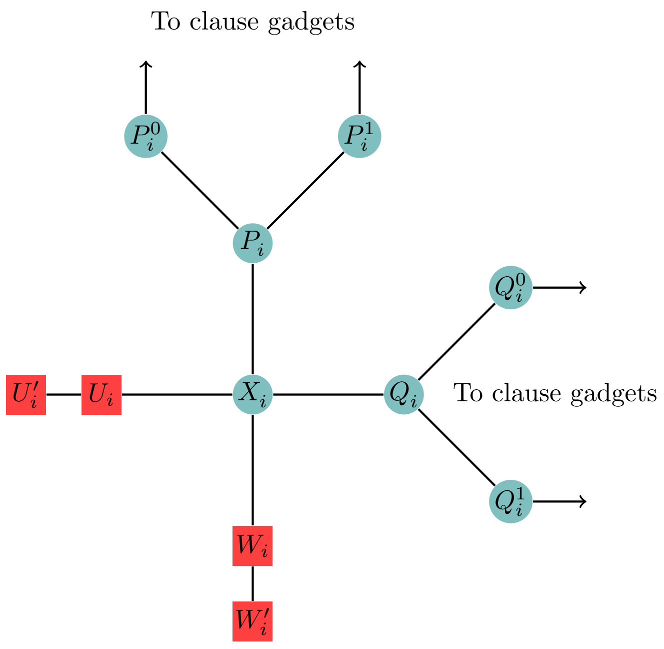
It is necessary to note that Theorem 6.1 holds irrespective of whether the vertices in the instance are oriented or unoriented. The reduction specified in the previous sections has been described for an instance with only oriented vertices, but it can easily be adapted to one with unoriented vertices. In fact, the gadgets in this case turn out to be much smaller, as depicted in Figures 6.4a and 6.4b.

Following this construction, the details of the proof are essentially identical, and the idea of assigning literals on the basis of the chosen literal intervals remains the same. The main difference is that the clause gadget does not have any cycles induced by the vertices in it. This is a consequence of the orientation of the vertices in the main proof. Furthermore, no repeat spanning interval in the unoriented reduction has to use a vertex more than once in this reduction, something that was not avoided in the oriented reduction. So, we have the following question.

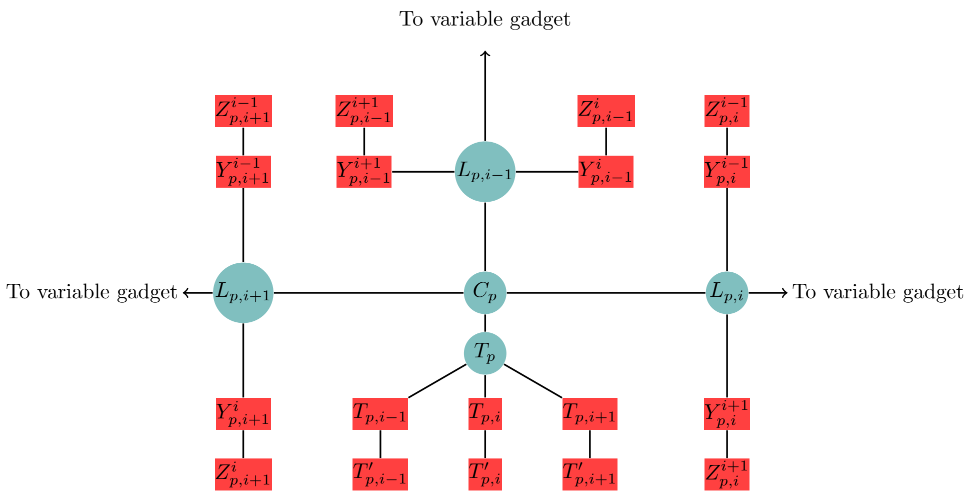
**Question 6.1.** Is Problem 5.1 tractable when every repeat spanning interval in the instance $(H, \mu, w)$ is framed by oriented vertices, and no repeat vertex occurs more than once in a repeat spanning interval?

## 6.2 Fixed parameter tractability

Following Theorem 6.1, the question that remains is if there is any way we can control the complexity in the problem in order to obtain polynomial time algorithms. Theorem 5.1 was a restriction on the size of the repeat spanning intervals. We now show an algorithm that runs in polynomial time for cases where the size of the repeat clusters, defined as the product of the number of vertices in the cluster and the maximum multiplicity, is fixed.



(a) Variable gadget for unoriented instances.



(b) Clause gadget for unoriented instances.

Figure 6.4: Gadgets for reduction using unoriented instances. Note that every vertex has degree at most 4. Since the underlying adjacency instance is realizable in the mixed genome model, the maximum multiplicity of the vertices can be restricted to 2. The literal intervals here run from the vertices $U_j$ or $W_j$ to $T_{p,i}$, where $l_{p,i}$ is a literal in the clause $c_p$ of the boolean variable $x_j$. The clause intervals run from $Y_{p,i}^{j}$ to $Y_{p,j}^{i}$ within a clause gadget, and use every repeat on the unique path between these two vertices exactly once.

### 6.2.1 Using the knapsack structure

We stated in Section 5.1.2 that Problem 5.1 is closely related to the multidimensional knapsack problem. This turns out to be useful, since there is a known dynamic programming formulation for solving the knapsack, and by extension, the multidimensional knapsack problem (a general survey is available in [86]). This is an exact exponential time algorithm in which the exponential factor is restricted to the total capacity of the knapsack. The result we get can be stated as follows.

**Theorem 6.2.** *Problem 5.1 is fixed parameter tractable for the mixed genome model when parameterized by the maximum size of a repeat cluster, $\rho$, and the maximum multiplicity, $\bar{\mu}$.*

The first thing to note is that we can always restrict the analysis to single repeat clusters. Recall that repeat clusters are connected components in the induced instance on the set of repeats. Assuming we have oriented frontier vertices, there is no repeat spanning interval such that it contains vertices from 2 extended repeat clusters. Since repeat spanning intervals are defined over individual repeat spanning intervals, such a vertex, if any, must belong to the frontier, and an oriented frontier vertex can belong to at most 1 extended repeat cluster. In the case of unoriented frontier vertices, we can consider an arbitrary doubling of the undoubled frontier vertex. Each end of this doubling can belong to exactly 1 extended repeat cluster, since the vertex itself has multiplicity 1 and the underlying adjacency instance is realizable in the mixed model.

We can now work on this modified instance without modifying any of the constraints. Then, we define the knapsack restrictions on the set of repeat spanning intervals arise from the extended repeat cluster it spans. These are restrictions defined by the multiplicity of the vertices, which make sure that no vertex exceeds its multiplicity in a genome map which is compatible with the chosen set of repeat spanning intervals. If the input instance contains several repeat clusters, we can treat them all as independent problems by partitioning the set $E_I$ into subsets that span each repeat cluster separately. Since a repeat belongs to exactly 1 repeat cluster, the set of repeat spanning intervals can be partitioned as such.

### 6.2.2 Complications in the new problem

There is a significant difference between the knapsack problem and Problem 5.1. Problem 5.1 specifies that the output instance must contain all adjacencies that were specified in the underlying adjacency instance given as input. If we add certain repeat spanning intervals, it may be possible that we use up all copies of a vertex while adding them. Then, adjacencies in the adjacency instance which were are not compatible with a repeat spanning interval added to the optimal set may no longer be compatible with a mixed genome map.

This means that we need to add additional constraints to the knapsack in order to keep track of which edges are compatible with the repeat spanning intervals in the set $S$ that we are constructing. Adjacencies between repeats within a repeat cluster will be given binary capacities, which keeps track of whether they are compatible with a repeat spanning interval in the set $S$ or not. A solution is accepted if this capacity is filled for every edge $\{u, v\}$ in the instance. Based on these observations, we now define notation to describe the algorithm.

Let the input instance $(H, \mu, w)$, where $H = (V, E_A \cup E_I)$ consist of a single repeat cluster $R$, the frontier vertices $F(R)$, adjacencies between $F(R)$ and $R$, and repeat spanning intervals over it. We denote the repeat spanning intervals in $E_I$ by $\{e'_0, \ldots, e'_{n-1}\}$, and the vertices in $R$ are labelled $\{v_0, \ldots, v_{\rho-1}\}$. The set $E_F = \{f_0, \ldots, f_{\kappa-1}\}$ is the set of edges to frontier vertices $F(R)$ of the repeat cluster.

Define a function $\mu^* \colon E_F \to \mathbb{N}$, which maps the frontier edges to 1. Also define a set of functions $\nu_t \colon E_I \to \mathbb{N}$ for each $t \in R \cup E_F$ as follows.

1. For $v_i \in R$, $\nu_{v_i}\left(e'_j\right) = q$ if $v_i$ appears exactly $q$ times in the sequence $o\left(e'_j\right)$.

2. For $f_i \in E_F$, $\nu_{f_i}\left(e'_j\right) = 1$ if $f_i$ is compatible with $o\left(e'_j\right)$, and 0 otherwise.

In general, we use $\nu_{v_i}$ (respectively $\nu_{f_i}$) to denote the vector $(\nu_{v_i}(e'_0), \ldots, \nu_{v_i}(e'_{n-1}))$ (resp. $(\nu_{f_i}(e'_0), \ldots, \nu_{f_i}(e'_{n-1})))$.

We also generalize the notation $\nu_t$ to accommodate the binary constraints we need for edges within a repeat cluster. Let $E_R = \left\{e_0, \ldots, e_{|E_R|-1}\right\}$ be the set of edges with both ends in $R$. Set $\nu_{e_i}\left(e'_j\right)$ to 1 if the repeat spanning interval $e'_j$ uses the edge $e_i$, and set it to 0 otherwise. These values are treated as booleans rather than integers.

We will find it convenient to define functions $\eta \colon E_I \to \mathbb{Z}_{\geq 0}^{\rho+\kappa}$ and $\theta \colon E_I \to \{0,1\}^{|E_R|}$. For $\eta$, the first $\rho$ indices run over the set of repeats $R$ and the next $\kappa$ indices run over the frontier edges of $R$. For $\theta$, the $|E_R|$ indices run over the edges between repeats in $R$. These functions are defined as follows.

1. $\eta(e'_i) = (\nu_{v_0}(e'_i), \ldots, \nu_{v_{\rho-1}}(e'_i), \nu_{f_0}(e'_i), \ldots, \nu_{f_{\kappa-1}}(e'_i))$. All entries in this array are integers.

2. $\theta(e'_i) = \left(\nu_{e_0}(e'_i), \ldots, \nu_{e_{|E_R|-1}}(e'_i)\right)$. This array only has boolean entries.

Given sets $A_0, \ldots, A_{q-1}$, we use $[A_0, \ldots, A_{q-1}]$ to denote the Cartesian product $A_0 \times \ldots \times A_{q-1}$ of these sets. We use $\mathbf{j}$ to denote a vector $(j_0, \ldots, j_{l-1},)$. The dimension of the vector space, $l$, will be specified in the context of the vector being referred to. We say $\mathbf{x} \leq \mathbf{y}$ if, for all $0 \leq i < l$, we have $x_i \leq y_i$. For two boolean arrays $\mathbf{k}$ and $\mathbf{q}$ of length $l$, we use the notation $\mathbf{k} \vee \mathbf{q}$ to denote the bitwise-OR operation at every coordinate of the array, i.e. $(\mathbf{k} \vee \mathbf{q})_i = k_i \vee q_i$ for all $0 \leq i < l$.

The dynamic programming array, which will be recursively constructed, is denoted by $M[k][\mu(v_0), \ldots, \mu(v_{\rho-1}), \mu^*(f_0), \ldots, \mu^*(f_{\kappa-1})]\left[\mathbb{1}_{|E_R|}\right]$. The first index goes over the set of $n$ repeat spanning intervals, $E_I$. The second set of indices goes over all possible subsets of occurrences of repeats and frontier edges, a multidimensional array of size $\rho + \kappa$. The third set goes over edges in $E_R$, with only boolean values, and is a multidimensional array of size $|E_R|$. The value $M[i][\mathbf{j}][\mathbf{q}]$ is the maximum weight set of repeat spanning intervals up to $e'_i$ such that the copies of a repeat $v_j$ (frontier edge $f_j$) used is at most $(\mathbf{j})_{v_j}$ ($(\mathbf{j})_{f_j}$), and the set of edges in $E_R$ that are compatible with a realization is given by the boolean array $\mathbf{q}$.

### 6.2.3 Setting up the recurrences

We now examine the various possibilities for augmenting the optimal set $S$ of repeat spanning intervals. Consider the entry $M[i][\mathbf{j}][\mathbf{k}]$ in the array. The restrictions we have at this point are the following.

1. Only a subset of the first $i$ repeat spanning intervals can be in the set $S$.
2. For every repeat $v_p$, at most $j_{v_i}$ copies are used at this point.
3. For every frontier edge $f_i$, the entry $j_{f_i}$ is at most 1.
4. For every adjacency $e_s$ between repeats $v_p$ and $v_q$, such that $e_s = 0$, at least one of the following conditions is true:
   (i) $j_{v_p} \neq 0$ and $j_{v_q} \neq 0$,
   (ii) or $e_s$ is compatible with the repeat spanning interval $e'_i$, i.e. $\nu_{e_s}(e'_i) = 1$
   (iii) or $k_s = 1$, i.e. it was compatible with one of the first $i - 1$ repeat spanning intervals which was already added to the optimal set.

The second and third conditions are enforced by checking if $\mathbf{j} \geq \eta(e'_i)$. If this condition is violated, it means that adding the repeat spanning interval $e'_i$ to the optimal set $S$ would result in a repeat, or a frontier edge being used more often than the constraints associated to it.

The last constraint is checked by iterating through all possibilities for the vector $\mathbf{k}$, which lies in $\{0,1\}^{|E_R|}$. We will not add the repeat spanning interval $e'_i$ if none of the conditions is satisfied.

For both the cases discussed above, the algorithm should not include $e'_i$ in the set $S$. The recurrence can be stated as follows.

$$M[i][\mathbf{j}][\mathbf{k}] = M[i-1][\mathbf{j}][\mathbf{k}]. \tag{6.1}$$

Finally, if all conditions are satisfied, the algorithm makes the choice to either add $e'_i$ to the set $S$ or not on the basis of the scenario that maximizes the weight of the set $S$ constructed up to that point. The maximum weight scenario must be chosen out of the following possibilities.

1. There is a subset of $S'$ of repeat spanning intervals $\{e'_0, \ldots, e'_{i-1}\}$ which does not contain $e'_i$, such that it satisfies both the constraints on $\mathbf{j}$ and $\mathbf{k}$.

2. There is a subset $S'$ of repeat spanning intervals $\{e'_0, \ldots, e'_{i-1}\}$, such that adding the interval $e'_i$ satisfies the constraints on $\mathbf{j}$ and $\mathbf{k}$.

The first is a single scenario, whose weight is given by $M[i-1][\mathbf{j}][\mathbf{k}]$. For the second, in order to make sure that adding $e'_i$ satisfies the constraint on $\mathbf{j}$, look at $M[i-1][\mathbf{j} - \eta(e'_i)]$. To make sure that the constraints on the inclusion of the $\mathbf{k}$ is satisfied, consider the following cases for every $e_j$ such that $k_j = 1$.

1. $e_j$ is not compatible with any of the intervals already added, but it is compatible with $e'_i$, i.e. there exists $\mathbf{q} \in \{0,1\}^{|E_R|}$ such that $q_j = 0$, and $(\theta(e'_i))_j = 1$.

2. $e_j$ is compatible with one of the intervals added before, and it is also compatible with $e'_i$, i.e. there exists $\mathbf{q} \in \{0,1\}^{|E_R|}$ such that $q_j = 1$, and $(\theta(e'_i))_j = 1$.

3. The edge is compatible with one of the intervals added before, but it is not compatible with $e'_i$, i.e. there exists $\mathbf{q} \in \{0,1\}^{|E_R|}$ such that $q_j = 1$, and $(\theta(e'_i))_j = 0$.

This means we have to take the maximum over all $M[i-1][\mathbf{j}][\mathbf{q}]$, where $\mathbf{q} \in \{0,1\}^{|E_R|}$ such that $\mathbf{q} \vee \theta\left(e'_j\right) = \mathbf{k}$. Thus, the recurrence can be written as

$$M[i][\mathbf{j}][\mathbf{k}] \leftarrow \max\left(M[i-1][\mathbf{j}][\mathbf{k}], \max_{\substack{\mathbf{q}\in\{0,1\}^{|E_R|},\\ \mathbf{q}\vee\theta\left(e'_i\right)=\mathbf{k}}} \{M[i-1][\mathbf{j} - \eta(e'_i)][\mathbf{q}] + w(i)\}\right). \tag{6.2}$$

The final value returned will be $M[n][\mu(v_0), \ldots, \mu^*(f_{\rho-1})]\left[\mathbb{1}_{|E_R|}\right]$, which indicates the following things.

1. The number of copies of every repeat $v_i \in R$ does not exceed $\mu(v_i)$.

2. The frontier edges are used exactly once.

3. Every adjacency within the cluster is compatible with a realization.

Backtracking from this value gives the set of repeat spanning intervals added. Algorithm 1 gives the pseudocode for Theorem 6.2.

### 6.2.4 Analysis

#### Algorithm correctness

At each iteration of the index $i$, the algorithm checks if a certain repeat spanning interval $e'_i \in E_I$ can be added, based on the intervals that have already been included, and the

**Algorithm 1** Dynamic programming scheme for Problem 5.1 on the mixed genome model.

```
Input Mixed genome realizable instance $(H_A, \mu, w_A)$, $H_A = (V, E_A)$, repeat spanning interval set $E_I$, $w\colon E_I \to \mathbb{R}^+$.
Output Maximum weight subset $S \subseteq E_I$ such that $(H', \mu, w')$, $H' = (V, E_A \cup S)$ is realizable in the mixed genome model.
for $\mathbf{j} \in [\mu(v_0), \dots, \mu(v_{\rho-1}), \mu^*(f_0), \dots, \mu^*(f_{\kappa-1})]$ do
    for $\mathbf{k} \in \{0,1\}^{|E_R|}$ do
        $M[0][\mathbf{j}][\mathbf{k}] \leftarrow 0$                              ▷ Initialize DP array.
    end for
end for
for $i \in [n]$ do                                      ▷ Iterate over set of repeat spanning intervals
    for $\mathbf{j} \in [\mu(v_0), \dots, \mu(v_{\rho-1}), \mu^*(f_0), \dots, \mu^*(f_{\kappa-1})]$ do
        if $\mathbf{j} \geq \eta(e'_i)$ then                                ▷ Check if $e'_i$ can be added
            for $\mathbf{k} \in \{0,1\}^{|E_R|}$ do
                $keeping\_edges\_constraint \leftarrow$ True
                for each edge $e_s = (v_p, v_q)$, $0 \leq s < |E_R|$ do
                    if $k_s = 0$ and $\nu_{e_s}(e'_i) = 0$ and
                      $\left((\mathbf{j} - \eta(e'_i))_p = \nu_{r_p}(e'_i)\right.$ or $\left.(\mathbf{j} - \eta(e'_i))_q = \nu_{r_q}(e'_i)\right)$ then
                                                  ▷ Edge not in $e'_i$, and one of the ends is exhausted
                        $M[i][\mathbf{j}][\mathbf{k}] \leftarrow M[i-1][\mathbf{j}][\mathbf{k}]$
                        $keeping\_edges\_constraint \leftarrow$ False
                        break
                    else if $k_s = 0$ and $\nu_{e_s}(e'_i) = 1$ then            ▷ Edge cannot be added
                        $M[i][\mathbf{j}][\mathbf{k}] \leftarrow M[i-1][\mathbf{j}][\mathbf{k}]$
                        $keeping\_edges\_constraint \leftarrow$ False
                        break
                    end if
                end for
                if $keeping\_edges\_constraint =$ True then                 ▷ Adding $e'_i$ possible
                    $$M[i][\mathbf{j}][\mathbf{k}] \leftarrow \max\left( M[i-1][\mathbf{j}][\mathbf{k}], \max_{\substack{\mathbf{q} \in \{0,1\}^{|E_R|}, \\ \mathbf{q} \vee \theta(e'_i) = \mathbf{k}}} \{M[i-1][\mathbf{j} - \eta(e'_i)][\mathbf{q}] + w(i)\} \right)$$
                                                                          ▷ Choose best scenario.
                end if
            end for
        else
            for $\mathbf{k} \in \{0,1\}^{|E_R|}$ do
                $M[i][\mathbf{j}][\mathbf{k}] \leftarrow M[i-1][\mathbf{j}][\mathbf{k}]$          ▷ Multiplicity constraints violated.
            end for
        end if
    end for
end for
return $M[n][\mu(v_0), \dots, \mu^*(f_{\kappa-1})]\left[\mathbb{1}_{|E_R|}\right]$
```

remaining multiplicity of the vertices involved in the interval. This condition is checked in line 8 of the algorithm. If this condition is violated, (6.1) can be applied to update the array.

The only extra condition needed is to make sure that every edge in $E_R$ remains compatible in the final instance. This is done by branching within the algorithm. The vector $\mathbf{k}$ keeps track of all possible combinations of edges that have been included via repeat spanning intervals up to $e'_i$.

Line 12 checks if adding $e'_i$ uses up all copies of one of the vertices adjacent to an edge $e_j \in E_R$, such that $k_j = 0$. If this is the case, then since $e_j$ must be compatible with a realization, but is not compatible with $o\left(e'_i\right)$, we cannot add the repeat spanning interval, and the recurrence is specified by (6.1).

If, on the other hand, $e_j$ is compatible with $e'_i$, but $k_j = 0$, this means that we are not allowed to have a set of repeat spanning intervals at this stage such that $e_j$ is compatible with the order associated to one of them. This implies that we still cannot add $e'_i$ to our set. This case is checked in line 16

The third case, implemented in line 23, is when adding the repeat spanning interval does not violate any constraints. This is the recurrence as specified in (6.2). The returned value, as desired, is $M\left[n\right]\left[\mu\left(v_0\right), \ldots, \mu^*\left(f_{\rho-1}\right)\right]\left[\mathbb{1}_{|E_R|}\right]$.

## Algorithm complexity

First, note that since the underlying adjacency instance is realizable in the mixed genome model, we get the inequality $|E_R| + |E_F| \leq \bar{\mu}\rho$, where $\bar{\mu} = \max_{v \in \overline{R}} \mu\left(v\right)$, and $\rho$ is the number of repeats in the cluster. We will use this inequality quite often in the runtime analysis.

The initialization of the array $M$ iterates through all possible assignments of knapsack capacities, and takes time $O\left(\prod_{v_i \in R} \mu\left(v_i\right) \times \prod_{f_i \in E_F} \mu^*\left(e_i\right) \times 2^{|E_R|}\right)$. The main loop checks if the $i^{th}$ interval can be added or not. Within this loop, we iterate over all possible indices $\mathbf{j} \times \mathbf{k}$ for the array $M\left[i\right]$. This is bounded by $\bar{\mu}^{\rho} 2^{\bar{\mu}\rho}$ (since $|E_F| = |F\left(R\right)|$ if all frontier vertices are oriented, and $E_R + E_F \leq \bar{\mu}\rho$) possibilities. Within this loop is a second loop which iterates over all possible boolean vectors in $\{0,1\}^{|E_R|}$, which takes time $O\left(2^{\bar{\mu}\rho}\right)$, since $E_R \leq \bar{\mu}\rho$.

There is an inner loop over the edges in $|E_R|$ and checks if a given edge has been added or not. This takes time $O\left(|E_R|\right)$, which is at most $\bar{\mu}\rho$ again. If a given edge has not been added, but adding the interval does not violate any constraints, then we are faced with the choice of adding the interval or not. In this case, take the maximum weight scenario between not choosing the repeat spanning interval, or adding it to one of the sets of repeat spanning intervals such that, on adding the interval, we include exactly the edges specified by the vector $\mathbf{k}$. This is a choice over the 1's in the vector $\mathbf{k}$, which iterates over all vectors in $\{0,1\}^{|E_R|}$. So, for the inner loop, over the choices of $\mathbf{k}$, we get a complexity of $O\left(2^{|E_R|}|E_R|\right)$. This gives us a total complexity of $O\left(2^{2|E_R|+\bar{\mu}\rho}\bar{\mu}^{\rho}|E_R|n\right)$, and using $|E_R| \leq \bar{\mu}\rho$, we get $O\left(2^{3\bar{\mu}\rho}\bar{\mu}^{\rho+1}\rho n\right)$.

The algorithm also uses exponential space to store the array $M$, of the order of $O\left(n\prod_{v_i\in R}\mu\left(v_i\right)\times\prod_{f_i\in E_F}\mu^*\left(e_i\right)\times 2^{|E_R|}\right)$.

## 6.3 Can we do better?

It is known that certain classes of packing problems admit polynomial time approximations to an arbitrarily small factor [45]. If such techniques can be adapted to Problem 5.1, this would result in a considerable speedup which can give solutions close to the optimal. When we consider the reasonably small number of repeat clusters one encounters in certain types of genomic data, this may prove to be a practical asset.

The theory of fixed parameter tractability usually makes heavy use of randomization to achieve better running times [53]. Considering that the parameters for the dynamic programming algorithm are not as natural as, say, the number of repeat spanning intervals to delete, randomization may be the key to achieving fixed parameter tractable algorithms in other parameters.

# Part III

# Vertex Orderings in Hypergraphs

Chapter 7

# Overview of vertex ordering problems

This part of the document is a marked shift in the theme of research. Up to now, we have viewed the problems we have had as either decision problems, or as optimization problems in which we are required to find a maximum weight subset of the hyperedges that satisfies some property. In this part, though, we will be considering optimization problems in which we have to find a *layout* of the vertices of the input instance which optimizes some objective function.

The next two chapters will always assume that the multiplicity function for the instance is set identically to 1 for all vertices, i.e. unique markers, and we will be studying realizability in the linear genome model. In other words, we shall be interested in the classical C1P problem, and how it relates to vertex ordering problems in graphs. We shall also assume that all the vertices are unoriented. Thus, the set of vertices $V$ only consists of vertices in $V^u$. Please note that the terms hyperedge and edge are used synonymously in the next two chapters, since we do not usually distinguish between adjacencies and intervals in the instance.

## 7.1 Motivation

Let $H = (V, E)$ be an instance in which all vertices are mapped by the multiplicity function to 1, and the hyperedges are weighted by a function $w \colon E \to \mathbb{R}^+$. Assume that we wish to check for realizability in the linear genome model, i.e. we wish to see if a given set of unique markers we have can be arranged into a linear genome such that the synteny information encoded in the instance is respected.

If there exists a map $M$ in the linear genome model such that every vertex appears exactly once in $M$, and every edge in $E$ is compatible with it, or in other words, if $H$ has the consecutive ones property (C1P), then $H$ is realizable. But more often than not, this is not the case on real data [43, 170]. Then, we have to come up with a notion of optimizing the instance in order to get one which does have the C1P.

In Problem 3.2, we stated that the aim is to discard a set of hyperedges of minimum weight such that the remaining instance is compatible with a map in the given genome model. Performing such an operation on $H$ would indeed output an instance which has the C1P, and for which we can find a realization in the linear model. Let $S \subset E$ be the set hyperedges discarded, and let $M$ be a realization of $H' = (V, E \setminus S)$ in the linear genome model. Some of the hyperedges in the set $S$ may exhibit the following two types of errors.

1. *Missing vertices*: There may be hyperedges $e \in S$ which are *almost compatible* with $M$. By this, we mean that there may be a consecutive subsequence of vertices in $M$ such that all the vertices in $e$ are represented, but there are a small number of vertices that are not in $e$ which also present in the subsequence. In evolutionary genomics, such an error may arise from small-scale convergent rearrangements.

2. *Chimeric hyperedges*: There may be hyperedges in $e \in S$ which can be partitioned into 2 or more 'large' sets of vertices, such that each such set of vertices is compatible with $M$. Such a hyperedge $e$ encodes false synteny information, but each of the compatible partitions is actually correct information.

Chimeric hyperedges are indeed a problem, and should be discarded as such. But in the first type of error, the hyperedges encode some synteny information, but they are being discarded outright in any exact, approximate or heuristic algorithm used to solve Problem 3.2. Furthermore, there are usually very few such hyperedges [43, 170]. But it is hard to distinguish them from the chimeric hyperedges in the data, and if we are optimizing the set of hyperedges, possibly useful synteny information may be discarded.

In order to get around this problem, there is one easy solution: we choose to keep all hyperedges rather than discarding any. The idea is to order markers (i.e. vertices) on a genome map (i.e. a linear order) while not distorting adjacencies or intervals (i.e. hyperedges) too much. Then, hyperedges that are not distorted a lot are either compatible with the order, or inferred to have missing vertices. Hyperedges that are distorted a lot can be putatively classified as chimeric. There are many ways to define what a good genome map means in this sense. This is the topic of discussion in the rest of the chapter.

## 7.2 The gapped C1P problems

One natural notion of what a good genome map is comes from observing the C1P from the point of view of the binary incidence matrix associated to the input instance. Recall that the incidence matrix of a hypergraph $H = (V, E)$, where $|V| = n$ and $|E| = m$ is the $m \times n$ binary matrix $M$ with rows indexed by hyperedges and the columns indexed by vertices,

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Figure 7.1: An example of a gap in a binary matrix. There are 2 gaps in the second row of the matrix, one having a length of 1, and one having length 2. There is no permutation of the columns which decreases the number of gaps in this instance, though the size of the gap can be reduced if we allow the creation of a gap in another row.

and $m_{ij} = 1$ if and only if the vertex corresponding to the index $j$ occurs in the hyperedge corresponding to the index $i$. Then, we can define the notion of a *gap*.

**Definition 7.1.** Given an $m \times n$ binary matrix $M$, a set of consecutive columns, indexed from $t$ to $t + \ell - 1$, where $t > 1$ and $\ell \geq 1$ is said to form a *gap* of length $\ell$ in a row $i$ of $M$ if $m_{ij} = 0$ for all $j \in \{t, \ldots, t + \ell - 1\}$, and $m_{i,t-1} = m_{i,t+\ell} = 1$.

The total number of gaps in a row $i$ of $M$, under a permutation $\pi$ of the columns, is given by $\gamma_\pi (M, i)$. The maximum length of a gap in a row $i$, under the permutation $\pi$ of the columns, is given by $\lambda_\pi (M, i)$.

A gap in a row translates into a split in the synteny information encoded by the genome map: for the hyperedge corresponding to the row, the vertices in the hyperedge do not occur consecutively in the map. The vertices in the gap, which are not in the hyperedge (since their columns have 0 in the corresponding row), correspond to markers that are inferred to not form a synteny with those in the hyperedge, and are obstacles to the compatibility of the hyperedge. Figure 7.1 shows an example of a matrix in which there is a row with a gap.

There are two major problems regarding gaps. The first problem is to find a permutation of the columns that minimizes the total number of gaps in a binary matrix.

**Problem 7.1.** Find a permutation $\pi$ of the columns of a binary $m \times n$ matrix $M$ such that the following objective function is minimized.

$$\sum_{\text{Rows } i \text{ of } M} \gamma_\pi (M, i) .$$

The second problem forces a restriction on both the number of gaps within a single row, and the length of these gaps.

**Problem 7.2.** Let $k, \delta \in \mathbb{N}$ be constants, and let $M$ be a binary matrix. Is there a permutation $\pi$ of the columns such that

$$\max_{\text{Rows } i \text{ of } M} \gamma_\pi (M,i) \leq k, \quad \text{and} \quad \max_{\text{Rows } i \text{ of } M} \lambda_\pi (M,i) \leq \delta.$$

Problem 7.1 was shown to be NP-hard by Kou [122]. It is, however, equivalent to the metric Travelling Salesman Problem, and thus, there is a polynomial time 3/2-approximation algorithm for it using the Christofides algorithm (folklore). However, the problem only asks for a bound on the number of gaps, rather than asking for a bound on the size of the gaps. This means that a permutation of the columns that optimizes this objective function may choose to create large gaps in hyperedges, rather than keep the size of the gaps small, which would mean that many hyperedges would be inferred to be chimeric.

Problem 7.2 quantifies the size of the gaps as well, which means that creating chimeric hyperedges are penalized. The problem is NP-complete if any of the parameters $d, k, \delta$ is allowed to vary, for all constant values of the two remaining parameters [39, 97, 100, 137], except for a single case. This forms the main open question regarding Problem 7.2.

**Question 7.1.** If $k = 2$ and $\delta = 1$, can Problem 7.2 be solved in polynomial time?

Currently, there is no major evidence for either side of this argument. There is also no known approximation result for this problem. The only known positive result for this problem is when the maximum number of 1's in a row of the matrix, denoted by $d$, is bounded, and both parameters $k$ and $\delta$ are kept constant [39, 137]. This is a result based on an extension of an algorithm for deciding if a graph has bandwidth $b$, where $b$ is a constant, by Saxe [186]. We provide the exact definition of the bandwidth of a graph and how it connects to the problems we discuss in the next section.

The gapped C1P is an example of a *vertex ordering* problem on hypergraphs. A permutation of the columns of the incidence matrix can be interpreted as a permutation of the vertices themselves. Vertex ordering problems, which are usually defined on graphs, are ubiquitous in genomics [97, 112, 215], and especially in ancestral and comparative genomics [39, 170, 211, 218].

## 7.3 Vertex ordering problems in graphs

Consider a graph $G = (V, E)$. Suppose we lay out the vertices of $G$ on a line, i.e. define a total order on $V$. This defines a permutation $\pi \colon V \to \{1, \dots, n\}$. We can ask if there is a permutation $\pi$ which does not 'stretch out' the edges too much. There are various classical notions of defining what 'stretch out' means, and the problem of finding a permutation that minimizes a given objective function which calculates this notion is a well studied topic in algorithm design and computational complexity.

### 7.3.1 Defining the problems

We are interested in one such measure of stretching an edge. In the graph $G = (V, E)$, assume $w\colon E \to \mathbb{R}$ is a weight function on the edges. Then, given a permutation $\pi$ on the set of vertices, the *stretch* of an edge $e = \{u, v\}$ is defined as the following quantity.

$$str_{\pi}(e) = |\pi(u) - \pi(v)|.$$

We define two objective functions on $G$ using the stretch of an edge.

1. The *minimum linear arrangement* of $G$ is the following quantity.

$$MLA(G) = \min_{\pi\colon V \to \{1,\ldots,n\}} \sum_{e \in E} w(e) \cdot str_{\pi}(e).$$

2. The *bandwidth* of $G$ is the following quantity.

$$bw(G) = \min_{\pi\colon V \to \{1,\ldots,n\}} \max_{e \in E} w(e) \cdot str_{\pi}(e).$$

Both, the minimum linear arrangement and the bandwidth of a general graph, are known to be NP-hard to compute [91, 171]. In fact, even computing the bandwidth of trees is hard [149], as is computing it for graphs of degree at most 3 [90]. As such, the computation of these objective values usually falls into the ambit of approximation algorithms, exact exponential algorithms, or fixed parameter tractability.

These are not the only linear arrangement problems encountered in literature. Other related problems include the minimum cutwidth problem and the minimum distortion problem (see [23] for a list of such problems). We do not discuss all vertex ordering problems in the manuscript, since we wish to introduce the basic notion of vertex ordering problems and get preliminary results on the same.

### 7.3.2 Known results

Many vertex ordering problems on graphs have existed for a long time in mathematical literature, and thus, there is a rich theory behind them. Some of the problems have been proved to be tractable for highly structured graph classes. For example, the minimum linear arrangement problem is known to be solvable in polynomial time on trees, a result of Goldberg and Klipker [96].

An approximation for the minimum linear arrangement problem was first obtained by Even et al. [76], and improved using rounding from a linear relaxation by Rao and Richa [178]. The latest results, obtained through semidefinite relaxations, and combining techniques from the previous results, has been obtained by Charikar et al. [34], and Feige and Lee [78].

**Theorem 7.1.** *[34, 78] Given a graph $G = (V, E)$, $|V| = n$, with non-negative edge weights $w\colon E \to \mathbb{R}$, there exists a polynomial time algorithm which can approximate $MLA\,(G)$ to within a factor of $O\left(\sqrt{\log n} \log\log n\right)$ of the optimum.*

At a high level, this result can be interpreted as a fallout of the $O\left(\sqrt{\log n}\right)$-factor approximation algorithm for sparsest cut in graphs [6, 7]. The latter result allows 'cheap partitions' of graphs, such that not too many edges cross the partition compared to the number of edges within each partition. The key to the approximation algorithm for the minimum linear arrangement problem is to recursively find such 'cheap partitions', and use it to order the vertices. Since the number of edges crossing a partition are few, the expected number of edges that are stretched too far is small. That being said, the rounding technique and the analysis are non-trivial [35].

On the other hand, Devanur et al. [59] proved the following result limiting the power of the semidefinite programming route to get better approximations.

**Theorem 7.2.** *[59] The semidefinite relaxation of the minimum linear arrangement problem with the triangle inequality constraints has an integrality gap of at least $\Omega\,(\log\log n)$.*

So, the results obtained by Charikar et al. [34] and Feige and Lee [78] are optimal for semidefinite relaxations that include the triangle inequality. At the moment, there is no known relaxation for the problem which has achieved a better result.

The case of the bandwidth problem is even harder. Blum et al. first obtained an approximation up to a factor of $O\left(\sqrt{n/b}\log n\right)$ of the bandwidth, where $b$ is the optimal bandwidth of the input instance [22]. Feige proved that, short of using a set of constraints different from those for the minimum linear arrangement problem, it is not possible to obtain an approximation for the bandwidth minimization problem which is better than $O\,(n^{\epsilon})$, for any $\epsilon > 0$ [77]. He used the concept of *volume respecting embeddings* in order to get an approximation ratio of $O\left((\log n)^3 \sqrt{\log n \log\log n}\right)$ for the problem.

There are also many heuristic approaches for these problems. Of particular interest are heuristics based on the spectral properties of the graphs for both the graph bandwidth problem [104] and the minimum linear arrangement problem [111]. These are results that build on the results of Cheeger [44], Alon and Milman [3], which establish that a 'cheap partition' of a graph can be obtained by computing the second smallest eigenvector corresponding to the Laplacian of a graph, and that the cost of this partition is bounded by functions of the second smallest eigenvalue.

Besides these, there are also exact exponential time algorithms for some of the problems [23], building on the dynamic programming algorithm for the Travelling Salesman problem. Where fixed parameter tractability is concerned, these problems usually cannot be parameterized by the value of the objective function or by the treewidth of the input graph, since they cannot be expressed in monadic second order logic [50]. However, many of the problems (save minimum linear arrangement) can be parameterized by the size of the vertex cover of the input graph in order to get a polynomial time algorithm [80].

## 7.4 Generalizing to hypergraphs

The minimum linear arrangement and bandwidth can be generalized to hypergraphs as well. But as in any generalization, we can take various interpretations of what the 'stretch' of a hyperedge means. The notion of gaps discussed in Section 7.2 localized the stretch within a hyperedge to the vertices contained in a gap.

Definitions for concepts such as bandwidth for hypergraphs are not unknown. For example, consider the following definition of *hypergraph bandwidth*. The *adjacency matrix* of a hypergraph $H = (V, E)$ is defined to be the symmetric real matrix $A$ with entry $a_{ij}$ being equal to the number of hyperedges in which the vertices corresponding to indices $i$ and $j$ both occur for $i \neq j$, and diagonal entries set to 0.

**Definition 7.2.** [182] The *bandwidth of a symmetric matrix* $A$ is said to be $k$ if $a_{ij} = 0$ for all $i, j$ satisfying $|i - j| > k$. The *bandwidth* of a hypergraph $H = (V, E)$ is the smallest possible bandwidth of its adjacency matrix.

Under this definition, there are also some results based on the eigenvalues of the 'Laplacian' matrix of hypergraphs, defined in terms of the adjacency matrix [182]. The problem with this definition is that the adjacency matrix of a hypergraph, as defined here, is not unique to it, and so notions like the minimum linear arrangement are less easy to interpret.

The interpretations we introduce here consider the stretch of a hyperedge as a global property of how the vertices contained in it are placed in the vertex order, and define stretch not just in terms of intervening vertices, but of the total length of the hyperedge.

### 7.4.1 Cumulative stretch

Assume that the notion of stretch is just that: how far is an edge stretched in a linear arrangement of the vertices. In other words, how many vertices does it span over? At this level, the notion of stretch of a hyperedge is just the same. Formally, given a hypergraph $H = (V, E)$, and a permutation $\pi \colon V \to \{1, \ldots, n\}$, the stretch of a hyperedge $e \in E$ is defined as follows.

$$str_{\pi}(e) = \max_{u,v \in e} |\pi(u) - \pi(v)|.$$

It follows that when $|e| = 2$ for all $e \in E$, i.e. when $H$ is a graph, the stretch coincides with the classical notion of the stretch of an edge in a graph. It is equivalent to the notion of hypergraph bandwidth as introduced by Rodríguez [182]. However, since it is defined for an edge, we need not quantify over pairs of vertices, and do not have to resort to the adjacency matrix.

We can define generalizations of the minimum linear arrangement and the bandwidth problems to hypergraphs.

**Definition 7.3.** Let $H = (V, E)$ be a hypergraph on $n$ vertices, and let $w\colon E \to \mathbb{R}^+$ be a weight function on the edges. The *minimum cumulative stretch* of $H$ is defined to be the following quantity.

$$MCS\left(H\right) = \min_{\pi\colon V \to \{1,\ldots,n\}} \sum_{e \in E} w\left(e\right) \cdot str_\pi\left(e\right).$$

This problem was considered by Banerjee et al. [11], who called it the *hypergraph optimal linear arrangement* (HOLA) problem, and proved that there is a $O\left(d\sqrt{\log n}\log\log n\right)$-approximation for it, where $d$ is the largest size of a hyperedge. We avoid their terminology for the sake of clarity, since we are defining two possible generalizations of this problem.

**Definition 7.4.** Let $H = (V, E)$ be a hypergraph on $n$ vertices, and let $w\colon E \to \mathbb{R}^+$ be a weight function on the edges. The *minimum edge stretch* of $H$ is defined to be the following quantity.

$$MES\left(H\right) = \min_{\pi\colon V \to \{1,\ldots,n\}} \max_{e \in E} w\left(e\right) \cdot str_\pi\left(e\right).$$

### 7.4.2 Spread

A less obvious way of generalizing the stretch is as a density measure. Suppose we care not only about how far a hyperedge $e \in E$ of a hypergraph $H = (V, E)$ stretches, but also on how evenly the vertices of the hyperedge are spaced through this stretch. This is the motivation behind the *spread* of a hyperedge $e \in E$, which we define as follows, given a permutation $\pi$ of the vertices.

$$spr_\pi\left(e\right) = \sum_{u,v \in e} \left|\pi\left(u\right) - \pi\left(v\right)\right|.$$

Again, note that when $|e| = 2$, the spread of $e$ is the same as the stretch of an edge in a classical graph. So, we get another possible generalization of the minimum linear arrangement and the bandwidth problems to hypergraphs.

**Definition 7.5.** Let $H = (V, E)$ be a hypergraph on $n$ vertices, and let $w\colon E \to \mathbb{R}^+$ be a weight function on the edges. The *minimum cumulative spread* of $H$ is defined to be the following quantity.

$$MCSP\left(H\right) = \min_{\pi\colon V \to \{1,\ldots,n\}} \sum_{e \in E} w\left(e\right) \cdot vol_\pi\left(e\right).$$

**Definition 7.6.** Let $H = (V, E)$ be a hypergraph on $n$ vertices, and let $w\colon E \to \mathbb{R}^+$ be a weight function on the edges. The *minimum edge spread* of $H$ is defined to be the following quantity.

$$MESP(H) = \min_{\pi\colon V \to \{1,\ldots,n\}} \max_{e \in E} w(e) \cdot vol_\pi(e).$$

### 7.4.3 Connection to the C1P

The minimum linear arrangement and the bandwidth of a graph on $n$ vertices measure, in some sense, how far the graph is from the path on $n$ vertices. Given a path, its minimum linear arrangement is exactly $n-1$, and its bandwidth is exactly 1. These are the best possible values for any graph, since the vertex ordering defined by the path does not induce any edge to stretch.

But is there a similar notion for hypergraphs? What exactly do the cumulative stretch and the cumulative spread measure? For that matter, what is the analogue of a path for hypergraphs? There could well be many suitable answers to that, and indeed, there have been many definitions for a 'path' in hypergraphs [114, 115, 199]. We will add another definition to this list, and make a case for why it is useful.

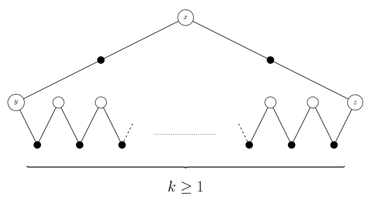
**Definition 7.7.** Given a hypergraph $H = (V, E)$, and a set of vertices $S \subseteq V$, the induced hypergraph $H[S]$ is a *path* if the incidence matrix associated with $H[S]$ has the consecutive ones property.

This definition needs some background. The notion is motivated by the forbidden substructure classification of consecutive ones matrices. Tucker proved that a binary matrix has the consecutive ones property if and only if it avoids 5 classes of submatrices, 2 of fixed size, and 3 of variable size [202]. This family submatrices have since been called Tucker patterns. The bipartite representation of these matrices/hypergraphs, with the white vertices representing vertices, and the black ones representing hyperedges, are shown in Figure 7.2.

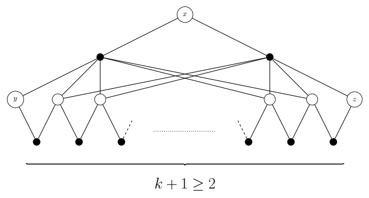
It is a simple observation to note that, for a graph, the presence of a branching (a vertex of degree 3) or a cycle is the only obstruction to a linear order of minimum value (i.e. bandwidth 1 and linear arrangement score $n-1$). In the matter of hypergraphs, this role is played by Tucker patterns, which again skew the minimum cumulative stretch, minimum edge stretch, minimum cumulative spread and the minimum edge spread costs.

*Observation.* The cumulative stretch and cumulative spread of a hypergraph $H = (V, E)$, $w\colon E \to \mathbb{R}^+$, as well as its edge stretch and edge spread, is minimized when $H$ has the consecutive ones property.

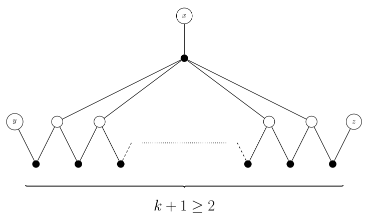
This fits the analogy to paths, and lends some credence to the claim. Note, however, that the guaranteed value of these objective functions is not going to be a constant in the number of vertices anymore. The value of the minimum cumulative stretch for a C1P



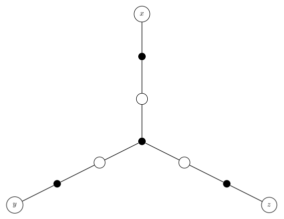
(a) $G_{I,k}$



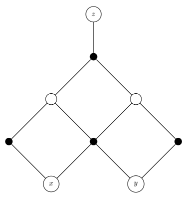
(b) $G_{II,k}$



(c) $G_{III,k}$



(d) $G_{IV}$



(e) $G_V$

Figure 7.2: The Tucker patterns. Hypergraphs are represented as bipartite graphs, with white nodes depicting vertices, and black nodes representing hyperedges. The first three patterns are parameterized by their size, while the other two are of fixed size. We have marked three vertices in each pattern. These form *asteroidal triples*, vertices such that there is a path between any two which avoids the closed neighbourhood of the third.

hypergraph $H = (V, E)$ with weights mapping to 1, for example, will be $\xi = \sum_{e \in E} |e|$, which depends on the structure of $H$.

Recall we stated that the bandwidth problem on graphs is known to be notoriously hard even for trees [149], while the minimum linear arrangement problem is more tractable [49, 96, 195]. The gist of the algorithm is simple enough to describe in a few words: find a vertex such that all the subtrees created by deleting this vertex are small, and order these subtrees recursively. Then combine the orders such that the vertices of the smallest trees are placed on either side of the chosen vertex, and the vertices of successively larger trees are placed framing the previously constructed ordering of vertices on either side. The analysis, though, is highly non-trivial.

One may ask if there are similar instances for hypergraphs. The minimal obstruction definition of the consecutive ones property allows us to formulate the following conjecture.

**Conjecture 7.1.** *Let $H = (V, E)$ be a hypergraph which avoids all Tucker patterns except possibly $G_{III,1}$ and $G_{IV}$. The minimum cumulative stretch problem on $H$ can be solved in polynomial space and time.*

The corresponding Tucker patterns are shown in Figure 7.3. These are the only Tucker patterns in which deleting a single hyperedge causes the hypergraph to become disconnected, and thus capture a 'tree-like' notion on hypergraphs. As a result, we call such instances *hypertrees*. Algorithms for the minimum linear arrangement on trees rely on recognizing a

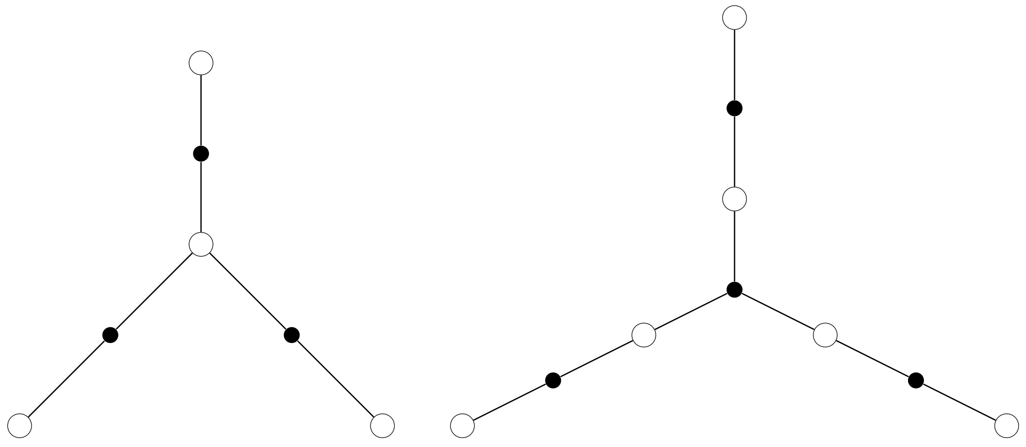

(a) $G_{III,1}$ (b) $G_{IV}$

Figure 7.3: Tucker patterns that are allowed in hypertrees. Note that deleting a single hyperedge in these two patterns renders them disconnected.

vertex which is used as a pivot around which the arrangement is made. A similar argument is shown in a minor result on a small class of hypertrees in Appendix D.

Finding a minimum edge stretch ordering of a hypertree seems a much more daunting challenge [149]. Perhaps more interesting is the existence of a fixed parameter tractable algorithm for this problem when parameterized by the size of the vertex cover [80]. The key behind such an algorithm for the graph bandwidth problem is the fact that the complement of a vertex cover is an independent set of the graph. The analogous concept in hypergraphs would be to parameterized the maximum edge stretch problem by the size of the hitting set. However, the complement of a hitting set need not be an independent set anymore, which means the arguments for graphs will not work here. It is more likely that we will have to include an extra parameter, such as the maximum hyperedge size, in order to retain fixed parameter tractability. So this remains a field to be explored.

Since we are motivating this study as a possible method to detect chimeric hyperedges, one particular point of interest would be to measure the relative deviation from the C1P brought about by the addition of a hyperedge to a hypergraph that has the C1P. If the hypergraph formed by adding the hyperedge still has the C1P, then of course the problem is trivial. But consider the following problem.

**Problem 7.3.** Let $H = (V, E)$ be a hypergraph that has the C1P, and let $e \in 2^V$ be a hyperedge such that $H' = (V, E \cup \{e\})$ does not have the C1P. Find a C1P ordering of $H$ such that the edge stretch (edge spread) of $e$ is minimized.

If a C1P ordering of $H$ that minimizes a consecutivity measure for the new hyperedge $e$ can be found efficiently, then one can infer whether the hyperedge $e$ is chimeric or not depending on the value of this measure. Finding such a C1P ordering can be interpreted

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

(a) A binary matrix $M$.

$$\begin{bmatrix} 5 & -2 & -1 & -2 \\ -2 & 4 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -2 & -1 & -1 & 4 \end{bmatrix}$$

(a) The Laplacian of $M$.

Figure 7.6: The Laplacian matrix associated to a binary matrix $M$. The Laplacian $L_M$ is not unique to $M$.

as searching and restricting the permutations encoded by a subtree of the PQ-tree data structure associated to the hypergraph $H$. If this can be done efficiently for the number of gaps, edge stretch or edge spread of $e$, it is possible to obtain some conclusions about the relative 'badness' of the hyperedge.

**Spectral methods for C1P problems**

Since the notions of stretch and spread can serve as a measure of how close a hypergraph is to having the C1P, one would expect that there are heuristic algorithms for the C1P that compute near optimal orders. However, to our knowledge there is only one algorithm that can handle non-C1P instances and return a meaningful vertex order for the same. This algorithm was originally defined for the following problem, which is called the *seriation problem*.

**Problem 7.4.** Let $A$ be an $n \times n$ real symmetric matrix. Does there exist an $n \times n$ permutation matrix $P$ such that the following properties holds true for the matrix $B = P^T AP$?

$$b_{i,j} \leq b_{i,j+1} \text{ for } j \leq i,$$
$$b_{i,j} \geq b_{i,j+1} \text{ for } j \geq i.$$

Atkins et al. developed a spectral algorithm for the seriation problem [9]. This algorithm is of interest because, given a hypergraph $H$ having the C1P, i.e. its incidence matrix $M$ has the C1P, the algorithm, when run on the symmetric matrix $A = M^T M$, outputs the set of permutations that are the C1P orderings of $M$. The crux of the algorithm is to minimize the following objective function.

$$\min_{\mathbf{x}\in\mathbb{R}^n, \mathbf{x}^T \mathbb{1}_n = 0} \frac{\mathbf{x}^T L_A \mathbf{x}}{\mathbf{x}^T \mathbf{x}}, \tag{7.1}$$

where $L_A$ is the *Laplacian* matrix associated to the incidence matrix $M$ and to the symmetric matrix $A$. This is the matrix with off-diagonal entries $(-a_{ij})$, and diagonal entries $\sum_{j\neq i} a_{ij}$. The optimal solution to this problem is the second eigenvalue of $L_A$. By sorting the eigenvector associated to that eigenvalue, and using a recursive divide-and-conquer strategy, the output can be read out as a PQ-tree.

The interpretation of the permutations encoded in the PQ-tree is not obvious when the matrix $M$ does not have the C1P. Vuokko showed that they minimize the principal component of a norm which computes the total number of gaps (consecutive blocks of 0-entries between 1's) in $M$ [205]. Formally, Vuokko's result reads as follows.

**Theorem 7.3.** *[205] Let $M$ be an $m \times n$ binary matrix. Let $P$ be an $(n+2) \times (n+2)$ permutation matrix, and $\mathbf{S}$ be the $(n+1) \times (n+2)$ discrete differential operator, and let $\hat{M}$ be the matrix in which a column of 0's is added to the beginning and end of $M$. Let $L$ be the Laplacian matrix associated to $M$, and let $L = R\Lambda R^T$ be its eigenvalue decomposition, where $R = (\mathbf{r}_0 \ldots \mathbf{r}_{k-1})$, and $\Lambda$ is a diagonal matrix with distinct eigenvalues of $L$, $\lambda_0 < \ldots < \lambda_{k-1}$ as entries. The Frobenius norm of the matrix $\|\mathbf{S}P\hat{M}^T\|_F$, which is given by*

$$\|\mathbf{S}P\hat{M}^T\|_F^2 = \sum_{i=0}^{k-1} (\alpha - \lambda_i)\,\|\mathbf{S}P\mathbf{r}_i\|^2,$$

*where $\alpha > \max_i \lambda_i$ is a positive real number, is the total number of gaps in the matrix $\hat{M}$ when ordered by the permutation matrix $P$. Then, Atkins et al.'s spectral algorithm outputs a set of permutations that minimize the second term of this formula.*

In this sense, Atkins et al.'s algorithm serves as a heuristic for Problem 7.1. However, it is instructive to note that the objective function (7.1) can be rewritten as follows.

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}\left(x_i - x_j\right)^2,$$

where $n$ is the number of columns in $M$, and $a_{ij}$ is the corresponding entry in $A$. This entry is simply the number of rows in which both column $i$ and $j$ have a 1. In terms of hypergraphs, this is the number of edges that contain both vertices, the one corresponding to column $i$, and the one corresponding to column $j$.

If all hyperedges in $H$ have weight 1, then $a_{ij}$ is precisely the number of times that the vertices $u, v$, corresponding to the indices $i$ and $j$ respectively, occur in the same hyperedge. Thus, we are summing over pairs of vertices within a single hyperedge for all hyperedges. If we replace the term $\left(x_i - x_j\right)^2$ by $|\pi(u) - \pi(v)|$, the resulting objective function is the following.

$$\sum_{u,v\in V}^{n}\sum_{e\in E} \delta_e(u,v)\,|\pi(u) - \pi(v)|,$$

where $\delta_e(u,v) = 1$ if $u, v \in e$, and 0 otherwise. If we minimize this objective function over the set of all permutations of the vertices instead of embeddings of vertices in $\mathbb{R}$, this is equivalent to the objective function for the minimum cumulative spread as stated in Definition 7.5. The difference is that Atkins et al.'s approach uses a linear relaxation of the objective with different constraints. It remains to be seen if there are spectral bounds to the spread that can be obtained through an algorithm similar to the one described by Atkins et al.

### 7.4.4 New results

Since the cumulative stretch and spread of a hypergraph are generalizations of the minimum linear arrangement problem on graphs, there is no known polynomial time algorithm to compute them. As stated, it is not even known if it is computable on hypertrees.

The main contribution in Chapter 8 is the adaptation of the approximation algorithm for the minimum linear arrangement problem to compute the cumulative stretch and spread with a factor of $O\left(\sqrt{\log n}\log\log n\right)$ of the optimum. This is the best possible approximation bound using semidefinite programming with the stated constraints [59].

# Chapter 8

# Approximating vertex ordering problems on hypergraphs

We described the motivation behind hypergraph vertex ordering problems in the previous chapter. We stated that recent advancements in graph partition problems motivated the state-of-the-art vertex ordering problems on graphs [6, 34]. Unfortunately, approximations for hypergraph partition problems, which lead to vertex ordering approximations, are not well-studied.[1]

In the case of the minimum cumulative stretch and spread problem, we show that we can achieve approximation ratios similar to that for the minimum linear arrangement problem on graphs. Using the techniques developed first by Rao and Richa [178], and later applied to the minimum linear arrangement problem by Charikar et al. [34], and independently by Feige and Lee [78], we can achieve an approximation ratio of $O\left(\sqrt{\log n}\log\log n\right)$ for these problems. To give a brief overview, the idea is to relax the integer linear program for the problems to a semidefinite program. After solving this semidefinite relaxation under a certain set of constraints, the solution can be rounded through a recursive divide-and-conquer procedure while keeping the cost of the linear arrangement bounded [76].

The result for the minimum cumulative stretch improve on the $O\left(d\sqrt{\log n}\log\log n\right)$-approximation obtained by Banerjee et al. [11], by making the approximation factor independent of the size of the hyperedge. The result for minimum cumulative spread presents a new way of characterizing how close to having the C1P an instance is.

Before we start discussing the results in this chapter, we present a few concepts on metric spaces, which will be referred back to through the rest of the chapter.

[1]There has been some recent work to rectify this [129, 130]. These results show that the theoretical approximation ratio achieved on hypergraph partitioning problems such as minimum expansion and small-set expansion matches the corresponding results on graphs [8].

## 8.1 Spreading metrics and $\ell_2^2$–representations

A crucial step in the algorithms for the minimum cumulative stretch and minimum cumulative spread is to map the vertices to points in a vector space. This embedding is expected to conform to a metric space on the set of points associated to the vertices of the hypergraph. Then, this metric space is 'flattened' by mapping it to a line, thus finding an $\ell_1$–metric. The expectation is that we can find a 'flattening' that makes sure that the hyperedges are not stretched too much.

The approach we take starts by defining the metric under which the points corresponding to the embedded vertices in the vector space are chosen. Informally, the vertices must map to vectors such that the square of the Euclidean distance between them forms a metric. For a set of $n$ objects, we define an $\ell_2^2$–representation as follows.

**Definition 8.1.** [8] An $\ell_2^2$*–representation* over $n$–points is a set of $n$ vectors $\{\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}\}$ such that the distance $\ell$ on this set, defined as $\ell(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^2$, forms a metric over the set.

The word 'representation' is used to indicate that the vectors represent combinatorial objects, such as the vertices of a graph, or, in our case, the vertices of a hypergraph. An $\ell_2^2$–representation can be described as a function $f\colon V \to \mathbb{R}^d$, and we represent $f(v)$, $v \in V$, by $\mathbf{x}_v$.

At the same time, we have to make sure that the points that the vertices map to are not too close to each other. Since we need a linear ordering, we need the points to be spread apart such that, on mapping them to the real line, every subset of points is far apart. Otherwise, we may end up with subsets of points that are clumped close together, and if the set of vertices corresponding to the points has too many edges that go to vertices outside this set, the stretch of these edges might not be easy to bound. This motivates the next definition.

**Definition 8.2.** We say that an $\ell_2^2$–representation on some set $V$ of objects is *strongly well-spread* if

$$\sum_{u \in S} \ell(\mathbf{x}_v, \mathbf{x}_u) \geq \frac{(|S|^2 - 1)}{4} \ \forall\, v \in S \ \forall S \subseteq V,\ |S| > 1, \tag{8.1}$$

where $\mathbf{x}_v$ is the vector associated with $v \in V$ in the representation, and $\ell(\mathbf{x}_v, \mathbf{x}_u) = \|\mathbf{x}_v - \mathbf{x}_u\|^2$.

It is important to note that this notion of being 'well-spread' is a property of a metric space, as compared to the 'spread' of a hypergraph, as used in Definitions 7.5 and 7.6.

Finally, we stated that we will be using a divide-and-conquer based rounding scheme for the algorithm. This necessitates that we are able to find a discrete divide in the set of points associated to the vertices. This is the motivation for the following definition.

**Definition 8.3.** Given a metric space $(X, \ell)$, two points $\mathbf{x}_i, \mathbf{x}_j \in X$ are said to be $\Delta$–*separated* if $\ell(\mathbf{x}_i, \mathbf{x}_j) \geq \Delta$ for $\Delta > 0$. Two sets $S, T \subset X$ are $\Delta$–separated if every pair of points $\mathbf{x}_i \in S$, $\mathbf{x}_j \in T$ are $\Delta$–separated.

## 8.2 Cumulative stretch

We will prove the following theorem on the approximability of the minimum cumulative stretch.

**Theorem 8.1.** *Given a hypergraph $H = (V, E)$, $|V| = n$ with non-negative edge weights $w \colon E \to \mathbb{R}$, there exists a polynomial time algorithm which can approximate $MCS(H)$ to within a factor of $O\left(\sqrt{\log n} \log \log n\right)$ of the optimum.*

In order to start designing the algorithm, the first observation we need to make is that a permutation $\pi$ on the set of vertices defines a natural metric, which is the difference $|\pi(u) - \pi(v)|$ in the positions of two vertices $u, v \in V$ when permuted by $\pi$. This metric is called a *permutation metric*, and it is unique to a permutation up to reversal. We will try to find a metric which approximates the permutation metric defined by an optimal order for the minimum cumulative stretch problem. To do this, we will relax the integer linear program (ILP) associated to the cumulative stretch problem to a semidefinite program (SDP).

In the next section, we first describe the linear program relaxation of the minimum linear arrangement problem for graphs, as obtained by Rao and Richa [178]. This was an important result, since the constraints used by them are the ones used for subsequent results on the minimum linear arrangement, and will be the same that we use for the minimum cumulative stretch problem. Furthermore, the methods used by them to obtain the approximation ratio will be reused by us. Following this, we give the semidefinite relaxation used to prove Theorem 8.1, and motivate it by the fact that the optimal ordering for the minimum cumulative stretch is a feasible solution to this relaxation.

After we have introduced the setting of the problem, we will explain an important result on well-spread representations, which forms a major argument in the algorithm. We use this result and the solution to the semidefinite program in order to design the algorithm, and follow this up by analyzing the performance of the algorithm.

### 8.2.1 Relaxing the integer linear program

The minimum linear arrangement problem for a graph $G = (V, E)$ can be formulated as an integer linear program, where we have to find a permutation $\pi \colon V \to \{1, \ldots, n\}$ which minimizes $\sum_{e \in E} w(e) \cdot str_\pi(e)$. Alternately, we can formulate this as finding a permutation metric which minimizes the ILP objective instead, where $str_\pi(e)$ is the distance between the two vertices in $e$, as defined by the metric. Since a permutation metric is unique to a permutation up to reversal, it is easy to see that if a permutation gives the optimal value of the linear arrangement, then so does its reversal.

The importance of a linear program formulation of a combinatorial problem cannot be overstated. Integer linear programs are well studied, and highly generalizable. Even though solving them is an NP-hard problem, they admit various relaxations in the constraints or the solution space. Since linear programs are solvable in polynomial time [98], these relaxations are often used as a first approximation to the solution to hard combinatorial problems, which are then rounded to get an approximation bound.

Rao and Richa [178] used this very idea, along with the notion of approximating permutation metrics by a spreading metric, to get an approximation of $O(\log n)$ for the minimum linear arrangement problem on graphs. Their relaxation, in which they minimize over the set of all spreading metrics $\ell'$, reads as follows.

$$\begin{aligned} &\min_{\ell' :\, E \to \mathbb{R}_{\geq 0}} \sum_{e \in E} w(e) \cdot \ell'(e) \\ &\text{subject to} \\ &\qquad \ell'(e) \geq 0 \;\forall u, v \in V, \\ \sum_{u} &\mathbf{dist}(u, v) \geq \frac{|S|^2}{4} \;\forall v \in S \;\forall S \subseteq V, |S| > 0, \end{aligned}$$

where $\mathbf{dist}(u, v)$ is defined as the distance from $u$ to $v$ in the graph $G$, assuming that each edge $e$ has length $\ell'(e)$.

To understand the motivation behind the last constraint, note that in its absence, it is possible for $\ell'$ to map edges to 0. This is obviously no good for us; in order to make sure that the vertices are spread apart by $\ell'$, the spread constraint is added, though it is slightly different from what we stated in Definition 8.2. This difference is a consequence of the fact that the spread constraints in Rao and Richa's formulation are based on the distances assigned to *edges* by the metric $\ell'$. As we shall see in the later sections, we choose to ignore distances defined by the underlying graph/hypergraph structure in subsequent algorithms. Thus, while the minimum distance between vertices in the ILP relaxation is 1, for the constraints we use, which are defined by (8.1), the minimum distance between any 2 vertices will be 3/4.

Under these constraints, the metric $\ell'$ is a relaxation of the permutation metric. Indeed, the permutation metric satisfies both constraints specified in the ILP. This can be checked with a polynomial time oracle specified by Even et al. [76], and which we explain in the next section. Since we are indeed looking for a permutation metric on the set of the vertices, this relaxation is suitable for our purposes.

### 8.2.2 Semidefinite Relaxation

The next evolution of the approximation for the minimum linear arrangement problem was the use of a semidefinite relaxation instead of the linear programming formulation given above. This was the idea behind the algorithms of Charikar et al. [34], and Feige and Lee [78] for the problem.

In the case of the minimum cumulative stretch problem, the corresponding semidefinite relaxation is given below.

$$\min_{\mathbf{x}} \sum_{e \in E} w(e)\, s_e \tag{8.2}$$

subject to

$$\ell(\mathbf{x}_v, \mathbf{x}_u) \leq s_e \qquad \forall\, u, v \in e,\ \forall\, e \in E, \tag{8.3}$$

$$\ell(\mathbf{x}_u, \mathbf{x}_v) \leq \ell(\mathbf{x}_u, \mathbf{x}_w) + \ell(\mathbf{x}_w, \mathbf{x}_v)\ \ \forall\, u, v, w \in V, \tag{8.4}$$

$$\sum_{u \in S} \ell(\mathbf{x}_v, \mathbf{x}_u) \geq \frac{(|S|^2 - 1)}{4}\ \ \forall\, v \in S, \forall\, S \subseteq V,\ |S| > 1, \tag{8.5}$$

where $\ell(\mathbf{x}_v, \mathbf{x}_u) = \|\mathbf{x}_v - \mathbf{x}_u\|^2$.

Constraints (8.3) enforce the stretch condition. According to the given inequality, no two vertices in a hyperedge $e$ are mapped farther than $s_e$ apart in the $\ell_2^2$-representation. Constraints (8.4) enforce the triangle inequality, making sure that the $\ell_2^2$ distances computed indeed form a metric. Constraints (8.5) make sure that the $\ell_2^2$-representation is well-spread.

Note that there are an exponential number of spreading constraints. Since we want a polynomial time approximation algorithm, checking all these constraints is not feasible. To get around this, we will need to define a polynomial time separation oracle. The purpose of this oracle is to check, for every candidate solution $X = \{\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}\}$, if there is a violated constraint, and if there is, to return that constraint. Such an oracle was specified by Even et al. [76], and is given below.

1. For every $i \in [n]$, order the distances $\ell(\mathbf{x}_i, \mathbf{x}_j)$ in ascending order for all $j \in [n]$ (including $j = i$).

2. For every $k \in [n]$, compute the sum of the distance from $\mathbf{x}_i$ to the $k$ closest points to $\mathbf{x}_i$. If this sum is under $(k^2 - 1)/4$, return the set containing these points as a set that violates a spreading constraint.

3. If no such $i$ is found, all constraints are satisfied, and return $X$ as a feasible solution.

It is easy to check that this algorithm runs in polynomial time. Given a point $\mathbf{x}_i$, ordering the points in ascending order of distance to this point takes $O(n \log n)$ time, and it takes $O(n)$ to compute the sum of the distances for all sets. Doing so for all points, the total runtime for the oracle is $O\left(n\left(n\log n+n\right)\right)$. In order to check that this oracle does indeed detect a violated constraint in (8.5), note that if a set of the $k$ closest points to $\mathbf{x}_i$ satisfies the constraint, every other set of size $k$ containing $\mathbf{x}_i$ must also satisfy the constraint.

It is also important to note that an optimal solution to the cumulative stretch integer program is a feasible solution to this SDP. Consider a permutation $\pi$ of the $n$ vertices of the hypergraph $H=(V,E)$ such that $\sum_{e\in E} w\left(e\right)\cdot str_{\pi}\left(e\right)$ is minimized. Construct the following set of vectors $X$: for each vertex $v$ such that $\pi\left(v\right)=i$, $\mathbf{x}_i$ is the vector in $\mathbb{R}^n$ with the first $i$ entries being $-1/2$, and the next $n-i$ entries being $1/2$. Given $\mathbf{x}_i$ and $\mathbf{x}_j$, the distance $\ell\left(\mathbf{x}_i,\mathbf{x}_j\right)=\|\mathbf{x}_i-\mathbf{x}_j\|^2$ will then be equal to $|i-j|$. So, it gives back the permutation metric, which is known to satisfy all constraints of the SDP.

### 8.2.3 A divide and conquer algorithm

#### Divide-and-conquer using spreading metrics

Divide-and-conquer algorithms using spreading metrics to specify the subproblems was first pioneered by Even et al. [76]. They used the concept for minimum linear arrangement in graphs, and designed a polynomial time algorithm that gave an approximation guarantee of factor $O\left(\log n\log\log n\right)$. While Rao and Richa improved upon their algorithm [178], it was not until Arora et al. first used a mixture of semidefinite programming and metric embeddings to get an $O\left(\sqrt{\log n}\right)$-factor approximation for sparsest cut that people started looking at divide-and-conquer methods in conjunction with semidefinite programs. The motivating theorem for the work on minimum linear arrangement since then was the following.

**Theorem 8.2.** *[6, 34, 78] There exist $c,\epsilon>0$ such that, given a finite, strongly well-spread $\ell_2^2$–representation on $n$ points, $(X,\ell)$, we can find $\Delta$–separated sets $S,T\subset X$ of size at least $\epsilon n$, where $\Delta\geq c\left(n/\sqrt{\log n}\right)$, in polynomial time.*

This implies that if there is a strongly well-spread $\ell_2^2$–representation formed by the set of vectors $X$, we can find sets $S,T\subset X$ which are sufficiently large and far apart. An intuitive explanation of the rationale behind Theorem 8.2 is that, given a well-spread $\ell_2^2$–representation, one can project the points on a random hyperplane, and with high probability, a large number (a $2\epsilon$ fraction at least) points will be projected 'far' (at distance $\Delta$) on the hyperplane, into two sets of comparable size. This is a stronger separability condition that the original separation argument made by Arora et al. [7], by virtue of the strong spreading constraints being used.

#### The algorithm

Assume that the input hypergraph is $H=(V,E)$, with weights $w\colon E\to\mathbb{R}^+$ on the edges. The first step of the algorithm is to solve the SDP specified by (8.2) for $H$. This can be done in polynomial time and space, by using the ellipsoid algorithm [98].

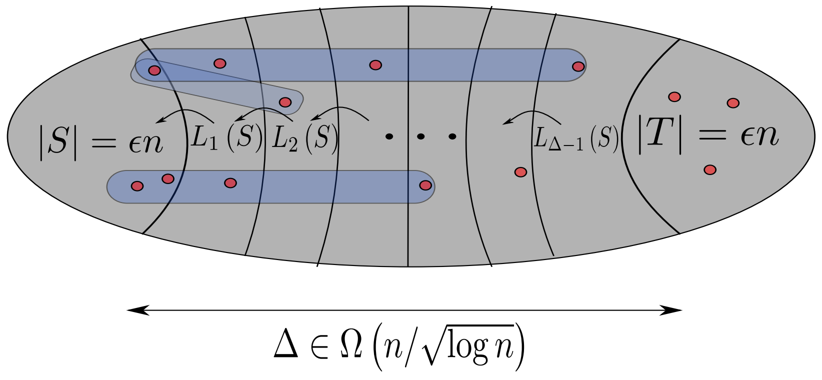


Figure 8.1: Splitting the set of well-spread vectors using Theorem 8.2. The overlays represent hyperedges containing the vertices corresponding to the points in them.

Let the optimum value of the objective function of the semidefinite relaxation be $W$. The output of the semidefinite program will be a positive-semidefinite $n \times n$ matrix. One can use a Cholesky decomposition to factor this matrix into $n$ vectors in $\mathbb{R}^n$, each associated to a vertex $V$ of the input hypergraph [98]. This gives an $\ell_2^2$–representation of the hypergraph $H = (V, E)$, which we call the set $X$ of vectors in $\mathbb{R}^n$. Let $\mathbf{x}_v$ denote the vector associated to vertex $v \in V$, and let $s_e$ denote the stretch of edge $e$ calculated by the semidefinite program.

The next step of the algorithm is to apply Theorem 8.2 in order to find sets $S, T$ of size $\Omega(n)$, which are $\Delta$-separated, for $\Delta \in \Omega\left(n/\sqrt{\log n}\right)$. The lower bounds on the size of the sets $S$ and $T$, as well as the exact value of $\Delta$ are specified by Theorem 8.2. From now, we will assume that the sets $S$ and $T$ have already been computed.

For a vertex $v \in V$, define $\ell(\mathbf{x}_v, S) = \min_{\mathbf{x}_u \in S} \ell(\mathbf{x}_v, \mathbf{x}_u)$, i.e. the distance from $\mathbf{x}_v$ to the closest point in $S$. For a non-negative integer $i$, we define the $i^{th}$ neighbourhood of $S$ as the set $N_i(S) = \{v \in V : \ell(\mathbf{x}_v, S) \le i\}$. This is the set of vertices whose associated vectors in $X$ are within distance $i$ of some vector in $S$, with the distance being defined by the metric $\ell$.

The $i^{th}$ *level* with respect to $S$, for $i \in [\Delta]$, is the set $L_i(S)$ defined as follows.

$$L_i(S) = \{e \in E : \exists u, v \in e,\ \ell(\mathbf{x}_u, S) \le i \wedge \ell(\mathbf{x}_v, S) > i\}. \tag{8.6}$$

In a particular iteration of the algorithm, we will always keep the set $S$ constant, and so we refer to the $i^{th}$-level simply as $L_i$. The total weight of edges in the level $L_i$ will be denoted by $\rho_i$, and will be called the *weight of* $L_i$. The notion of levels is important because if an edge is contained in levels $L_i$ and $L_j$, $i < j$, then it is also contained in $L_k$ for all $i \le k < j$. Thus, the set $L_i$ forms an edge cut in a hypergraph. Given $L_i(S)$, we denote this edge cut by $(N_i(S), V \setminus N_i(S))$. We now specify the recursive algorithm used to prove Theorem 8.1.

1. If $|E| = 0$ or $n \leq \lfloor 2/\epsilon \rfloor$, where $\epsilon$ is the constant defined in Theorem 8.2, return a random permutation of the vertices.

2. Find $\Delta$–separated sets $S, T \subset X$ using Theorem 8.2, $|S|, |T| \geq \epsilon n$, where $\Delta \geq (cn/\sqrt{\log n})$.

3. Define levels $L_i$ with weight $\rho_i$, for $i \leq [\Delta]$, as stated in (8.6).

4. Check if there exists a level $L_\delta$ such that $\rho_\delta \leq W/(\Delta \log n)$. If yes, find the edge cut $(N_\delta(S), V \setminus N_\delta(S))$ associated with $L_\delta$. Delete the edges in $L_\delta$ and compute the output of the algorithm on the two components of the cut. Let the vertex orders output for the subproblems on the hypergraphs induced by $N_\delta(S)$ and $V \setminus N_\delta(S)$ be $\sigma_\delta$ and $\sigma'_\delta$ respectively. Concatenate the two orders to get $\pi = \sigma_\delta.\sigma'_\delta$, and return $\pi$.

5. If no such $L_\delta$ exists, partition the levels into indices $I_t$ for $t \in \mathbb{N}$, where
$$I_t = \left\{ L_i : \frac{2^t W}{(\Delta \log n)} < \rho_i \leq \frac{2^{t+1} W}{(\Delta \log n)} \right\}. \tag{8.7}$$

6. For $t \in [2 + \log \log n]$, find an index $I_t$ of maximum cardinality. Let this index be $I_k = \{L_{a_1}, L_{a_2}, \ldots, L_{a_\kappa}\}$, where $a_1 \leq \ldots \leq a_\kappa$. Delete the hyperedges in the levels in $I_k$, and then define $H_i$ as the subhypergraph induced by the vertices in $N_{a_{i+1}}(S) \setminus N_{a_i}(S)$, for $i \in \{1, \ldots, \kappa - 1\}$. Let $H_0$ be the hypergraph induced by $N_{a_1}(S)$, and let $H_\kappa$ be the hypergraph induced by $V \setminus N_{a_\kappa}(S)$. Let $n_i$ be the number of vertices in $H_i$.

7. Recurse on each $H_i$. Let the vertex order output for $H_i$ be called $\sigma_i$.

8. Concatenate the vertex order $\sigma_i$ obtained for each $H_i$, and return $\pi = \sigma_0.\sigma_1.\ldots.\sigma_\kappa$.

Note that there are two branching steps in the algorithm. The first, in Step 4, decomposes the instance into two subproblems, each of which are solved independently, and whose results are later concatenated. The second, in Step 6, branches into multiple subproblems, and concatenates the results of each in the order of the subproblems.

### 8.2.4 Analyzing the algorithm

To analyze the algorithm, we set up a recurrence relation for the cumulative stretch under the vertex order obtained. The value of the cumulative stretch for a particular vertex ordering on a hypergraph $H$ specifies a *cost*, for which we will find an upped bound using this recurrence. The crucial ingredient is a charging scheme for the hyperedges, which distributes the stretch of the hyperedge over its length.

## Intermediary results

The following intermediary results are required to establish the recurrence relation for the cost of the cumulative stretch. These results are extensions of arguments made by Charikar et al. [34] for establishing approximation bounds for the minimum linear arrangement problem. The first is a lemma which establishes a lower bound on the stretch of a hyperedge based on the indices of the levels it is contained in.

**Lemma 8.1.** *Let $\{L_{b_1}, \ldots, L_{b_k}\}$ be a set of levels, as defined in (8.6) such that $b_1 < b_2 < \ldots < b_k$. Let $e \in E$ be a hyperedge which is in levels $L_{b_i}$ and $L_{b_j}$, $b_i < b_j$. Then, $s_e \geq (j-i)/2$.*

*Proof.* Consider an edge $e \in L_{b_i}, L_{b_j}$. This implies that there are vertices $u, v \in e$ such that $\ell(\mathbf{x}_v, S) \leq b_i$ and $\ell(\mathbf{x}_u, S) > b_j$. Here, we use the triangle inequality and the fact that $b_i < b_j$. Consider a point $\mathbf{x}_w \in S$.

$$\begin{aligned} \ell(\mathbf{x}_v, \mathbf{x}_w) + \ell(\mathbf{x}_v, \mathbf{x}_u) &\geq \ell(\mathbf{x}_u, \mathbf{x}_w) \\ \Rightarrow \ell(\mathbf{x}_v, \mathbf{x}_u) &\geq \ell(\mathbf{x}_u, \mathbf{x}_w) - \ell(\mathbf{x}_v, \mathbf{x}_w) \\ \Rightarrow \ell(\mathbf{x}_v, \mathbf{x}_u) &\geq b_j - b_i \\ &> b_j - b_i - 1, \end{aligned}$$

since $\ell(\mathbf{x}_u, \mathbf{x}_w) > b_j$ and $\ell(\mathbf{x}_v, x_w) \leq b_i$. We also know that $b_j - b_i \geq j - i$ for $i \leq j$, giving $\ell(\mathbf{x}_v, \mathbf{x}_u) > j - i - 1$ From constraints (8.5), we know that $\ell(\mathbf{x}, \mathbf{y}) \geq 3/4$. So, $\ell(\mathbf{x}_v, \mathbf{x}_u) \geq \max\{3/4, j-i-1\} \geq 1/2(j-i)$. Since constraints (8.3) specify that $s_e \geq \ell(\mathbf{x}_v, \mathbf{x}_u)$ for all $u, v \in e$, we get $s_e \geq (j-i)/2$. ☐

A special case of this lemma happens when the set of level $\{L_{b_1}, \ldots, L_{b_k}\}$ includes all levels $\{L_1, \ldots, L_\Delta\}$.

**Corollary 8.1.** *Let $e \in E$ be a hyperedge which is in levels $L_i$ and $L_j$, $i < j$. Then, $s_e \geq (j-i)/2$.*

We will use this corollary in the next result. This result states that the total mass of all levels $\{L_1, \ldots, L_\Delta\}$ is not too large. This is important, because it places a constraint on how much an edge can stretch, or on how many edges can have a high stretch.

**Lemma 8.2.** $\sum_{i=1}^{\Delta} \rho_i \leq 2W$.

*Proof.* Use Lemma 8.1 on the set of levels $L_1, L_2, \ldots, L_\Delta$. This gives $2s_e \geq (j-i)$ for any edge in levels $i, j$, $1 \leq i < j \leq \Delta$. Now, consider $\sum_{i=1}^{\Delta} \rho_i$. We can rewrite this as follows.

$$\begin{aligned}
\sum_{i=1}^{\Delta} \rho_i &= \sum_{i=1}^{\Delta} \sum_{e \in L_i} w(e) \\
\Rightarrow \sum_{i=1}^{\Delta} \rho_i &= \sum_{e \in E} w(e) \left| \{L_i : e \in L_i\} \right| \\
&\leq \sum_{e \in E} w(e) \times 2s_e = 2W.
\end{aligned}$$

This proves the lemma. □

These results are sufficient to explain what happens when the bipartition step is chosen in the course of the algorithm (Step 4). In order to analyze the rest of the algorithm, though, we need to know the size of the largest index, which tells how many subproblems the algorithm will recurse over (Step 6). We start by finding a lower bound on the total number of levels having a weight at most $4W/\Delta$.

**Lemma 8.3.** $\sum_{t=0}^{1+\log\log n} |I_t| \geq \Delta/2$.

*Proof.* From Lemma 8.2, we know that $2W \geq \sum_{i=1}^{\Delta} \rho_i$. If we wanted to sum up the weights of all levels having weight greater than $4W/\Delta$, this value is an upper bound, since we do not add up the weights of levels that are smaller. Let the number of levels with weight greater than $4W/\Delta$ be $\beta$. Then,

$$\begin{aligned}
2W &\geq \frac{4W}{\Delta}\beta \\
\Rightarrow \frac{\Delta}{2} &\geq \beta.
\end{aligned}$$

So, the number of levels with weight at most $4W/\Delta$, is at least $\Delta - \Delta/2 = \Delta/2$. By the definition of indices, this is the same as the sum $\sum_{t=0}^{1+\log\log n} |I_t|$. □

Using this sum, we can use a simple averaging argument to prove that the largest index must contain a 'lot of' levels. This is the purpose of the next lemma.

**Lemma 8.4.** $|I_k| \geq \Delta / (4 \log\log n)$.

*Proof.* Lemma 8.3 tells us that the total number of levels of weight less than $4W/\Delta$ is at least $\Delta/2$, and they are contained in $2 + \log\log n$ indices. Averaging this weight over all levels, we can conclude that there must be at least one index of size greater than $\Delta / [2(2 + \log\log n)] \geq \Delta / (4\log\log n)$. This proves the result. □

### Charging scheme for the hyperedges

The proof of the approximation bound relies on a amortized-like analysis of the cost of the stretch of the hyperedges. Consider a hyperedge $e \in E$. Assume that the index chosen in Step 6 of the algorithm is $\{L_{a_1}, \ldots, L_{a_\kappa}\}$, where $a_i < a_j$ for all $1 \leq i < j \leq \kappa$. Let edge $e$ belong to the levels $L_{a_i}$ and $L_{a_j}$, $i < j$, and to no level $L_{a_k}$ such that $k < i$ or $k > j$. By definition, $e$ must also belong to all edges $L_{a_p}$, where $i \leq p \leq j$. When we partition into subproblems using the levels in the index, we get subproblems $\{H_0, \ldots, H_\kappa\}$, and of these, the edge $e$ would have stretched over all the vertices in each $H_p$, $i \leq p < j$. It would also have possibly stretched over some vertices of $H_{i-1}$ and $H_j$. This gives an upper bound of $\sum_{k=i-1}^{j} n_k$ on the stretch of $e$ under the vertex order obtained.

Instead, assign a charge to each hyperedge, with the rule that a hyperedge is charged $n_{k-1} + n_k$ when it stretches over the vertices of $H_{k-1} \cup H_k$. This is the charge associated to the edge for belonging to the level $L_{a_k}$. For the edge $e$, which is contained exactly in the levels from $L_{a_i}$ to $L_{a_j}$, the total charge under this scheme will be $\sum_{k=i}^{j} (n_{k-1} + n_k) = n_{i-1} + 2(n_i + \ldots + n_{j-1}) + n_j$.

When we partition the instance in many subproblems, and charge the hyperedges as stated above, the total contribution of these edges to the cost of the ordering will be bounded above using the charging scheme. In other words, the contribution is bounded above by $\sum_{i=1}^{\kappa} [\rho_i (n_{i-1} + n_i)]$, since every edge in level $i$ is charged $n_{i-1} + n_i$.

### Bounding the total cost

For convenience, round up every $s_e$ as obtained from the semidefinite relaxation to $\lceil s_e \rceil$ for all $e \in E$. Let $\overline{W}$ be the value of the objective function (8.2) using the rounded values of $s_e$. Since $s_e \geq 3/4$ for all hyperedges, using $|S| = 2$ in constraints (8.5), we can conclude that $\overline{W} \leq W + \sum_e w(e) \leq (7/3)W \leq (7/3)MCS(H)$.

Let $C\left(\overline{Z}, n\right)$ denote the maximum cost of the cumulative stretch ordering on $n$ vertices obtained by the algorithm such that the value of the rounded objective function (8.2) is $\overline{Z}$. The recurrence relation is set up into two cases. If the instance is bipartitioned, as in Step 4 of the algorithm, this means that there existed a level $\delta$ such that $\rho_\delta \leq \overline{W}/(\Delta \log n)$. In the course of this bipartitioned, the hyperedges which were stretched out and removed from the subproblems had a cumulative weight of at most $\overline{W}/(\Delta \log n)$, and each of the hyperedges in $L_\delta$ is stretched by at most $n$, i.e. from the first vertex in the ordering to the last vertex in the ordering. Let the two subproblems obtained have maximum cost $C\left(\overline{W}_0, N_0\right)$ and $C\left(\overline{W}_1, N_1\right)$. The recurrence relation in this case is given below.

$$C\left(\overline{W}, n\right) \leq C\left(\overline{W}_0, N_0\right) + C\left(\overline{W}_1, N_1\right) + \frac{n\overline{W}}{\Delta \log n}, \tag{8.8}$$

where, $N_0, N_1 \geq \epsilon n$, since the subproblems contain the sets $S$ and $T$ respectively, and using

Theorem 8.2, $S, T \geq \epsilon n$. Also, $\overline{W}_0, \overline{W}_1 \leq W$, since these are strict subproblems, in the sense that they contained fewer vertices and hyperedges.

If such a level does not exist, on the other hand, the partition in Step 6 divides the instance into many smaller subproblems, depending on the number of levels in the chosen index $I_k$. By definition, as stated in (8.7), each of the $\kappa$ levels in $I_k$ has weight greater than $2^k\overline{W}/(\Delta \log n)$ and at most $2^{k+1}\overline{W}/(\Delta \log n)$. Using Lemma 8.4, we can conclude that $\kappa \geq \Delta/(4 \log\log n)$. Removing all the edges in the levels contained in $I_k$, the total weight of the edges removed amounts to $\sum_{i=1}^{\kappa} \rho_{a_i}/2$, where $\{L_{a_1}, \ldots, L_{a_\kappa}\}$ are the levels contained in $I_k$. This follows from Lemma 8.1, in the case we consider a hyperedge which only belongs to a single level, when its stretch is at least 1/2. Now use the charging scheme described in the previous section in order to bound the cost of the cumulative stretch from above in this case.

$$\begin{aligned} C\left(\overline{W}, n\right) &\leq C\left(\overline{W} - \sum_{i=1}^{\kappa} \frac{\rho_{a_i}}{2}, n\right) + \sum_{i=1}^{\kappa} \left[\rho_{a_i}\left(n_{i-1} + n_i\right)\right] \\ &\leq C\left(\overline{W} - \frac{2^{k-3}\overline{W}}{\log n \log\log n}, n\right) + \frac{2^{k+2} n \overline{W}}{\Delta \log n}. \end{aligned} \tag{8.9}$$

In the first line, the first term follows from the fact that the total cost of the edges removed is at least $\sum_{i=1}^{\kappa} \rho_{a_i}$. The second term is a consequence of the charging scheme being used. In the second line, we use the fact that the levels in $I_k$ have tight bounds on their weights, by their definition in (8.7). The lower bound is used in the first term to get an upper bound on the total cost of the subproblems, and the upper bound is used in the second term.

Using recurrences (8.8) and (8.9), we can write the final recurrence relation for $C\left(\overline{W}, n\right)$ as follows.

$$C\left(\overline{W}, n\right) \leq \max \begin{cases} C\left(\overline{W}_0, N_0\right) + C\left(\overline{W}_1, N_1\right) + \frac{n\overline{W}}{\Delta \log n}, \\ C\left(\overline{W} - \frac{2^{k-3}\overline{W}}{\log n \log\log n}, n\right) + \frac{2^{k+2} n \overline{W}}{\Delta \log n}. \end{cases}$$

To solve this recurrence relation, we use an inductive argument based on those of Charikar et al. [34]. The key to solving this recurrence is to consider each case separately. In the bipartition case, Theorem 8.2 provides tight constraints on the size of the subproblems which can be used for the inductive hypothesis. In the multipartition case, the inductive hypothesis is applicable since the total cost of the subproblems is guaranteed to be less than that of the original instance.

**Lemma 8.5.** *[34] There exists $c' > 0$ such that $C\left(\overline{W}, n\right) \leq c' \left(\sqrt{\log n} \log\log n\right) \overline{W}$ for all $n \geq 3$ and $\overline{W} \geq 0$.*

*Proof.* The induction is carried out over $\overline{W} + n$. For the base case, consider Step 1 of the algorithm. When $\overline{W} + n \leq \lfloor 2/\epsilon \rfloor$, we automatically get $n \leq \lfloor 2/\epsilon \rfloor$, and output a random ordering of the vertices. In this case, every layout has cost bounded from above by $c' \left(\sqrt{\log n} \log\log n\right) \overline{W}$.

The induction hypothesis is that the result is true for all $\overline{Z} + n' \leq \overline{W} + n$. Now assume that the branching used is specified by recurrence (8.8). The stated recurrence is given below for convenience.

$$C\left(\overline{W}, n\right) \leq C\left(\overline{W}_0, N_0\right) + C\left(\overline{W}_1, N_1\right) + \frac{n\overline{W}}{\Delta \log n}.$$

By Theorem 8.2, we know that the two subproblems must contain the sets $S$ and $T$ of size at least $\epsilon n$ each respectively. However, since the two subproblems consist of complementary sets of vertices, we also obtain $|N_0|, |N_1| \leq (1-\epsilon)\, n$. Also, as we stated, $\overline{W}_0, \overline{W}_1 \leq \overline{W}$. Then, using the induction hypothesis on $\overline{W}_0 + N_0$ and $\overline{W}_1 + N_1$, we get the following expansion of the right hand side, in which we assume that $C\left(\overline{W}_0, N_0\right) \geq C\left(\overline{W}_1, N_1\right)$.

$$\begin{aligned} C\left(\overline{W}, n\right) &\leq c'\left(\sqrt{\log N_0} \log\log N_0\right) \overline{W}_0 + c'\left(\sqrt{\log N_1} \log\log N_1\right) \overline{W}_1 + \frac{n\overline{W}}{\Delta \log n} \\ &\leq 2c'\left(\sqrt{\log N_0} \log\log N_0\right) \overline{W}_0 + \frac{n\overline{W}}{\Delta \log n} \\ &\leq 2c'\left(\sqrt{\log\left(1-\epsilon\right) n} \log\log n\right) \overline{W} + \frac{\overline{W}}{c\sqrt{\log n}}. \end{aligned}$$

For the last inequality, we use the fact that $\Delta \geq cn/\sqrt{\log n}$. Now, given $a, a+b > 0$, we have $\sqrt{a+b} \leq \sqrt{a} + b/\left(2/\sqrt{a}\right)$. This lets us resolve the right hand side as follows.

$$\begin{aligned} C\left(\overline{W}, n\right) &\leq 2c'\left[\left(\sqrt{\log n} + \frac{\log\left(1-\epsilon\right)}{2\sqrt{\log n}}\right) \log\log n\right] \overline{W} + \frac{\overline{W}}{c\sqrt{\log n}} \\ &\leq c'\left(\sqrt{\log n} \log\log n\right) \overline{W} + \frac{\overline{W}}{\sqrt{\log n}} \left(\frac{c' \log\left(1-\epsilon\right)}{2} \log\log n + \frac{1}{c}\right). \end{aligned}$$

For $n \geq 3$, if we set $c' \geq 2/\left(c|\log\left(1-\epsilon\right)|\right)$, the second term on the right is negative. This completes this case for the recurrence.

For the second case, we reproduce recurrence (8.9) here.

$$C\left(\overline{W}, n\right) \leq C\left(\overline{W} - \frac{2^{k-3}\overline{W}}{\log n \log\log n}, n\right) + \frac{2^{k+2} n\overline{W}}{\Delta \log n}.$$

Using the induction hypothesis, and the fact that $\Delta \geq cn/\sqrt{\log n}$, we can expand the right hand side as follows.

$$C\left(\overline{W},n\right)\leq c'\left(\overline{W}-\frac{2^{k-3}\overline{W}}{\log n\log\log n}\right)\sqrt{\log n}\log\log n+\frac{2^{k+2}\overline{W}}{c\sqrt{\log n}}.$$

By rearranging terms, this becomes

$$C\left(\overline{W},n\right)\leq c'\left(\sqrt{\log n}\log\log n\right)\overline{W}-c'\frac{2^{k-3}\overline{W}}{\sqrt{\log n}}+\frac{2^{k+2}\overline{W}}{c\sqrt{\log n}}.$$

In order to complete the proof, choose $c'\geq 32/c$.

$$C\left(\overline{W},n\right)\leq c'\left(\sqrt{\log n}\log\log n\right)\overline{W},$$

Now, we can choose $c'=\max\left\{32/c,2/\left(c|\log\left(1-\epsilon\right)|\right)\right\}$ to finish the proof of the lemma.

□

Since $\overline{W}$ is a lower bound on $(7/3)\,MCS\,(H)$, this establishes an approximation guarantee of at most $O\left(\sqrt{\log n}\log\log n\right)$ times the optimum.

**Algorithm complexity**

We already stated that the semidefinite relaxation can be solved to within an arbitrary additive error in polynomial time in the number of variables, as can the Cholesky decomposition into $n$ vectors. Notably, we only need to solve the relaxation once, and we use the same set of vectors $\{\mathbf{x}_0,\ldots,\mathbf{x}_{n-1}\}$ for all the recurrences within the algorithm. So, we only need to prove that the number of recursive calls to the algorithm is polynomially bounded. Let $T\left(n\right)$ be the number of recursive calls to the routine on an instance of $n$ vertices. We always branch into at least 2 strictly smaller subproblems. If we branch into more than 2, then we branch into $\kappa$ subproblems, where $\kappa\geq cn/\left(4\sqrt{\log n}\log\log n\right)\geq 2$, so we can always assume 2 subproblems instead. We can establish the following recurrence relation.

$$T\left(n\right)\leq 1+T\left(N_0\right)+T\left(N_1\right),$$

where $N_0,N_1\geq\left(1-\epsilon\right)n$. We use induction on $n$ to prove that $T\left(n\right)\leq 2n-1$. For the base case, we use $n\leq\lfloor 2/\epsilon\rfloor$. In this case, $T\left(n\right)=1$, since the algorithm returns a random order of the vertices without recursing on any components. Now, assume that the result is true for all $N \leq n$. We can use the induction hypothesis, since $N_0, N_1 \leq n$.

$$\begin{aligned} T(n) &\leq 1 + 2N_0 - 1 + 2N_1 - 1 \\ &= 2(N_0 + N_1) - 1 \\ &= 2n - 1, \end{aligned}$$

where the last equality follows from the partitioning of the hypergraph. This proves that the algorithm runs in polynomial time.

## 8.3 Spread

In this section, we prove the following theorem.

**Theorem 8.3.** *Given a hypergraph $H = (V, E)$ with non-negative edge weights $w\colon E \to \mathbb{R}$, there exists a polynomial time algorithm which outputs a vertex order that approximates $MCSP(H)$ to within a factor of $O\left(\sqrt{\log n}\log\log n\right)$ of the optimum.*

The proof of this theorem is relatively straightforward after we define the notion of a primal graph of a hypergraph.

**Definition 8.4.** Let $H = (V, E)$ be a hypergraph, and let $w\colon E \to \mathbb{R}^+$ be positive weights on the edges. The *primal graph* of $H$, denoted by $\mathcal{P}(H)$ is a graph on the set of vertices $V$ such that two vertices $u, v \in V$ are adjacent in $\mathcal{P}(H)$ if and only if there exists a hyperedge $e \in E$ in which both $u$ and $v$ are present.

The *weighted primal graph* of $H$ is the primal graph $\mathcal{P}(H)$ with a weight function $\omega$ on its edge set, defined as follows.

$$\omega(\{u, v\}) = \sum_{e \in E} w(e)\,\delta_{uv}(e),$$

where $\delta_{uv}(e) = 1$ if $u, v \in e$ and 0 otherwise.

We prove the following claim, which automatically gives the result of Theorem 8.3.

**Claim.** *The spread of a hypergraph $H = (V, E)$ with non-negative edge weights $w\colon E \to \mathbb{R}$ under a permutation $\pi\colon V \to \{1, \ldots, n\}$ is equal to the cost of the linear arrangement of its weighted primal graph $\mathcal{P}(H)$ under $\pi$.*

*Proof.* The objective function for the weighted minimum spread problem is given as follows.

$$\sum_{e \in E} w(e) \left[ \sum_{\{u,v\} \subseteq e} |\pi(u) - \pi(v)| \right],$$

where $\pi\colon V \to \{1, \ldots, n\}$ is a vertex ordering. Switching the order of the sums, we can instead express this as

$$\sum_{\{u,v\} \subseteq V} \omega\left(\{u, v\}\right) \left|\pi\left(u\right) - \pi\left(v\right)\right|,$$

where $\omega\left(u, v\right) = \sum_{e \in E, u,v \in e} w\left(e\right)$. The pairs of vertices which contribute to this sum will be those that occur together in at least one $e \in E$, which are exactly the edges of the weighted primal graph $\mathcal{P}\left(H\right)$. The contribution of the pair $\{u, v\}$, given by $\omega\left(\{u, v\}\right)$, is the weight of the edge $\{u, v\}$ in the weighted primal graph $\mathcal{P}\left(H\right)$.

By definition, the value of the rearranged objective function is the linear arrangement of $\mathcal{P}\left(H\right)$ under $\pi$. So, the linear arrangement of $\mathcal{P}\left(H\right)$ under $\pi$ has the same value as the spread of $H$ under $\pi$.

□

We can now run the approximation algorithms already known for the minimum linear arrangement [34, 78] on $\mathcal{P}\left(H\right)$ to solve the problem. This proves Theorem 8.3.

## 8.4 A note on the bandwidth generalizations

While we have given approximation algorithms for both the minimum cumulative stretch and minimum cumulative spread of a hypergraph, we have kept silent on how to approximate the minimum edge stretch and minimum edge spread. Feige proved that the approach we have taken for the algorithms in this chapter cannot yield good approximations for bandwidth problems on graphs [77]. The notion of *volume respecting embeddings*, used to get polylogarithmic approximations for the bandwidth problem [77] might prove a way around this. Alternately, there has been work done on generalizing the triangle inequality [28, 29]. This concept, combined with suitable restrictions, might prove a way beyond the approximation guarantees we are able to obtain.

# Part IV

# Applications: Software and Results

Chapter 9

# A package for ancestral genome map reconstruction

The theoretical problems we have discussed till now have important implications in genome mapping and scaffolding. Here, we will see how some of the results and methods discussed have been incorporated into software for ancestral genome mapping. The main topic in this chapter is the basic structure of software implemented for ancestral reconstruction. We introduce this through ANGES, a package implemented for reconstructing ancestral genome maps based on phylogenetic information [110]. Many of the basic techniques used in further chapters are implemented in this piece of software, and it serves as a good example to demonstrate comparative methods in ancestral reconstruction problems.

ANGES was joint work with Cedric Chauve, Bradley Jones and Eric Tannier. The data structures, the common intervals algorithm for permutations and the main C1P routines were coded by Bradley Jones, while the common intervals algorithm for sequences and the spectral heuristic were coded by Rajaraman.

## 9.1 ANGES: Ancestral genome mapping ignoring repeats

ANGES (Reconstructing Ancestral Genomes Maps) is a package written in Python which is designed to reconstruct ancestral genome maps given a set of non-overlapping ancestral genomic segment (markers), with their positions on the genomes of related extant species, and a species tree for the species of interest. The package is designed to handle maps with single occurrences of a marker, i.e. unique markers, though the input may contain extant genomes with multiple occurrences of a marker. The principle underlying the workings of the package are related to the C1P, and work by constructing a binary matrix and optimizing it to get a C1P matrix.

### 9.1.1 Input data

ANGES takes in a species tree as input, with the ancestor of interest marked in the tree. It also takes in a set of non-overlapping markers in the extant species present in the tree, with their positions on the extant genome marked. The markers need not be unique in the extant species, and may be missing in some of them.

Both the species tree and the set of markers are expected to be precomputed. It is important that the markers be non-overlapping in the extant genomes. Otherwise, it is difficult to determine the syntenies between markers, especially if we have some markers which are completely contained in another.

### 9.1.2 Inferring adjacencies and intervals

The input set of markers in ANGES can be filtered by various criteria. For each of these options, we may choose to filter on only the ingroup species as defined by the position of the ancestor in the phylogeny.

1. *Unique markers*: This option discards all markers that appear more than once in an extant genome. Thus, it filters repeats in the extant genomes.

2. *Universal markers*: This option discards markers that are not present in all extant genomes.

3. *No filter*: In this case, the set of markers is taken as given.

Once the filtering step is performed, adjacencies between markers are computed using a Dollo parsimony argument in ANGES.

**Definition 9.1.** [81] Consider a phylogeny over some evolving binary character, with the values of the character known at the leaves of the phylogeny. The *Dollo principle* states that if there are two leaves in which the character is present, then it is also present in all ancestors lying on the evolutionary path between these two leaves.

Thus, for any ancestor, if there exist two extant species which contain an adjacency, and the unique path in the species tree between these species passes through the ancestor, then the adjacency is present in the ancestor [43].

ANGES also provides a routine to compute *reliable adjacencies*. These are adjacencies which are present in all solutions for the double-cut and join median of 3 extant genomes, the extant genomes being defined such that the ancestor lies on the evolutionary path of at least 2 of them [38].

ANGES provides 2 different routines for recognizing unordered intervals of markers. In each case, the routine used computes *common intervals*, intervals that exist in two extant genomes on whose evolutionary path the ancestor lies.

1. If the markers have exactly one occurrence in every extant species (i.e. they are *unique* and *universal*), ANGES uses the common intervals algorithm of Bergeron et al. [18] to determine common intervals in a set of permutations. This can be done in linear time in the number of markers.

2. If the number of occurrences of the markers varies in the extant species, ANGES uses the algorithm of Schmidt and Stoye [188] to compute common intervals in sequences. This procedure takes quadratic time in the number of markers, and is a bottleneck for large scale computations.

The set of adjacencies and intervals computed describe *ancestral contiguous sets* (ACS), sets of ancestral markers that are inferred to be contiguous in the ancestral genome of interest. ANGES does not currently provide the capability to compute ordered intervals.

ANGES can take parameters which restrict the set of intervals computed to adjacencies, maximal intervals (common intervals which are not contained in any other interval), and strong intervals (common intervals which are either completely contained in or completely contain other intervals). The intervals are weighted using the phylogenetic weighting scheme of Ma et al. [135]. Alternately, ANGES can take a precomputed, preweighted set of adjacencies and intervals as input.

In the event that certain markers are missing from some of the extant genomes due to evolutionary events such as gene loss, ANGES implements methods to post-process ACS in order to account for missing markers.

### 9.1.3 Finding a C1P submatrix

The adjacencies and intervals are compiled to form a binary matrix, with the markers as columns. Since the evolutionary hypothesis is that the ACS should be consecutive in the ancestor of interest, ANGES checks if this matrix has a C1P variant, specified by the user depending on the structure of the ancestral genome. If it does have the C1P variant, it computes a PQ tree, or a related data structure [142], and outputs the set of $Q$ nodes attached to the root as a set of *contiguous ancestral regions* (CARs).

For most data, however, the obtained matrix does not have the C1P variant desired. In this case, ANGES provides a greedy heuristic that discards the lowest weight adjacency or interval until the remaining submatrix has the C1P variant. While not optimal in general, experiments on test data show that the number of rows of the matrix which need to be discarded is usually quite small. The PQ(PQC)-tree is then computed for the remaining submatrix, and the CARs are output accordingly.

Alternately, ANGES also implements a branch-and-bound algorithm that computes the maximum weight subset of ACS which satisfy the C1P variant. This algorithm takes exponential time in the worst case, but if the number of ACS to be discarded is very small, then this is reasonably efficient.

ANGES also computes a PQR-tree [142], which contains $R$-nodes that localize the parts of the matrix which lead to a breakdown in the C1P structure.

### 9.1.4 A spectral heuristic

As a final option, ANGES allows us to use the spectral algorithm of Atkins et al. [9] as a heuristic. This is particularly useful when there is uncertainty regarding the presence of a marker in an adjacency or interval. In such cases, the user can assign a confidence measure for the uncertain markers, and the usual adjacency-interval binary matrix is allowed to have entries in the interval $[0, 1]$. The resulting matrix, say $M$ is then used to compute a square matrix $A = M^T M$. The matrix $A$ is used as input to the spectral algorithm for the seriation problem. The output of the algorithm is a PQ tree, which, if the matrix $M$ has the C1P, is the correct PQ tree.

If $M$ does not have the C1P, then, through Vuokko's work [205], we know that we are minimizing the principal component of a norm related to the number of gaps in the matrix, so in some sense, these are permutations that keep colocalized markers 'close'. An interpretation in terms of the cumulative spread of the corresponding hypergraph would allow us to get a more concrete characterization of the output.

## 9.2 Applications and extensions

ANGES has been used to map ancestral grass genomes [156], and was also used recently used to map ancestral mosquito genomes, a project in which we participated [164]. We also used it as a platform for the development of more sophisticated tools for mapping ancient genomes.

The next two chapters cover the applications of variants of ANGES to different data sets. In Chapter 10, we introduce FPSAC, a software for scaffolding ancient contigs with repeats. The underlying framework for FPSAC was based on ANGES, and developed to include the computation of the markers, ordered intervals, and gap filling. FPSAC was used to scaffold the ancestral Black Death genome [177]. This builds upon the work of Bos et al., who provided an initial set of ancestral contigs to work with [25].

Chapter 11 discusses the results of ANGES while mapping the genomes of ancestral *Anopheles* mosquitoes [164]. This is followed by the introduction of FPMAG, an augmentation of ANGES using techniques from FPSAC. This allows us to reconstruct ancestral genome maps with repeated markers. We introduce preliminary results of FPMAG applied to the *Anopheles* data.

These new methods and software use the maximum matching algorithm of Maňuch et al. for optimizing over adjacencies [138], which returns a set of adjacencies that are realizable in a mixed genome model. Linear genome maps are reconstructed by, deleting the lowest weight adjacency from circular CARs.

# Chapter 10

# Scaffolding ancient contigs

One of the applications for the methods discussed in this document is in the field of palaeomicrobiology, which aims at analyzing ancient microorganisms, especially pathogens obtained from the remains of infected hosts [68, 72]. A major motivation for the field is to understand the evolution of pathogens and of their relation with their hosts [67, 207]. The genome sequence of ancient pathogens, thus, is of significant interest in palaeomicrobiology, and with modern sequencing technologies, getting DNA sequence data for these organisms has become considerably easier [25, 140, 168, 190, 191, 212].

In this chapter, we will discuss a pipeline for scaffolding contigs assembled from ancient reads. The pipeline, called FPSAC, uses phylogenetic information in order to compute and finish the ancestral genome sequence. This was work done in collaboration with Cedric Chauve and Eric Tannier.

## 10.1 Ancient DNA: challenges and solutions

The problem with ancient DNA sequences is that DNA decays fast, and so the reads obtained are usually of low-quality and very short. When these reads are assembled into contigs, there may be considerable fragmentation in the contig assembly, and a large part of the genome may be left uncovered. For example, in a recent study assembling the ancestral plague genome, over 2.3 million 'high quality' reads were obtained, with an average length of 55.53nt, and they were assembled into 2,134 contigs of size greater than 500nt [25]. This precludes a detailed genome-scale study of the evolution of the structural organization of the genome. Such organization may be related to the pathogeneticity of an ancient microorganism, so obtaining it is a key step in studying ancient pathogens.

In order to accurately scaffold contigs from ancient reads, one cannot use existing methodologies. These techniques rely on data such as mate-pair libraries with mixed insert sizes [12, 33, 66, 89, 180, 183], genome maps [127], or comparison with one or several closely related genomes [94, 118]. Due to the decay and fragmentation of ancient DNA molecules (whose length depends on many factors, but that can be as short as 300nt [72]), reads from ancient genomes are expected to be short, and genome maps or mate-pair libraries with long inserts are not available. With the lack of this data, and such extensive fragmentation, it is not possible to scaffold ancestral contigs, except perhaps for short DNA molecules such as plasmids. These may be sufficiently covered by the assembled contigs, which allows us to scaffold them into molecules.

The only option left is to use a comparative approach. This involves comparing the contigs with one or more closely related genome sequences or maps [106, 154, 181]. For an ancestral genome, however, comparison with a single reference genome, either a descendant or an outgroup, is likely to predict derived syntenic features as ancestral [181]. This becomes a problem when the genomes contain many repeats and are highly rearranged [54].

This is the specific problem we seek to address, by using phylogenetic information to scaffold ancient contigs. These methods were applied to scaffold the main chromosome of the ancestral Black Death genome, for which a draft assembly was provided by Bos et al. [25].

Available methods for ancestral reconstruction were not usable on the type of data we were considering: highly conserved sequences with significant repeat content. Some methods cannot reconstruct ancient genomes with repeats [135]. While software like DUPCAR [134] can indeed reconstruct ancient genomes with repeats, they also expect the gene trees of the repeated genes as input. With high sequence conservation being one of the major characteristics of the data, it is not possible to accurately infer such gene trees. Other software does take into account repeats, but reconstruction may be based on genome rearrangement models. In GapAdj, for example, the only duplication mechanism considered is whole genome duplication [88].

As a result, we developed a scaffolding pipeline called FPSAC which was tailored to our purpose. The next section discusses the working of this pipeline, and the features implemented in it.

## 10.2 Methods: FPSAC

FPSAC (Fast Phylogenetic Scaffolding of Ancient Contigs) is a Python-based pipeline that is used to scaffold ancestral contig sequences into chromosomal segments, using phylogenetic information. The scope of FPSAC takes into account repeated ancestral segments. The algorithms are based on the theory behind the consecutive ones property with multiplicity.

FPSAC takes a set of contigs for an ancestral species, a species tree containing this species and the complete DNA sequences of the extant species. FPSAC follows a generic scheme for ancestral genome reconstruction [43, 110, 135], a pipeline that can be roughly divided into four sequential parts.

1. *Defining homologous marker families.* We define a homologous marker family as

one or more contig segments (*ancestral markers*) and several non-overlapping extant genome segments (*extant markers*) that have a high pairwise alignment score along their whole length. Each such family is assigned a *multiplicity* which bounds the number of ancestral markers from the family in the ancestral genome of interest.

2. *Selecting putative ancestral adjacencies.* Two ancestral markers are said to form an ancestral adjacency if they are inferred to have occurred consecutive to each other on the genome of the ancestor of interest. Ancestral adjacencies are predicted using a Dollo parsimony principle, using the internal position of the ancestor in the phylogenetic tree. Adjacencies are weighted according to their phylogenetic conservation, and define a weighted adjacency graph.

3. *Computing the ancestral scaffolds.* The set of putative adjacencies may not be realizable in the mixed genome model while respecting the multiplicities of the markers. If so, we compute a maximum weight subset of the adjacencies that permits a valid genome structure. Next, in order to resolve ambiguities and repeated marker organization, we compute conserved intervals spanning repeats and use these to resolve marker ordering in a way similar to the use of mate-pairs to scaffold extant genomes.

4. *Finding ancestral gap sequences.* We estimate the length of the ancestral gap between the markers involved in ancestral adjacencies from the length of the gap between the corresponding extant adjacencies (extant gaps). We perform a multiple sequence alignment on the sequences of the extant gaps whose length agrees with the estimated ancestral gap length in order to reconstruct a putative ancestral gap sequence.

### 10.2.1 Defining homologous marker families

A marker family is composed of at least one ancestral contig, and several non-overlapping extant genomic segments that can be pairwise aligned with high similarity. To define them, we map the ancestral contigs onto the extant genomes using Megablast [5], such that an alignment of at least 100nt is found, which has a similarity of at least 95%. Megablast was used since we expect highly conserved sequences in the ancestral and extant genomes that we wish to study; for genomes with less sequence conservation, it may be more beneficial to use other tools, such as Blast [5]. As a result, we can obtain relatively long hits having high similarity.

The aligned segment defines a *significant hit* on both the ancestral contig and the extant genomes. Then, using an iterative segmentation procedure, we produce contig and extant genome segments such that the following two conditions are satisfied.

1. Contig segments must align over their whole length to extant genomes segments.

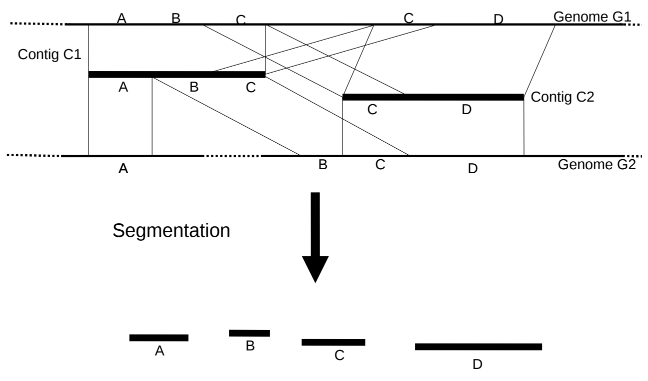


Figure 10.1: A schematic of the segmentation procedure used to obtain homologous marker families. For this example we consider two contigs $C1$ and $C2$ and their alignments on two genomes $G1$ and $G2$. Segment $C$ from $C1$ and $C2$ align to the same positions in both genomes, including two different positions on $G1$. Segments $A$ and $B$ of $C1$ align at two different positions of $G2$. After segmentation, we obtain four families with non-overlapping ancestral markers $A$, $B$, $C$ and $D$, which satisfy properties (1) and (2) given in the text. Note that family corresponding to segment $C$ contains two ancestral segments, from each of the contigs, two extant segments from $G1$ and one from $G2$. According to the number of occurrences in other genomes, this family may have a multiplicity $> 1$.

2. Contig segments must align to extant genome segments without two alignments overlapping.

If either of the two conditions is violated, the contigs or the corresponding extant genome segments are cut in order to get a full-length, non-overlapping alignment on the extant genomes.

Assume that a there is a segment $[a, b]$ of a contig with length $\ell$ which aligns to an extant genome, such that $a > 1$ or $b < \ell$, i.e. the alignment is not over the entire length of the contig. We can assume $a > 1$, since the other cases are similar. Then divide the contig into 2 segments, one being the segment on the interval $[1, a-1]$ of the contig, and the other being the segment on $[a, \ell]$. Divide the corresponding genome segments accordingly, and also cut all other alignments of this contig which overlap the coordinate $a$ into 2 segments. Apply this procedure till (1) is satisfied for all pairwise alignments, and no further segmentation can be carried out. This defines a new set of pairwise contig/genome alignments.

Now, condition (2) may be violated, since there may be two different contigs with segments aligning to two overlapping regions of an extant genome, say $[a, c]$ and $[b, d]$, with $a < b < c < d$. In this case, the two contigs are cut into two segments so that the four resulting segments align to genome segments with coordinates $[a, b]$, $[b, c]$ (for two of them) and $[c, d]$. This is shown in Figure 10.1. Iteratively apply the procedure until (2) is satisfied. This might lead to condition (1) being violated again. In order to make sure that the procedure converges, we remove short alignments (under 100nt, the minimum length of a significant hit) and repeat the two procedures until (1) and (2) are both satisfied. The set of all aligned sequences can then be clustered into sets of highly similar ancient and extant sequences, forming the homologous marker families.

Note that some ancestral contigs may not map to every extant genome. This may indicate potential evolutionary breakpoints.

**Marker multiplicities.** Once the marker families are defined, ancestral multiplicities, or copy numbers, are assigned to these families. These indicate the number of times an ancestral marker of the family is expected to occur in the ancestral genome. To do this, we compute the number of occurrences of extant markers in the family along the extant genomes, and minimize the number of evolutionary gain-loss events along the branches of the species tree. This is done using a linear time dynamic programming scheme [52].

### 10.2.2 Selecting putative ancestral adjacencies

In FPSAC, as in ANGES, ancestral adjacencies are inferred using a Dollo criterion on the phylogenetic tree [43]. The adjacencies are also weighted by the same procedure.

Each ancestral and extant marker is decomposed into two marker extremities, a head and a tail, in order to account for the orientation of markers in predicted ancestral syntenic features. This is a standard approach in genome rearrangement studies [38]. Adjacencies are defined in terms of marker extremities instead of the markers themselves, and are computed following a Dollo parsimony principle [43]: two ancestral marker extremities form an ancestral adjacency if they are contiguous (no other marker is between them in the chromosome) in at least two extant genomes whose evolutionary path in the species tree $T$ contains the ancestor.

Adjacencies are then weighted according to their patterns of phylogenetic conservation [43, 135]. The weighted adjacency graph is defined as follows: its vertices are the markers extremities and its edges are the weighted adjacencies.

#### Computing ancestral scaffolds.

We define an ancestral scaffold as a linear or circular order of the ancestral markers. It is possible that the set of putative ancestral adjacencies does not directly give us a set of ancestral scaffolds. This may be because (1) there does not exist a set of linear or circular marker orders in which each adjacency is represented, and which respects the multiplicity of each marker. It is also possible that (2) even if the adjacencies can be organized into ancestral scaffolds, there is more than one possible set of scaffolds which contain all adjacencies and still respect the multiplicities of the marker families. The latter case occurs due to the presence of repeats.

To address concern (1), we compute a maximum weight subset of ancestral adjacencies such that every marker extremity belongs to a number of adjacencies that is at most the multiplicity of the marker family [138,208]: for an ancestral marker of multiplicity $k$, each of its extremities can belong to at most $k$ ancestral adjacencies. It is possible to compute such a subset of ancestral adjacencies which is compatible with an order of the markers into a set of linear and/or circular scaffolds in polynomial time [138]. However, the algorithm does not allow us to control the resulting chromosomal structure, or the number of molecules obtained. Such a constraint would render the problem NP-hard [134]. But if the resulting set of adjacencies can be organized into a set of linear scaffolds, then this defines an optimal set of scaffolds.

As stated, though, the presence of markers with multiplicity greater than 1 may cause there to be multiple marker orders with which the optimized set of adjacencies is compatible. These markers cause tangles in the adjacency graph, leading to ambiguities regarding the structure of the ancestral genome.

In order to address concern (2), we infer *conserved intervals*, sequences of markers that span markers of multiplicity $> 1$. We first recognize *repeat clusters*, maximal connected components of markers having multiplicity $> 1$ in the adjacency graph. We define repeat spanning intervals on these repeat clusters by Definition 3.2. A repeat spanning interval is said to be *conserved* if it is compatible with two extant genomes in the species tree $T$, such that the evolutionary path between the two genomes in $T$ passes through the ancestor. These intervals are also weighted by a phylogenetic score, as in the case of adjacencies. We can compute all conserved intervals in time linear in the total size of all the repeat clusters.

Theorem 6.1 states that it is not possible to optimize over the set of conserved intervals in order to find a subset of intervals which is both compatible with the set of adjacencies retained in the previous step, and which satisfy the multiplicity constraints on the markers. Instead, we use Theorem 4.1 as a heuristic: for each repeat cluster $R$, we greedily select repeat spanning intervals that are compatible with the adjacencies selected during the previous step which containing markers of $R$ and satisfy the multiplicity constraints of the markers of $R$.

### 10.2.3 Finding ancestral gap sequences

A pair of adjacent marker families in the ancestral scaffolds, say X and Y, define an *ancestral gap*. In order to complete the scaffold, we need to fill in the gap sequence between all such pairs X-Y. To do this, we consider the extant genomes in which occurrences of X and Y are consecutive (no extant marker is between them) and in the same respective orientations as in the ancestor, thus defining an *extant gap* X-Y.

If we find the same adjacent marker families, say X and Y, forming an adjacency in an extant species, we find the *extant gap* corresponding to X-Y. A *conserved extant gap* is an extant gap whose length is equal in 2 extant genomes whose evolutionary path passes through the ancestor of interest, i.e. the length is conserved by a Dollo criterion. The minimum and maximum lengths of these extant gaps specify a *length interval* for the ancestral gap X-Y. In the absence of a conserved extant gap, the length interval of an ancestral gap X-Y is assigned to be the interval between the minimum and maximum lengths between X-Y in all extant species that the adjacency is present.

The ancestral gap sequence is estimated by performing a multiple sequence alignment between all the extant gap sequences which lie within the defined length interval of th ancestral gap. Then, we use a parsimony scheme to compute the ancestral gap sequence from the alignment obtained, using the Fitch algorithm [83]. We remove all gaps in the resulting sequence and assign this as the gap sequence for the ancestral gap X-Y.

## 10.3 Scaffolding the Black Death genome

FPSAC was used as a pipeline to scaffold the main chromosome of the ancestral Black Death agent [177]. We present this work in the current section.

### 10.3.1 History

The Black Death was a pandemic that occurred mostly in Europe, from 1346 to 1353. The pandemic is believed to be the cause of death of an estimated 75-200 million people in those years. Since this was before the germ theory of disease was proposed, and indeed, before even the tenets of hygiene had been lain down (something that was finally recognized only in the nineteenth century), the cause of the pandemic was unknown, and basic preventative measures could not be implemented.

A side effect of the lack of knowledge about the Black Death was that, until recently, the causative agent of the pandemic was not known. The contemporary explanation in the fourteenth century was that the pandemic was caused by 'bad air'. The prevailing medical theory was that it was a form of plague, caused by a variant of the bacterium *Yersinia pestis*. But solid genetic evidence to support this took a long time coming.

### 10.3.2 The East Smithfield study

In a 2011 paper, Bos et al. sequenced DNA extracted from the dental pulp from bodies in the East Smithfield burial ground in London [25]. This ground is believed to have been established for the burial of victims of the Black Death. They obtained over 10 million short reads, which were mapped to an extant strain of *Yersinia pestis* (CO92). Using Velvet [214], these were assembled into 2134 large ancestral contigs which occurred on the ancestral chromosome. This confirmed that the causative agent of the Black Death was a

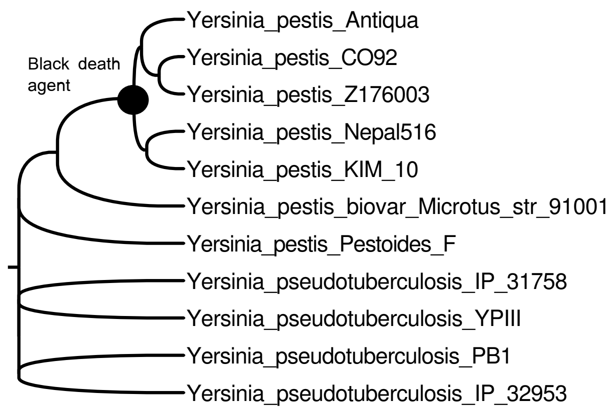

Black death agent
Yersinia_pestis_Antiqua
Yersinia_pestis_CO92
Yersinia_pestis_Z176003
Yersinia_pestis_Nepal516
Yersinia_pestis_KIM_10
Yersinia_pestis_biovar_Microtus_str_91001
Yersinia_pestis_Pestoides_F
Yersinia_pseudotuberculosis_IP_31758
Yersinia_pseudotuberculosis_YPIII
Yersinia_pseudotuberculosis_PB1
Yersinia_pseudotuberculosis_IP_32953

Figure 10.2: The *Yersinia* species tree. The Black Death agent is depicted as the bold, black node in the tree. It is conjectured to be ancestral to five *Y.pestis* species.

species of *Yersinia pestis* which is ancestral to most of the extant species they mapped the reads against (see Haensch et al. [101], which gives previous evidence for this).

The ancestral contigs obtained in the process, however, only covered around 85% of the expected length of the ancestral chromosome. Furthermore, the order of these contigs on the chromosome was unknown. The *Yersinia* genomes have well conserved sequences, but have high rearrangement rates.

### 10.3.3 Data

The input data consisted of 2,134 ancestral contigs assembled by Bos et al. [25]. Each of these contigs had length greater than 500nt, and they were estimated to cumulatively cover around 85% of the ancestral genome. We also had access to the DNA sequences of eleven fully assembled chromosomes of *Yersinia* genomes. Four of these genomes were *Yersinia pseudotuberculosis* strains, and seven were *Yersinia pestis* strains. Of the *Y.pestis* strains, five are conjectured to be descendants of the Black Death strain. The species tree, depicted in Figure 10.2, was inferred using SNP data [25, 150].

### 10.3.4 Inferred homologous marker families

Homologous marker families for the ancestor were identified by mapping the 2,134 ancestral contigs to 11 extant genomes in the species tree using the online version of Megablast [216].

Of the 2,134 contigs, 29 contigs did not map to any of the 11 genomes, leaving 2,105 ancestral contigs [25]. After recognizing significant hits and applying the pairwise trimming process, we were left with 2,616 homologous marker families.

**Ancestral multiplicities.** Most marker families were inferred to have copy number 1. Of the 21 remaining families, 20 had copy number 2 or 3. The remaining family was found to correspond to the 5S ribosomal protein family. Since it had a copy number of 8, and it was also short (133 nt), this family was discarded from the data.

Accounting for the copy number, the total length of the remaining families was 3,846,616 nt. The original set of contigs encoded 4,013,159 nt.

### 10.3.5 Selecting adjacencies and repeat spanning intervals

There were 2,634 parsimoniously inferred ancestral adjacencies. On running the optimization algorithm on this set, only 6 adjacencies needed to be discarded in order to obtain a maximum weight subset of adjacencies which could be realized into a set of linear and circular scaffolds.

There were 29 conserved repeat spanning intervals detected. Using a greedy heuristic, only 2 of them needed to be discarded to yield a subset which could be realized along with the set of adjacencies into an unambiguous set of linear and circular scaffolds.

### 10.3.6 Completing the chromosome

Using the adjacencies and the repeat spanning intervals, we obtained 3 large linear scaffolds, in which all ancestral families were present. Naively, there are 6 circular genomes that can be constructed from these scaffolds. To find the correct genome, we looked for extant adjacencies between marker families on the extremities of each linear scaffold. There was no such adjacency supported by the extant ingroup species, but 2 were supported by all outgroup species. This gave us a single linear scaffold. Finally, the adjacency between the extremities of this scaffold were supported by a single outgroup species (*Y.pestis Microtus*), and involved a marker which was absent from all *Y.pseudotuberculosis* species. This gave us the final circular chromosome, as described in Figure 10.3.

### 10.3.7 Filling gaps

Out of the 2,636 ancestral gaps, only 22 did not have a length interval supported by the Dollo parsimony criterion. Most of the gaps (2,561 of them) had a short length interval, of under 10 nt. The sequences obtained from the alignment and parsimony were used to fill the gaps, which filled the part of the chromosome between markers.

In order to fill the ancestral gaps, we aligned all extant gaps whose length fell in the ancestral gap length interval using MUSCLE [74] (version 3.8.31). We constructed the ancestral sequence from each alignment using Fitch's algorithm [83]. This resulted into a single sequence containing alternating sequenced ancestral contig segments and estimated ancestral gap sequences, illustrated in Fig. 10.3. The total computation time, including the Megablast alignments and the gap filling, took less than an hour on a dual core personal desktop machine.

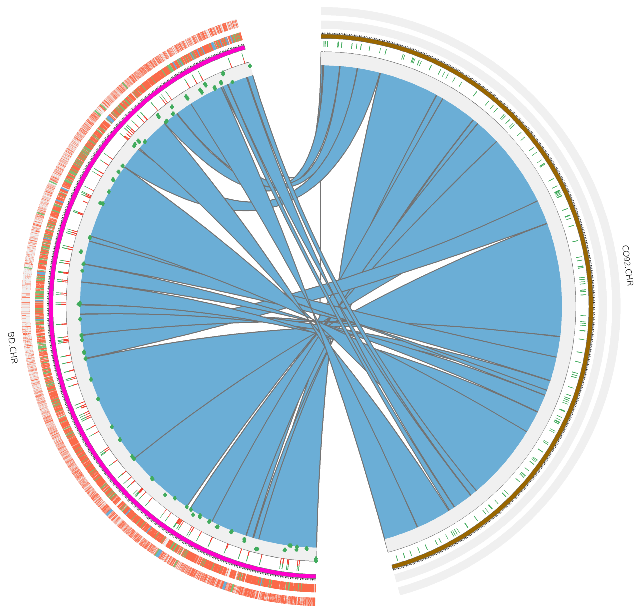


Figure 10.3: The reconstructed chromosome of the Black Death agent, on the left, compared against the chromosome of *Y. pestis CO92*. The ribbons connect collinear chromosomal segments. The outer track of the Black Death chromosome shows gap sizes, and the next track shows the lengths of the marker families which have contributed to these gaps, with red (respectively green, blue) indicating shorter (resp. mid-level, longer) sequence lengths. The first two inside tracks represent annotated (green) and inferred (red) insertion sequences. The inside track represents the level of breakpoint reuse between the ancestor and the following strains: *Y. pestis Antiqua*, *Y. pestis KIM10* and *Y. pestis Microtus str. 91001*. This image was created using Circos [124]

### 10.3.8 Validating the method

In order to validate the method, we ran FPSAC on 50 simulated datasets. Each simulation consisted of a randomly chosen extant genome from our data, which was designated as a proxy for the Black Death genome. This genome was evolved along the *Yersinia* phylogeny by performing a fixed number of random inversions along each branch. The number of inversions was chosen from the set $\{10, 20, 30, 40, 50\}$, and is an upper bound on the estimated number of rearrangements in the real data. The extant genomes obtained after this process are expected to be more scrambled than the extant genomes in the real data. Following this, 2,134 contigs were chosen along the simulated ancestral genome, having the same length distribution as the original set of contigs. Ten pairs were chosen from these, and merged to create chimeric contigs. The FPSAC pipeline was then applied to all 50 simulated data sets, with the parameters carried over from those used for the actual *Yersinia* data.

The resulting output from FPSAC consisted of 2,808.42 families on average over all 50 datasets, from which 130.64 have copy number greater than 1. The scaffolding process resulted in a single scaffold in all cases except for five, and the average number of scaffolds was 1.18. Only 2 adjacencies, and only 3 repeat spanning intervals were discarded over all 50 datasets. We found that 99.47% of the initial non-chimeric contigs appear in the reconstructed ancestral sequence with at least 95% identity over 95% of their length. FPSAC also reconstructed 98.66% of the gaps between consecutive ancestral contigs with the exact length as in the simulated ancestor. Finally, 99.14% of the simulated chimeric contigs were detected as chimeric.

These results are consistent with the high accuracy of reconstruction of ancestral gene orders from randomly rearranged extant genomes [135]. The significant information recovered by FPSAC is the high accuracy of ancestral gap reconstruction.

### 10.3.9 Contig correction

An occurrence of an ancestral marker in the scaffold corresponds to one or several segments of the initial contigs obtained by Bos et al. [25]. As such, one would expect to find the original contigs on the final scaffold. The ordering of the markers is mostly compatible with the initial contigs. Only one *chimeric* contig was found, split into two non-adjacent markers in the ancestral genome organization. There were four contig segments which had large subsequences (over 500 nt) duplicated in the ancestral genome, contradicting the initial assembly, which predicted that these segments only had 1 occurrence. Also, 63 contigs were found to have sequences that are actually contained in, up to small variations, within the sequence of another contig. The initial assembly predicted them to be contigs with copy number 1, so they seem to be either redundant, or they are derived from mutations of the ancestral genome.

### 10.3.10 Analyzing the ancestor

The final ancestral sequence we obtained was of length about 4.6Mb, which is roughly 600kb more than the total length of the ancestral contigs obtained by Bos et al. [214]. Note, however, that the total length of the marker families themselves is only around 3.8Mb. This means about 775kb were added during the process of gap sequence reconstruction. The marker families occurring on the ancestral scaffold corresponded to one or more fragments of the original 2,134 ancestral contigs, and the order of the families is mostly consistent with the order of these fragments. Only one chimeric contig was detected, which was split into 2 non-adjacent marker families in the scaffold.

Two of the six discarded adjacencies output by FPSAC provide evidence of a large-scale inversion. Since both the selected adjacencies and repeat spanning intervals, as well as the discarded ones, have similar phylogenetic support, the alternative genome structure, which takes into account this inversion, cannot be ruled out as non-ancestral.

We also studied the evolution of the main chromosome of the Black Death agent at a whole-genome scale. Insertion sequence (IS) elements have been suspected to be involved into the high rearrangement rate of *Y. pestis* genomes [32]. On mapping extant IS to the reconstructed ancestral chromosome, we obtained 92 ancestral gaps containing IS. This comparative annotation agreed with an automatic annotation of the reconstructed chromosome sequence. We observed that a large proportion of these IS (at least 58) were already present in older *Y. pestis* ancestral strains, but they are almost completely absent from the genomes of the *Y. pseudotuberculosis* represented in the species tree. This provides evidence that the *Y. pestis* speciation from its *Y. pseudotuberculosis* ancestor was characterized by a burst of IS insertions [32].

We analyzed the genome rearrangement distances between the reconstructed ancestral chromosome and the extant chromosomes by sampling inversion scenarios using DCJ2HP [148]. There are 8-9 inversions between the *Y. pseudotuberculosis* strains and the Black Death genome, and 9-22 inversions when compared to the *Y. pestis* strains. This shows an acceleration of evolutionary rearrangement following the *Y. pestis* diversification.

We also observed that inversion breakpoints are not randomly distributed and used: highly used ones are concentrated in one third of the chromosome, around its probable replication origin, as noted by Darling et al. [54]. The positions of the inversion breakpoints are highly correlated with IS, as remarked earlier [55]. Out of the 118 mapped breakpoints, 76 are close ($<$1,000 nt distant) to some predicted IS. This number drops to 39 for uniformly sampled random coordinates (p-value $<10^{-3}$). Rearrangements are numerous in all *Y. pestis* branches, strongly suggesting that they could be driven by the IS.

## 10.4 Subsequent work

Future work on the *Yersinia* phylogeny will focus on reconstructing the whole genomes of all ancestral species in the phylogenetic tree, including reconstructing plasmids. At this point, the reconstruction and scaffolding problem becomes more complicated, since the number of plasmids in the extant species varies between 1 and 3. While doing this, the choice of a mixed genome model becomes an asset, since we can get multiple linear and circular sequences of markers.

One of the avenues for FPSAC to expand into is the problem of trying to assemble contigs that come from a mixture of microbial backgrounds, for example from de novo assembly of reads obtained through shotgun sequencing. If we can classify the contigs into subsets that denote well identified clades, then theoretically FPSAC can be used on each of these subsets to obtain a set of ancestral scaffolds. This classification step poses the major obstacle to such analysis, since it may be confounded by problems such as repeated sequences belonging to several genomes [172]. In the event such challenges can be overcome, FPSAC may be extended to scaffolding ancient metagenomes.

# Chapter 11

# Reconstructing *Anopheles* genomes

This chapter describes an applied project which has been undertaken in collaboration with the *Anopheles* Genomes Cluster Consortium, the study of the genomes of 16 anopheles mosquitoes [163], known to be vectors for the malaria parasite *P. falciparum.* The full scope of this project was to sequence, map and analyze the given genomes in order to recognize traits that influence vectorial capacity and might provide useful hints to develop strategies to reduce the impact of malaria worldwide. The results of the project have been accepted for publication in Science [164].

## 11.1 Context and overview

We were part of the team that analyzed chromosomal evolution, and our work addressed three questions: (1) clearing some uncertainty in the phylogenetic subtree of a subset of the species called the Gambia complex, (2) reconstructing gene orders for several ancestral nodes of the *Anopheles* phylogeny, and (3) analyzing genome rearrangements along branches of the *Anopheles* phylogeny using the reconstructed ancestral gene order. This project was interesting to us, aside from its potential impact regarding an important health problem, because it was an opportunity to illustrate the usefulness of reconstructing ancestral gene orders in evolutionary studies.

However, the available data proved to be challenging in one important aspect, that is the high fragmentation of the assembled genomes. Due to the difficulty in resolving repeats in genome assembly and the large scale of the project, most assemblies were provided in the form of large sets of scaffolds or supercontigs. This prevented the reconstruction of ancestral gene orders at the resolution of full chromosomes; instead, the reconstructed CARs (Contiguous Ancestral Regions) were themselves fragmented, basically providing ancestral genomes at the same resolution as the extant ones. Since the inception of the current high-throughput sequencing technologies, that provide quick and reasonably priced genome sequence data at the expense of poor assemblies, it has been assumed in the genome rearrangement community that starting from fragmented assemblies would likely prevent the study of evolution in terms of large-scale rearrangements.

Our effort with the *Anopheles* data showed that while it is true that we could not provide whole-genome evolutionary scenarios, we were able to highlight important evolutionary facts in terms of genome rearrangements, such as an apparently higher rearrangement rate in the X chromosome of *Anopheles* genomes than in autosomes (non-X chromosomes), thus showing the reward was worth the effort.

In this chapter, we present our results, that were integrated in the flagship paper of the *Anopheles* project [164], which were obtained using ANGES. Furthermore, we will describe a new pipeline for ancestral genome mapping, which augments the existing ANGES software. The main advantage of this pipeline is the ability to account for, and output, genome maps with repeated markers. This pipeline, which we call FPMAG, is to be an extension of the next major ANGES update, and is based on combining the principles of FPSAC to handle repeats into ANGES. Here, we provide the details of the process, and the preliminary results on the *Anopheles* genome data. We contrast the results obtained using FPMAG with the original results obtained using ANGES.

## 11.2 Data: 16 *Anopheles* genomes

The input data consisted of gene families of orthologous groups delineated at the Dipteran level from the OrthoDBmoz2 database of orthologous groups in mosquitoes [206], and GFF gene files for the following 11 species: *An. gambiae*, *An. arabiensis*, *An. quadriannulatus*, *An. merus*, *An. stephensi*, *An. minimus*, *An. funestus*, *An. dirus*, *An. farauti*, *An. atroparvus* and *An. albimanus*. These species were chosen on the basis of the quality of their assemblies as given in Neafsey et al. [163] and after discussions with specialists of *Anopheles* genomes involved in the project. These files were processed to extract gene families that occurred exactly once in each extant genome. The total number of gene families thus defined was 5,343.

Anopheles genomes are formed of 5 main elements, called chromosome arms, which in turn form 3 chromosomes: one sex chromosome (the X chromosome) containing a single chromosome, arm and of two autosomes, each containing two arms. However, the arrangements of the arms of the autosomes vary among the different *Anopheles* species. Table 11.1 indicates the level of fragmentation of the assemblies when limited to scaffolds and contigs containing these 5,343 genes, that ranges from 6 segments for *An. gambiae* (a fully assembled genome) to 826 for *An. merus*. In the case of *An. gambiae*, the assembled genome consisted of 5 chromosomal arms, and 45 genes in unknown locations.

The species tree we used for the analysis defined over the 11 species is given in Figure 11.1.

| Species name | Number of arms/contigs/scaffolds | Number of genes |
|---|---|---|
| *An. gambiae* | 6 | 7,645 |
| *An. arabiensis* | 153 | 8,258 |
| *An. quadriannulatus* | 369 | 8,244 |
| *An. merus* | 826 | 7,771 |
| *An. stephensi* | 389 | 8,217 |
| *An. minimus* | 48 | 8,145 |
| *An. funestus* | 443 | 8,148 |
| *An. dirus* | 141 | 11,508 |
| *An. farauti* | 296 | 7,798 |
| *An. atroparvus* | 201 | 7,643 |
| *An. albimanus* | 44 | 7,544 |

Table 11.1: Details of the 11 extant anopheles genomes. The number of genes is given with multiplicity.

## 11.3 Confirmation of the species tree

The topology of the tree at the ancestor G3 marked in Figure 11.1 (the *An. gambiae* complex) is still subject to some uncertainty regarding the location of *An. arabiensis* and *An. quadriannulatus* (see [163, Fig. 1]).

To validate the chosen topology of the *An. gambiae* complex, we considered the subset of 446 one-to-one orthologous gene families which occurred in *An. gambiae* on the X chromosome, since X chromosomes are known to be subject to less introgression [85]. We noted that if *An. gambiae* and *An. arabiensis* form a clade, we get 20 syntenic conflicts during the assembly procedure into CARs. On the other hand, if *An. quadriannulatus* and *An. arabiensis* form a clade, only 4 syntenic conflicts needed to be discarded to get a set of linear CARs, which supported the hypothesis stated by Fontaine et al. [85] regarding the accuracy of this clade. This is also supported by the fact that with this topology, the number of discarded adjacencies and intervals to clear syntenic conflicts on the X chromosome dataset is upper-bounded by 10 at each ancestral node, thus showing a low level of syntenic conflict. In the case of deep ancestors, though, the increasing fragmentation, even in the X chromosome dataset, shows a decay in the syntenic signal. This may be a result of the fact that most extant genomes were fragmented into a large number of chromosomal segments, except for *An. gambiae*, and to a lesser extent *An. albimanus* and *An. minimus.* Thus, we would expect fewer conserved syntenies in deeper ancestors, which complicates reconstructing a map with low fragmentation.

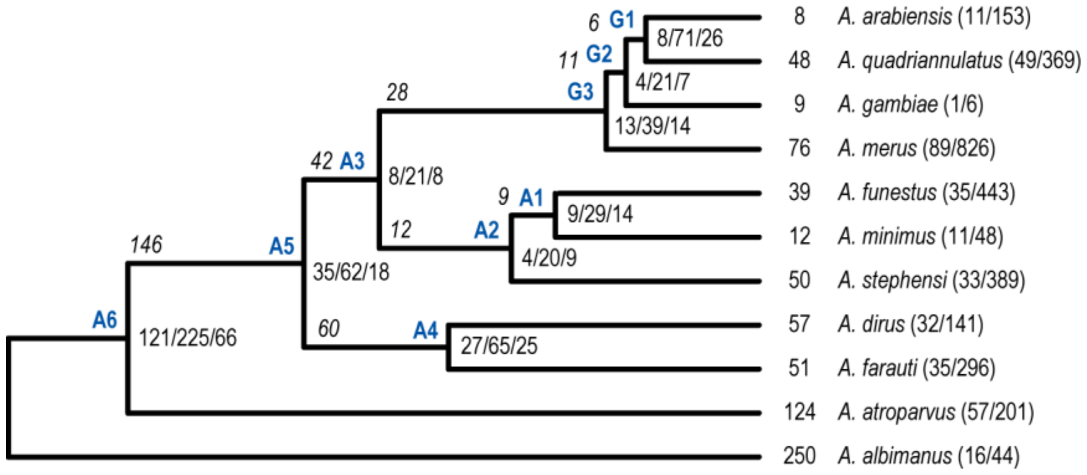


Figure 11.1: The *Anopheles* species tree, annotated with the results from ANGES. The numbers at the extant species indicate the one-to-one orthologous families in the X chromosome dataset (left annotation), and the whole genome (right annotation). The labels on the internal nodes indicate the number of CARS on the X chromosome (left), the number of CARS in the whole genome (middle), and the number of whole-genome CARS which cover 90% of the one-to-one orthologous families (right), each as predicted by ANGES. The numbers on the branches indicate the DCJ distance between two sets of X chromosome CARS for the genomes at the extremities of that branch, obtained using UniMOG [105].

## 11.4 Reconstructed ancestral gene order with ANGES

ANGES was used to reconstruct the ancestral genome maps for all internal nodes of the tree, except for the root, that appeared to be too fragmented from preliminary experiments. Markers were doubled to account for gene orientation. The computed conserved syntenies were composed of oriented gene adjacencies and strong common intervals using Dollo parsimony, as well as gene adjacencies which were present in three extant genomes such that the ancestor lies on the evolutionary path of at least two of them [38]. The locally conserved syntenies were assembled into linear CARs using the greedy heuristic option of ANGES.

The results obtained by ANGES are summarized in Figure 11.1, which presents the data on the whole genome as well as the X chromosome of the respective species.

The end result was still encouraging, and the data allowed the reconstruction of a relatively well-defined sets of CARs. Fewer than 1% of the detected syntenies were discarded during the assembly phase to obtain a set of linear CARs. In the worst case, an ancestor was predicted to have 225 ancestral CARs.

To evaluate the impact of the fragmentation on the ability of addressing whole genome evolutionary questions, we introduced a measure similar to the popular N50 measure in genome assemblies: we define the NCARX as the number of CARs that contains at least X% of the input (here 5,343) genes. We could observe that, for most ancestors, the NCAR50 and NCAR90 were relatively low: in general, less than 50% of the predicted number of CARs at any ancestor were needed to cover 90% of the unique orthologous families. This suggests that a few CARs capture the evolution of most of the anopheline genomes, except for the ancestor of all anophelines and *An. albimanus*. For this extant species, the gene order appears to be less conserved.

Finally, as an accuracy check, we could observe that for a very large majority of the ancestors, CARs were limited to genes from a single *An. gambiae* arm. This shows while that it is not possible to reconstruct the chromosomal arm structure of the ancestral genomes, it is possible to reconstruct chromosomal segments at the current fragmentation level of the extant genomes used for the reconstruction.

## 11.5 Genome rearrangements in the Anophele phylogeny

Preliminary experiments showed that the chromosomal evolution of the *Anopheles* genomes seemed to display an interesting phenomenon: the rate of rearrangements in the X chromosome seemed to be much higher than the rate of rearrangements in the autosomes. The ratio of rearrangement rates in the X chromosome was assessed by other members in the chromosomal evolution group to be 2.7 times higher than the rate of genome rearrangements in the autosomes, a striking difference to similar insects such as *Drosophila* where the ratio was only of 1.2 [164].

However the fragmentation of the assembly was seen as a possible source of bias in inferring genome rearrangements, due to possible genome breakpoints being attributed to assembly discontinuity and rather than being actual evolutionary breaks. To account for this effect, we computed the single cut-or-join distances [79] from the ancestral genomes computed by ANGES to the extant genomes, and corrected these distances for potential adjacency breaks due to assembly fragmentation as described below. These experiments were meant as an independent confirmation of the rearrangement model used for reconstruction by others in the group, as compared to the model-free approach taken by ANGES.

### Analysis

We reconstructed the genomes for all ancestral species using the same gene family data as before, but with syntenies restricted to supporting adjacencies between the markers. The adjacencies in each ancestral and extant genome were classified according to the location of the orthologous markers on *An. gambiae* orthologs: adjacencies for which both genes have have orthologous markers on the *An. gambiae* X chromosome (respectively the autosomes) are called X-adjacencies (resp. A-adjacencies), while the remaining adjacencies, whose location as X chromosome adjacencies or autosome adjacencies is unclear, are called AX-adjacencies.

Then, for each branch of the phylogeny tree, for a pair of genomes A and B, we used a variant of the Single-Cut-and-Join (SCJ) distance which is aimed at correcting potential false positive genome rearrangements induced by the fragmentation of assemblies or ancestral genome reconstruction [20]. Formally, the SCJ distance between two genomes is defined as the number of adjacencies present in exactly one of the two genomes. The correction checks if an adjacency (x,y) is present in exactly one of the two genomes, and if the gene extremities defining it are both scaffold or CAR extremities in the other genome, then we label it as a "potential false positive SCJ", and do not consider it in the distance calculation.

We considered all pairs of genomes (A5,E) where E is an extant genome and A5 is the ancestor labelled as such in Figure 11.1. We chose ancestor A5 as the oldest ancestor with an acceptable level of fragmentation. For each such pair, we computed a proxy of the rate of autosomal genome rearrangements, given by the ratio of the corrected SCJ distance for A-adjacencies by the number of considered genes on the *An. gambiae* autosomes. There were 4,877 such genes on the autosomes. Similarly, we computed the same proxy for with the potential X chromosome genome rearrangement using the 466 genes on the *An. gambiae* X-chromosome.

### Results and discussion

The total number of A-adjacencies (respectively X-adjacencies, XA-adjacencies) found was 91,991 (respectively 8,595 and 252). The small number of XA-adjacencies justifies discarding them from further studies. The complete results are summarized in Table 11.2.

The SCJ distance computed is a simplified rearrangement model, which makes many problems concerning rearrangement distances tractable. However, the simplicity of the model also means that any inference from the results obtained should take this into account. The rearrangement distances and rates shown in Table 11.2 are likely to be higher than in reality. Also, the definition of potential false positive SCJ might reduce the real SCJ distance. But the corresponding biases must also be accounted for in the X chromosome and the autosomes. In order to compare the rearrangement rates between the autosomes and the X chromosomes, we take the ratio of the rates obtained, which we expect to be a good estimation for the real ratio after such correction is made. The ratio shows a trend of a higher rearrangement rate in the X chromosome than in the autosomes. This agrees with data obtained in the study of chromosomal evolution in extant species *An. gambiae*, *An. stephensi*, *An. funestus*, *An. atroparvus*, and *An. albimanus* using physically mapped scaffolds [164].

We also observed lineage specificity in rates of chromosomal evolution. Since all studied extant species are at equal distances from the common ancestor A5, we concluded that *An. dirus* and *An. farauti* belong to the lineages with the most rapidly evolving chromosomes, while the lineages that lead to *An. minimus* and *An. merus* have a relatively slow speed of

| Genome paired with A5 | Autosome SCJ distance (corrected) | X chrom. SCJ distance (corrected) | Rate in autosomes | Rate in X chrom. | Rate in all chrom. | Rate ratio Xchrom. /autosomes |
|---|---|---|---|---|---|---|
| An. farauti | 282 | 85 | 0.058 | 0.182 | 0.24 | 3.154 |
| An. dirus | 263 | 98 | 0.054 | 0.210 | 0.264 | 3.899 |
| An. stephensi | 201 | 65 | 0.041 | 0.139 | 0.18 | 3.384 |
| An.minimus | 207 | 34 | 0.042 | 0.073 | 0.115 | 1.719 |
| An. funes-tus | 229 | 50 | 0.047 | 0.107 | 0.154 | 2.285 |
| An. merus | 200 | 45 | 0.041 | 0.096 | 0.137 | 2.355 |
| An. gam-biae | 220 | 47 | 0.045 | 0.101 | 0.146 | 2.236 |
| An. quadria-nnulatus | 205 | 63 | 0.042 | 0.135 | 0.177 | 3.216 |
| An. arabi-ensis | 216 | 60 | 0.044 | 0.128 | 0.172 | 2.91 |

Table 11.2: Rates of chromosome evolution between ancestral and extant *Anopheles* species. The rearrangement rates supported those predicted by other members in the chromosomal evolution group of the project [164].

rearrangements.

## 11.6 FPMAG: Using FPSAC techniques in ANGES

A new technique for genome mapping in the presence of repeats, which was proposed for this project, and remained unused since we were working with unique markers, was to use the methods implemented in FPSAC for genome mapping. These methods, compiled into a series of scripts jointly called FPMAG (Fast Phylogenetic Mapping of Ancient Genomes), can be stated as a 2-step process.

1. In the first step, we use adjacencies to find a genome map using just the markers having copy number 1 (*unique markers*) in each of the extant species (*universal markers*). We may also use unique markers, while allowing them to be missing in some of the extant genomes.

2. In the second step, we separately find a map of the region between two unique markers on the previous map, using techniques borrowed from FPSAC. Since all unique, universal markers were already considered, the set of markers in this maps will consist only markers which are repeated in some of the extant genomes.

### 11.6.1 Identifying marker families and copy numbers

In this stage, the process used is very much the same as FPSAC. Given a set of possibly overlapping ancestral gene families, an iterative truncating and merging procedure is used to define non-overlapping families above a certain threshold length. The copy number of these families on the extant genomes is defined as the number of times they appear on the extant genome map. The copy number of the families in the ancestral species is defined through a parsimony scheme [52], as in the case of FPSAC.

In the *Anopheles* data, 20,564 predefined families and their positions on the extant genomes were given, of which 5,343 occurred in every extant species exactly once. We assumed a copy number threshold of 5, and discarded all families having higher copy numbers. Alternately, we can retain only the unique marker families, or unique marker families that are present in all extant genomes descended from the ancestor under consideration.

### 11.6.2 Obtaining a genome map

We can use FPSAC's methods for inferring and optimizing adjacencies to find a set of syntenies that admit a realization in the mixed genome model. The set of inferred adjacencies using the Dollo criterion is the same as the supported adjacencies inferred by ANGES, and they are weighted by the same method. This does not include *reliable adjacencies*, adjacencies which are present in 3 extant genomes such that the ancestor lies on the evolutionary path between at least two of them. After obtaining the adjacencies, they are optimized by the maximum matching algorithm [138], and we use the set of retained adjacencies to construct a set of CARs, and hence, a genome map. However, this genome map does not contain any repeated markers.

The genomes of the *Anopheles* ancestors are linear, but the optimization algorithm finds a set of adjacencies which are realizable in the mixed genome model. Therefore, we have to linearize any circular CARs that we obtain. In order to do this, for every circular CAR, we discard the lowest weighted adjacency and make it linear.

The comparison of the number of CARs obtained by this methods to that obtained by ANGES using the same set of unoptimized adjacencies, and by ANGES using unoptimized adjacencies and strong intervals on the *Anopheles* data is illustrated in Table 11.3. The map, in the case of both ANGES and FPMAG, was constructed using only unique and universal markers. We can see that the number of CARs obtained is often significantly reduced by using FPMAG, as long as we are only using adjacencies to assemble the map. On including strong intervals in ANGES, we may obtain better results using ANGES, since many CARs may be scaffolded together by the intervals. The key factor to preferring FPMAG over ANGES, however, is that it uses an exact optimization routine, and it lets us incorporate the next step, in which we include repeated markers in the map.

| Species | Adjacencies found | ANGES (with adjacencies) | ANGES (with adjacencies, strong intervals) | FPMAG |
|---|---|---|---|---|
| ADIRW, AFARF | 5,289 | 104/50 | 4/54 | 95/50 |
| AFUNF, AMINM | 5,321 | 39/17 | 34/17 | 36/17 |
| AFUNF, AMINM, ASTES | 5,338 | 31/26 | 27/26 | 28/25 |
| AQUAS, AARAD | 5,273 | 76/6 | 75/6 | 49/6 |
| AQUAS, AARAD, AGAMP | 5,349 | 33/39 | 23/42 | 20/39 |
| AQUAS, AARAD, AGAMP, AMERM | 5,320 | 65/42 | 50/42 | 49/42 |
| AQUAS, AARAD, AGAMP, AMERM, AFUNF, AMINM, ASTES | 5,356 | 41/54 | 31/55 | 36/53 |
| AQUAS, AARAD, AGAMP, AMERM, AFUNF, AMINM, ASTES, ADIRW, AFARF | 5,291 | 115/63 | 83/64 | 96/62 |
| AQUAS, AARAD, AGAMP, AMERM, AFUNF, AMINM, ASTES, ADIRW, AFARF, AATRE | 5,060 | 321/38 | 277/40 | 226/35 |

Table 11.3: Comparison of ANGES with adjacencies, ANGES with adjacencies and intervals, and FPMAG on the ancestral *Anopheles* data, using unique and universal markers. The ancestors are labelled by their extant descendants, with the key as follows: *A. arabiensis* (AARAD), *A. quadriannulatus* (AQUAS), *A. gambiae* (AGAMP), *A. merus* (AMERM), *A. funestus* (AFUNF), *A. minimus* (AMINM), *A. stephensi* (ASTES), *A. dirus* (ADIRW), *A. farauti* (AFARF), *A. atroparvus* (AATRE). In the other columns, the first number is the number of CARs found in the map, while the second number shows the number of discarded adjacencies and intervals.

### 11.6.3 Identifying repeated markers between unique markers

Up to now, the only difference in the ANGES and FPMAG pipelines has been the optimization criterion for the set of adjacencies/intervals. In ANGES, it is possible to infer intervals, and the optimization algorithm used to find a set of CARs is a heuristic which discards adjacencies and intervals by their weight. In FPSAC, the only intervals which we can infer are repeat spanning intervals, but due to the absence of repeats in the previous stage, no such interval will be found. The optimization step is again based on the maximum matching argument of Maňuch et al. [138].

In the next stage, FPMAG treats each adjacent pair of unique markers on the ancestral map as the telomeres of a chromosome. Then, for all extant species which contain this adjacency on filtering for repeats, it finds repeated gene families that occurred between this adjacent pair of unique markers. Thus, we get a set of repeated markers and the two unique markers that frame them, forming a repeat cluster and frontier vertices respectively. This defines a new instance for each adjacent pair of unique markers on the map obtained at the end of the first stage of FPMAG.

On each such instance, we can run FPMAG again, this time without discarding repeats, since there are only 2 unique markers. By doing so, for each adjacent pair of unique markers on the original ancestral map obtained, we can get a *submap*, which is either empty, i.e. there were no repeats mapped between the two unique markers, or we get a set of CARS consisting of only repeats which fills the space between the unique markers.

### 11.6.4 Preliminary results

The results obtained by running FPMAG for the *Anopheles* ancestors is summarized in Table 11.4. In each case, at least 80% of the expected number of markers were recovered on the reconstructed ancestral map. The NCAR50 was always under 20% of the total number of CARs, even for the oldest ancestor. At most a total of around 15% of the adjacencies inferred within the subproblems between unique markers had to be discarded, which is higher than the usual expected value discarded for optimization. For repeat spanning intervals, as many as 40% of the inferred intervals in the subproblems had to be discarded.

## 11.7 Observations

The conclusion reached from the preliminary experiments using FPMAG is that the current iteration of the method does not immediately offer massive benefits over ANGES, other than being able to incorporate repeated markers. ANGES has the added advantage of being able to use intervals to piece together larger CARs, which means that the obtained genome map may be far less fragmented than the one obtained by FPMAG.

This does not mean that the development on the FPMAG front has stopped. Since

| Species | Expected genes | Genes found | Largest CAR | NCAR50/ CARs | NCAR90/ CARs | Adj. disc. (%) | RSIs disc. (%) |
|---|---|---|---|---|---|---|---|
| ADIRW, AFARF | 12,020 | 10,327 | 743 | 288/12 | 56/44 | 8.76 | 27.60 |
| AFUNF, AMINM | 11,644 | 10,521 | 1,949 | 519/5 | 158/20 | 5.67 | 15.52 |
| AFUNF, AMINM, ASTES | 11,764 | 10,518 | 1,400 | 790/5 | 173/15 | 8.708 | 21.67 |
| AQUAS, AARAD | 11,983 | 10,929 | 1,011 | 447/8 | 122/26 | 4.36 | 15.85 |
| AQUAS, AARAD, AGAMP | 12,437 | 11,006 | 2,312 | 1,274/3 | 460/10 | 10.60 | 32.28 |
| AQUAS, AARAD, AGAMP, AMERM | 12,461 | 10,523 | 1,662 | 453/6 | 130/22 | 13.83 | 37.33 |
| AQUAS, AARAD, AGAMP, AMERM, AFUNF, AMINM, ASTES | 12,263 | 10,428 | 1,472 | 650/6 | 166/36 | 15.27 | 40.34 |
| AQUAS, AARAD, AGAMP, AMERM, AFUNF, AMINM, ASTES, ADIRW, AFARF | 12,371 | 10,378 | 819 | 264/12 | 59/42 | 14.61 | 39.56 |
| AQUAS, AARAD, AGAMP, AMERM, AFUNF, AMINM, ASTES, ADIRW, AFARF, AATRE | 12,111 | 9,728 | 517 | 127/25 | 19/100 | 9.60 | 27.68 |

Table 11.4: Summary of FPMAG results on the *Anopheles* ancestors. The NCAR50 (NCAR90) columns give the size of the CAR needed to achieve 50% (90%) coverage of the total number of genes expected in the ancestral genome. The second number in the same column indicates the number of CARs needed to achieve this coverage. The last 2 columns is the number of adjacencies and repeat spanning intervals discarded in the sub-problems between unique markers, respectively.

the main obstacle to using FPMAG is the higher level of fragmentation, one logical way to overcome this problem is to use the methods in ANGES in order to get a less fragmented genome map on unique markers. Since we are using unique markers in the first step of FPMAG anyway, this may be one way to resolve the issue. The other idea is to use FPMAG to get a fragmented set of CARs, and then include intervals of unique markers, computed by ANGES. These intervals can be used to scaffold together the fragmented CARs. Since the sole intervals we are interested in are those that contain the extremities of the CARs computed by FPMAG, we can compute such intervals efficiently.

The other major problem see from the FPMAG results is the large number of discarded adjacencies and repeat spanning intervals when we add repeated markers in gaps between unique markers. There could be many reasons for this, with one possible source of error being the copy number estimation using Fitch parsimony. There are alternative approaches to copy number inference for ancestral markers [52, 102]. It is not known if the results obtained by using such approaches can significantly reduce the error.

Error may also arise from the fact that the gaps are often inferred to have very few repeated markers. The repeat cluster in each gap over all species had at most 5 repeated markers. This may point to two possible sources of error. Since the cluster is so small, it is possible that certain markers are missing from the repeat cluster. Thus, certain adjacencies and repeat spanning intervals may not be inferred.

Alternately, this may be a signal of introgression, and markers from a species may be breaking up the synteny in another. In either case, it is worth considering if it is better to use an explicit rearrangement model for small gaps. Ideally, this model should take into consideration insertions and deletions in order to account for missing or extra markers, which would account for the small number of syntenies discovered in each of these gaps.

# Part V

# Conclusion

# Chapter 12

# Conclusion

We will conclude this dissertation by summarizing the main contributions that have been described in the previous chapters, and a list of open questions and projects.

## 12.1 Summary of contributions

### 12.1.1 Reconstructing genome maps with repeats

The first section dealt solely with the problem of reconstructing genome maps in the presence of repeated markers. This is a well studied problem, and the methods we introduce in order to establish repeat order and resolve ambiguities form a theoretical basis for many heuristic methods already in present in some assemblers, which use long reads to resolve repeats [214], or resolve small repeats [209]. A theoretical framework for the same makes sense from the point of view of extending the approach to different problems. The framework we use is a generalization of the consecutive ones property [15, 87], which has long been used for reconstructing physical maps. The generalization incorporates data to represent repeated genomic markers, and there has been a slew of recent work done on it [138, 208].

An important theoretical concept we introduce into this framework, which seeks to capture the essential information in long range synteny information spanning repeats, is that of repeat spanning intervals. These are pieces of synteny information which encode the order of a set of repeats which are framed by unique marker elements on a genome. Repeat spanning intervals represent the internal organization of clusters of repeats on the genome, and the unique markers framing the repeats provide information to resolve ambiguities while constructing the map. The results we presented exploit the structure of the repeat spanning intervals to obtain decision and optimization results.

The two main positive results we have to take back are the following.

1. Theorem 4.1 states, in brief, that if we are given adjacency information concerning repeats, or the only long range synteny information containing repeats is in the form

of repeat spanning intervals, it is possible to determine if the data is consistent with a viable marker order on a genome map having any given structure.

2. In Theorem 5.1, we are given a list of adjacent markers, including repeats, that can be realized as a genome having both linear and circular chromosomes. Then, given a list of short repeat spanning intervals, which contain exactly 1 repeat occurrence, we can find the largest set of such intervals which is consistent with a genome map that also contains all adjacencies.

The main negative result is a counterpoint to Theorem 5.1. The result established by Theorem 6.1 states that if the synteny information available to us contains more than one occurrence of a repeat, then even if we know that the underlying set of adjacencies admits a genome map with linear and circular chromosomes, it is generally computationally hard to find the largest set of repeat spanning intervals which is still consistent with a genome map in this model which contains all adjacencies.

In addition, we presented two theoretical fixed parameter tractability results. Theorem 4.2 states that if the set of repeats and the expected number of occurrences of the repeats is small, then, if every piece of synteny information only involves a bounded number of markers, it is possible to determine if the data is consistent with a genome map in any given model. Theorem 6.2 states that if the number of repeats and their expected number of occurrences is small, it is possible to optimize on the set of repeat spanning intervals to get a consistent set.

### 12.1.2 Determining genome organization through vertex ordering

The second set of theoretical results is related to vertex ordering problems on hypergraphs. We generalized the problem of determining the minimum linear arrangement to hypergraphs. The motivation behind these problems was the possibility of being able to detect synteny information with small errors in linear genomes without repeats. Such inferred syntenies would be usually classified as chimeric by a heuristic while assembling the synteny data into a viable linear genome. However, discarding such data could lead to the loss of potentially useful information during the mapping process.

In this section, we presented two approximation results for the generalizations discussed. Theorem 8.1 and Theorem 8.3 presented $O\left(\sqrt{\log n}\log\log n\right)$ approximations for both the presented generalizations, where $n$ is the number of vertices in the hypergraph. These are based on the corresponding approximation algorithm for the minimum linear arrangement problem in graphs [34, 78].

### 12.1.3 Implementing and using the methods

The final contribution presented in the document is the implementation of the methods discussed and developed in it, and using them to analyze real genomic data. There are two major pieces of software we have presented here.

**ANGES.** ANGES is a software for reconstructing ancestral genome maps using phylogenetic information and the framework of the classical consecutive ones property [110]. It uses combinatorial algorithms for detecting conserved syntenies on extant genomes, and uses algorithms for detecting the consecutive ones property [142] in order to implement heuristics that output a linear or circular genomic marker order. It also incorporates a spectral algorithm for the consecutive ones property which serves as a heuristic for producing a similar order while retaining the detected synteny information [9].

ANGES was used to find the ancestral genome maps for ancestors in the phylogeny of the *Anopheles* mosquitoes [164]. For these genomes, using both adjacencies between the given markers and some long range synteny information on them, we obtained genome maps with reasonable levels of fragmentation considering the quality of the input data.

We also introduced an augmentation to ANGES, which we called FPMAG. This incorporates some of the techniques developed for reconstructing genome maps with repeats in order to include repeated markers into reconstructions. We presented some preliminary results using this software on the *Anopheles* data, and concluded with a number of suggestions how the results can be further improved.

**FPSAC.** The other major piece of software we introduced was FPSAC, which implements a pipeline for scaffolding contigs that have been assembled from ancient reads using phylogenetic information. The software significantly accounts for repeats in the ancestral genome, resolving them using long range synteny data extracted from related extant genomes. It also uses the extant sequences to correct and fill the gaps between the scaffolded contigs.

FPSAC was used to scaffold the main chromosome of the ancestral Black Death agent genome. Simulations done on the dataset using FPSAC provided evidence for the robustness of the method used for scaffolding on the *Yersinia* data. The results obtained corroborated earlier studies on the correlation between the locations of the breakpoints for the *Yersinia* phylogeny and insertion sequences [32, 55], and show evidence of a large-scale inversion during the evolution from the Black Death agent to the descendant species.

## 12.2 Extending this work

### 12.2.1 Theoretical questions

#### Genome maps with repeats

There are a number of theoretical questions that remain at the moment. A natural one would be to ask if we can combine the strategy used in Theorem 5.1 and by Maňuch et al. [138] to get an optimization algorithm which optimizes not only over the set of repeat spanning intervals, but also over the set of adjacencies in the instance. The general case is NP-hard [138], but there may still be restricted instances in terms of intervals which allow a polynomial time optimization algorithm. This would provide an algorithm to compute unambiguous genome maps when repeat content is masked, but repeat locations are fixed. This can be followed up by using local methods to determine the content of the repeated regions.

Question 6.1, which asks for the complexity of the partial optimization problem when all repeat spanning intervals contain at most 1 copy of a repeat, points to another important problem. A positive answer to this question would allow us to at least handle instances in which a single repeat does not have 2 or more occurrences in close proximity.

The definition of a repeat spanning interval as specified by us is quite strict: it imposes requirements for both the content and the order of the repeats. But it might be worthwhile to consider relaxing this definition. Such a relaxation may be in the form of merely stipulating a content requirement, or even asking for a partial order on the vertices in the interval. This would imply that we would lose control over the exact order of the repeats in the interval, but it might allow the design of algorithms that exploit the relaxed structure.

#### Vertex ordering problems

In the case of hypergraph vertex ordering problems, there are quite a few open questions. Both Question 7.1 and Conjecture 7.1 are major open questions regarding the polynomial time solvability of restricted instances of vertex ordering problems. As of the writing of this document, Question 7.1 has been open for over 4 years, and there is little to no known convincing evidence for either side of the argument. Conjecture 7.1 is a newly formulated problem, and the only advance made in proving it is the minor result provided in Appendix D.

There are also few known approximability results for such problems. Theorems 8.1 and 8.3 are, to our knowledge, the first instances of such results, save the result by Banerjee et al. [11]. This is not for lack of tools to obtain such results, but more a result of the relatively recent interest in analyzing partitioning and ordering problems on hypergraphs [129, 130]. Generalizing the known results for problems such as graph bandwidth, cutwidth and distortion to hypergraphs may have significant implications on the applicability of these parameters as measures of the error in synteny information obtained from data.

In a similar vein, the absence of fixed parameter tractable algorithms and exact exponential algorithms for hypergraph vertex ordering problems means that the comparative virtues of such approaches to reconstructing genome maps are hard to analyze. Unfortunately, while there are many such algorithms available for the corresponding problems on graphs [23, 80], generalizing these algorithms directly to handle hypergraphs may not be possible. The structure given by the bounded edge size in graphs is lost in hypergraphs. For example, we discussed how the parameterization of the bandwidth problem on graphs by the vertex cover [80] loses its nice structural properties when discussing hypergraphs. It is more likely that we will have to include an extra parameter, such as the maximum hyperedge size, in order to retain fixed parameter tractability.

Finally, the spectral approach we briefly discussed [9] should always be kept in mind. This method has proved useful in effectively dealing with non-realizable instances and returning a probable vertex order to reconstruct the genome map. However, apart from Vuokko's work [205], there are no theoretical justifications for why the order output by the algorithm for non-realizable instances should perform well as a candidate genome map.

While this section has been mostly theoretical, we motivated it as a means of discerning chimeric syntenies from those containing small errors. As of this moment, we still do not have a precise idea as to how to tell them apart. Ideally, we would like to have a measure of the veracity of a synteny. In terms of a vertex order, chimeric syntenies would correspond to hyperedges that are stretched more than the prediction by this measure, and then we would be able to actually use the ideas developed in this section for real applications. One idea, as formalized in Problem 7.3 which might be useful is to take the PQ-tree of a hypergraph which has the C1P, and find a permutation encoded in this tree such that, for some separate hyperedge not in the hypergraph, some notion of stretch is minimized.

### 12.2.2 Extending and adapting the methods

An immediate concern is to refine the current FPMAG pipeline discussed in Chapter 11 to obtain better ancestral genome maps. One idea is to use ANGES to assemble the backbone map consisting of unique markers and then using the FPMAG pipeline to include repeats. We could also try to use an explicit evolutionary model for the evolution of the genome between two unique markers. This model should account for insertions, deletions and duplications of genes, which would take into consideration the fact that the content between two unique markers will be repeats, and that there can be possible introgression.

Where FPSAC is concerned, the pipeline is as yet untested on larger, non-bacterial genomes. In this context, while the principle behind the methods used should still apply, the data becomes harder to analyze. As a consequence of the larger genome size, it is reasonable to expect much higher fragmentation in the initial assembly. The extant genomes being used for comparative scaffolding may also be longer. This may mean that different parts of an ancestral contig may map to many places on an extant genome, not necessarily consecutively, and we may also have many repeated regions. In this case, the final scaffold obtained will show much higher fragmentation. One way to try and combine scaffolds is to use reads that were discarded during contig assembly in order to fill the gaps. This is a direction that has not yet been taken for ancestral scaffolding.

The scope of the methods discussed has been limited to ancestral genome reconstruction till now. FPSAC and ANGES are examples of how the theoretical framework fits with the data available for such applications. However, the underlying theory should be extendable to many similar problems. The framework is not dissimilar to other models used in bioinformatics. There has been some recent work done on reconciling the results from two models from completely different fields in computational biology [128]. Such reconciliation may result in methods from either model being used for the other.

As a result, we believe that the model presented in this manuscript can be generalized and reconciled with existing models for genome assembly, mapping and scaffolding problems, of both extent and ancestral genomes. The issue will be to assess the suitability of the model, accounting for the nuances of the different data. Since the model is presented in such a general context, it is also essential to know the possible pitfalls when attacking other problems using it. For example, one may wish to use the model for scaffolding reliable extant contigs. In this context, we can define the markers as contigs, and syntenies as sets of overlapping contigs. Gaps can be filled using overlapping short reads discarded during contig assembly. Using such a definition, perhaps the FPSAC method can be extended to assembly and scaffolding of short genomes. Such a pipeline would necessitate the development of other tools however. It is hard to determine repeated genomic regions in this approach, and we have to resort to using coverage statistics in order to predict copy numbers. At the same time, the model may be unsuitable for such applications, in which case the reason that the model fails may be a pointer to better models for genome mapping and scaffolding.

## 12.3 Final thoughts and comments

Reconstructed ancestral genomes are a stepping stone for a large class of problems. It adds to the already massive amounts of genomic data available, and moreover, it provides a new perspective [131, 132, 152]. Accurately reconstructed ancestral genome maps are an invaluable resource to understanding the factors behind evolution. With the availability of good ancestral genome maps, we can start exploring whole genome evolution from the point of view of the ancestral genomes rather than working from the ground up. This direction of research can validate or point out flaws in the current models of evolution which are accepted as standard in the community.

As a consequence of their role in studying evolutionary mechanisms, ancestral genomes are also a key component in studying the effects of evolution on an organism. For example, ancestral genome maps of bacteria also permit an extended study of the syntenic factors behind the pathogeneticity of these organisms, and a parallel view of their evolution along with the evolution of the host organism is a promising direction. Such a study can provide enlightening clues to how the pathogen evolved to infect the host, and how it adapted to various evolutionary changes in the host genome. The *Yersinia* project [177] and the *Anopheles* project [164] can be seen as preliminary steps in such analysis. As more data accumulates, and as better techniques and computational resources develop, one would expect to uncover more and more biologically relevant data regarding the evolution of pathogens and vectors, what evolutionary change made them harmful to the host, how did they adapt to evolution within the hosts and so on.

Another long term project would be the use of ancestral genomes in reference assisted assembly. Reference based assembly is gaining a lot of popularity [118, 120] due to the large number of completely sequenced genomes that are appearing at a rapid rate. In the presence of ancestral reads or sequences, it is desirable to have a global assembly framework, in which we can jointly assemble extant and ancestral genomes in a phylogenetic context. Using such a framework, it may be possible to correct assembly errors using data from multiple related sequences, at the same time keeping in mind the evolutionary history of the genomes. It may also be possible to set up a cycle in which an assembled ancestral genome, containing information from ancestral reads, can be used to correct the assembly, mapping and scaffolding of extant genomes which were used as a reference to assemble it.

A technical point which we did not discuss was the possibility of more than 1 optimal solution to the mathematical problems. This means there may be more than 1 possible reconstruction of the ancestral genome. In such a scenario, it is useful to have a method to sample from the possible genomes. This is especially true when we are talking about evolutionary scenarios for various genomes, as even knowing the ancestor and the descendant does not necessarily mean that there is only one possible evolutionary pathway from the ancestral genome to the extant genome. Recent studies are increasingly focussing on these possibilities, which takes into account uncertainties in genome evolution and reconstruction [42, 147]. As a result, we can obtain a more accurate insight into the mechanism of genome evolution.

All these projects are tied to the production and availability of high quality ancestral genome maps, which brings us back to the methodology used in many reconstruction studies [43, 135, 170], including this dissertation. The primary advantage of the methodology is that it is well-established, and the independence from models of evolution is a useful feature to have if one wishes to evaluate such models. Along with alternative methods for ancestral reconstruction which do consider well-defined models of genome rearrangement [217], one can finally consider embarking on the projects mentioned earlier.

# Bibliography

[1] Adam, Z., Turmel, M., Lemieux, C., and Sankoff, D. Common intervals and symmetric difference in a model-free phylogenomics, with an application to streptophyte evolution. *J. Comput. Biol. 14*, 4 (2007), 436–445 (electronic). Special issue on the RECOMB Satellite Workshop on Comparative Genomics. 12

[2] Alizadeh, F., Karp, R. M., Newberg, L. A., and Weisser, D. K. Physical mapping of chromosomes: a combinatorial problem in molecular biology. *Algorithmica 13*, 1-2 (1995), 52–76. 3, 8, 16, 27

[3] Alon, N., and Milman, V. D. $\lambda_1$, isoperimetric inequalities for graphs, and superconcentrators. *J. Combin. Theory Ser. B 38*, 1 (1985), 73–88. 96

[4] Alon, N., and Spencer, J. H. *The probabilistic method*, third ed. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., Hoboken, NJ, 2008. With an appendix on the life and work of Paul Erdős. 55

[5] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. Basic local alignment search tool. *Journal of molecular biology 215*, 3 (1990), 403–410. 3, 128

[6] Arora, S., Lee, J. R., and Naor, A. Euclidean distortion and the sparsest cut [extended abstract]. In *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*. ACM, New York, 2005, pp. 553–562. 96, 105, 110

[7] Arora, S., Rao, S., and Vazirani, U. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing* (2004), ACM, New York, pp. 222–231. 96, 110

[8] Arora, S., Rao, S., and Vazirani, U. Expander flows, geometric embeddings and graph partitioning. *J. ACM 56*, 2 (2009), Art. 5, 37. 105, 106

[9] Atkins, J. E., Boman, E. G., and Hendrickson, B. A spectral algorithm for seriation and the consecutive ones problem. *SIAM J. Comput. 28*, 1 (1999), 297–310. 102, 125, 154, 156

[10] Bafna, V., and Pevzner, P. A. Genome rearrangements and sorting by reversals. *SIAM J. Comput. 25*, 2 (1996), 272–289. 5

[11] Banerjee, P., Sur-Kolay, S., Bishnu, A., Das, S., Nandy, S. C., and Bhattacharjee, S. Fpga placement using space-filling curves: Theory meets practice. *ACM Trans. Embed. Comput. Syst. 9*, 2 (Oct. 2009), 12:1–12:23. 98, 105, 155

[12] Bashir, A., Klammer, A., Robins, W. P., Chin, C.-S., Websetr, D., Paxinos, E., Hsu, D., Ashby, M., et al. A hybrid approach for the automated finishing of bacterial genomes. *Nature Biotechnol 30*, 7 (2012), 701–707. 10, 126

[13] Batzoglou, S., and Istrail, S. Physical mapping with repeated probes: the hypergraph superstring problem. *J. Discrete Algorithms 1*, 1 (2000), 51–76. 35

[14] Belcaid, M., Bergeron, A., Chateau, A., Chauve, C., Gingras, Y., Poisson, G., and Vendette, M. Exploring Genome Rearrangements using Virtual Hybridization. In *APBC'07: 5th Asia-Pacific Bioinformatics Conference* (Hong Kong, China, Jan. 2007), David Sankoff, Lusheng Wang, Francis Chin, Ed., vol. 5 of *Advances in Bioinformatics and Computational Biology*, Imperial College Press 2007, pp. 205–214. 13

[15] Benzer, S. On the topography of the genetic fine structure. *Proc. Natl. Acad. Sci. U.S.A. 47*, 3 (Mar 1961), 403–415. 27, 152

[16] Berger, B., Peng, J., and Singh, M. Computational solutions for omics data. *Nature Reviews Genetics 14*, 5 (2013), 333–346. 3

[17] Bergeron, A., Blanchette, M., Chateau, A., and Chauve, C. Reconstructing ancestral gene orders using conserved intervals. In *Algorithms in bioinformatics*, vol. 3240 of *Lecture Notes in Comput. Sci.* Springer, Berlin, 2004, pp. 14–25. 12, 16

[18] Bergeron, A., Chauve, C., de Montgolfier, F., and Raffinot, M. Computing common intervals of $K$ permutations, with applications to modular decomposition of graphs. *SIAM J. Discrete Math. 22*, 3 (2008), 1022–1039. 3, 14, 124

[19] Bergeron, A., Mixtacki, J., and Stoye, J. A unifying view of genome rearrangements. In *Algorithms in bioinformatics*, vol. 4175 of *Lecture Notes in Comput. Sci.* Springer, Berlin, 2006, pp. 163–173. 3, 5, 12

[20] Biller, P., Feijão, P., and Meidanis, J. a. Rearrangement-based phylogeny using the single-cut-or-join operation. *IEEE/ACM Trans. Comput. Biol. Bioinformatics 10*, 1 (Jan. 2013), 122–134. 3, 144

[21] Blanchette, M., Kunisawa, T., and Sankoff, D. Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution 49*, 2 (1999), 193–203. 6, 11

[22] Blum, A., Konjevod, G., Ravi, R., and Vempala, S. Semi-definite relaxations for minimum bandwidth and other vertex-ordering problems. In *STOC '98 (Dallas, TX)*. ACM, New York, 1999, pp. 100–105. 96

[23] Bodlaender, H. L., Fomin, F. V., Koster, A. M. C. A., Kratsch, D., and Thilikos, D. M. A note on exact algorithms for vertex ordering problems on graphs. *Theory Comput. Syst. 50*, 3 (2012), 420–432. 95, 96, 156

[24] Booth, K. S., and Lueker, G. S. Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms. *J. Comput. System Sci. 13*, 3 (1976), 335–379. Working Papers presented at the ACM-SIGACT Symposium on the Theory of Computing (Albuquerque, N. M., 1975). 35, 50

[25] Bos, K. I., Schuenemann, V. J., Golding, G. B., Burbano, H. A., Waglechner, N., Coombes, B. K., McPhee, J. B., DeWitte, S. N., Meyer, M., Schmedes, S., et al. A draft genome of Yersinia pestis from victims of the Black Death. *Nature 478*, 7370 (2011), 506–510. 10, 125, 126, 127, 132, 133, 136

[26] Bourque, G., Pevzner, P. A., and Tesler, G. Reconstructing the genomic architecture of ancestral mammals: Lessons from human, mouse, and rat genomes. *Genome Research 14*, 4 (2004), 507–516. 11

[27] Braga, M. D. V., Willing, E., and Stoye, J. Double cut and join with insertions and deletions. *J. Comput. Biol. 18*, 9 (2011), 1167–1184. 11

[28] Bryant, D., and Tupper, P. F. Hyperconvexity and tight-span theory for diversities. *Adv. Math. 231*, 6 (2012), 3172–3198. 120

[29] Bryant, D., and Tupper, P. F. Diversities and the geometry of hypergraphs. *Discrete Math. Theor. Comput. Sci. 16*, 2 (2014), 1–20. 120

[30] Cahill, M. J., Köser, C. U., Ross, N. E., and Archer, J. A. C. Read length and repeat resolution: Exploring prokaryote genomes using next-generation sequencing technologies. *PLoS ONE 5*, 7 (07 2010), e11518. 31

[31] Caprara, A. The reversal median problem. *INFORMS J. Comput. 15*, 1 (2003), 93–113. 11

[32] Chain, P., Carniel, E., Larimer, F., Lamerdin, J., Stoutland, P., et al. Insights into the evolution of *Yersinia pestis* through whole-genome comparison with Yersinia pseudotuberculosis. *Proc Natl Acad Sci U S A 101*, 38 (2004), 13826–13831. 4, 137, 154

[33] Chapman, J., et al. Meraculous: De novo genome assembly with short paired-end reads. *PLoS One 6* (2011), e23501. 10, 126

[34] Charikar, M., Hajiaghayi, M. T., Karloff, H., and Rao, S. $\ell_2^2$ spreading metrics for vertex ordering problems (extended abstract). In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2006), ACM, New York, pp. 1018–1027. 95, 96, 105, 109, 110, 113, 116, 120, 153

[35] Charikar, M., Hajiaghayi, M. T., Karloff, H., and Rao, S. $\ell_2^2$ spreading metrics for vertex ordering problems. *Algorithmica 56*, 4 (2010), 577–604. 96

[36] Chauve, C., El-Mabrouk, N., Guéguen, L., Semeria, M., and Tannier, E. Duplication, rearrangement and reconciliation: a follow-up 13 years later. In *Models and Algorithms for Genome Evolution*. Springer, 2013, pp. 47–62. 11

[37] Chauve, C., El-Mabrouk, N., and Tannier, E. *Models and Algorithms for Genome Evolution*. Springer Publishing Company, Incorporated, 2013. 3

[38] Chauve, C., Gavranovic, H., Ouangraoua, A., and Tannier, E. Yeast ancestral genome reconstructions: the possibilities of computational methods II. *J. Comput. Biol. 17* (2010), 1097–1112. 12, 13, 16, 123, 130, 142

[39] Chauve, C., Maňuch, J., and Patterson, M. On the gapped consecutive-ones property. In *European Conference on Combinatorics, Graph Theory and Applications (EuroComb 2009)*, vol. 34 of *Electron. Notes Discrete Math.* Elsevier Sci. B. V., Amsterdam, 2009, pp. 121–125. 94

[40] Chauve, C., Maňuch, J., Patterson, M., and Wittler, R. Tractability results for the consecutive-ones property with multiplicity. In *Combinatorial Pattern Matching*, vol. 6661 of *Lecture Notes in Comput. Sci.* Springer, Heidelberg, 2011, pp. 90–103. 36

[41] Chauve, C., Patterson, M., and Rajaraman, A. Hypergraph covering problems motivated by genome assembly questions. In *Combinatorial Algorithms - 24th International Workshop, IWOCA 2013, Rouen, France, July 10-12, 2013, Revised Selected Papers* (2013), pp. 428–432. 41, 57

[42] Chauve, C., Ponty, Y., and Zanetti, J. P. P. Evolution of genes neighborhood within reconciled phylogenies: An ensemble approach. In *Advances in Bioinformatics and Computational Biology*, S. Campos, Ed., vol. 8826 of *Lecture Notes in Comput. Sci.* Springer International Publishing, 2014, pp. 49–56. 14, 158

[43] Chauve, C., and Tannier, E. A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its applications to mammalian genomes. *PLoS Comput. Biol. 4* (2008), e1000234. 8, 12, 13, 14, 16, 92, 123, 127, 130, 158

[44] Cheeger, J. A lower bound for the smallest eigenvalue of the Laplacian. In *Problems in analysis (Papers dedicated to Salomon Bochner, 1969).* Princeton Univ. Press, Princeton, N. J., 1970, pp. 195–199. 96

[45] Chekuri, C., and Khanna, S. On multidimensional packing problems. *SIAM J. Comput. 33*, 4 (2004), 837–851. 89

[46] Chen, Y. P., and Chen, F. Using bioinformatics techniques for gene identification in drug discovery and development. *Curr. Drug Metab. 9*, 6 (Jul 2008), 567–573. 2

[47] Chial, H., and Craig, J. mtDNA and mitochondrial diseases. *Nature Education 1*, 1 (2008), 217. 5

[48] Christof, T., Jünger, M., Kececioglu, J., Mutzel, P., and Reinelt, G. A branch-and-cut approach to physical mapping of chromosomes by unique end-probes. *J. Comput. Biol. 4*, 4 (1997), 433–447. 16

[49] Chung, F. R. K. On optimal linear arrangements of trees. *Comput. Math. Appl. 10*, 1 (1984), 43–60. 100

[50] Courcelle, B. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. and Comput. 85*, 1 (1990), 12–75. 96

[51] Cristianini, N., and Hahn, M. W. *Introduction to computational genomics: a case studies approach.* Cambridge University Press, 2006. 3

[52] Csurös, M. Count: evolutionary analysis of phylogenetic profiles with parsimony and likelihood. *Bioinformatics 26* (2010), 1910–1912. 13, 130, 146, 150

[53] Cygan, M., Marx, D., Pilipczuk, M., Pilipczuk, M., and Schlotter, I. Parameterized complexity of Eulerian deletion problems. *Algorithmica 68*, 1 (2014), 41–61. 89

[54] Darling, A. E., Miklós, I., and Ragan, M. A. Dynamics of genome rearrangement in bacterial populations. *PLoS Genet 4*, 7 (2008), e1000128. 127, 137

[55] Deng, W., Burland, V., Plunkett, G., Boutin, A., Mayhew, G. F., Liss, P., Perna, N. T., Rose, D. J., et al. Genome sequence of *Yersinia pestis KIM* . *J Bacteriol 184*, 16 (Aug 2002), 4601–4611. 137, 154

[56] Denoeud, F., Carretero-Paulet, L., Dereeper, A., et al. The coffee genome provides insight into the convergent evolution of caffeine biosynthesis. *Science 345*, 6201 (2014), 1181–1184. 11

[57] Deshpande, V., Fung, E. D. K., Pham, S., and Bafna, V. Cerulean: A hybrid assembly using high throughput short and long reads. In *WABI* (2013), vol. 8126 of *Lecture Notes in Computer Science*, Springer, pp. 349–363. 9

[58] Dessmark, A., Lingas, A., and Garrido, O. On parallel complexity of maximum $f$-matching and the degree sequence problem. In *Mathematical foundations of computer science 1994 (Košice, 1994)*, vol. 841 of *Lecture Notes in Comput. Sci.* Springer, Berlin, 1994, pp. 316–325. 39, 55, 68

[59] Devanur, N. R., Khot, S. A., Saket, R., and Vishnoi, N. K. Integrality gaps for sparsest cut and minimum linear arrangement problems. In *STOC'06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing.* ACM, New York, 2006, pp. 537–546. 96, 104

[60] Diallo, A. B., Makarenkov, V., and Blanchette, M. Ancestors 1.0: a web server for ancestral sequence reconstruction. *Bioinformatics 26*, 1 (2010), 130–131. 13

[61] Distelfeld, A., Uauy, C., Fahima, T., and Dubcovsky, J. Physical map of the wheat high-grain protein content gene gpc-b1 and development of a high-throughput molecular marker. *New Phytologist 169*, 4 (2006), 753–763. 8

[62] Djelouadji, Z., Raoult, D., and Drancourt, M. Palaegenomics of *Mycobacterium tuberculosis*: epidemic burst with a degrading genome. *Lancet Infect Dis 11* (2011), 641–650. 10

[63] Dobzhansky, T., et al. *Genetics of the evolutionary process*, vol. 139. Columbia University Press New York, 1970. 4

[64] Dom, M. Algorithmic aspects of the consecutive-ones property. *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS*, 98 (2009), 27–59. 37

[65] Dom, M., Guo, J., and Niedermeier, R. Approximation and fixed-parameter algorithms for consecutive ones submatrix problems. *J. Comput. System Sci. 76*, 3-4 (2010), 204–221. 38, 57

[66] Donmez, N., and Brudno, M. Scarpa: scaffolding reads with practical algorithms. *Bioinformatics 29*, 4 (2013), 428–434. 10, 126

[67] Donoghue, H. Insights gained from paleomicrobiology into ancient and modern tuberculosis. *Clin Microbiol Infect 17* (2011), 821–829. 10, 126

[68] Donoghue, H., and Spigelman, M. Pathogenic microbial ancient dna: a problem or an opportunity. *Proc R Soc B 273*, 1587 (2006), 641–642. 126

[69] Downey, R., and Fellows, M. Fixed-parameter tractability and completeness. III. Some structural aspects of the $W$ hierarchy. In *Complexity theory*. Cambridge Univ. Press, Cambridge, 1993, pp. 191–225. 180

[70] Downey, R. G., and Fellows, M. R. Fixed-parameter tractability and completeness. I. Basic results. *SIAM J. Comput. 24*, 4 (1995), 873–921. 180, 181

[71] Downey, R. G., and Fellows, M. R. Fixed-parameter tractability and completeness. II. On completeness for $W[1]$. *Theoret. Comput. Sci. 141*, 1-2 (1995), 109–131. 180, 181

[72] Drancourt, M., and Raoult, D. Palaemicrobiology: current issues and perspectives. *Nature Rev Microbiol 3* (2005), 23–35. 126, 127

[73] Earnest-DeYoung, J. V., Lerat, E., and Moret, B. M. Reversing gene erosion âĂŞ reconstructing ancestral bacterial genomes from gene-content and order data. In *Algorithms in Bioinformatics*, I. Jonassen and J. Kim, Eds., vol. 3240 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, pp. 1–13. 11

[74] Edgar, R. C. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res. 32*, 5 (2004), 1792–1797. 134

[75] El-Mabrouk, N. Reconstructing an ancestral genome using minimum segments duplications and reversals. *J. Comput. Syst. Sci. 65*, 3 (2002), 442–464. 11

[76] Even, G., Naor, J., Rao, S., and Schieber, B. Divide-and-conquer approximation algorithms via spreading metrics. *J. ACM 47*, 4 (2000), 585–616. 95, 105, 108, 109, 110

[77] Feige, U. Approximating the bandwidth via volume respecting embeddings. *J. Comput. System Sci. 60*, 3 (2000), 510–539. 30th Annual ACM Symposium on Theory of Computing (Dallas, TX, 1998). 96, 120

[78] Feige, U., and Lee, J. R. An improved approximation ratio for the minimum linear arrangement problem. *Inform. Process. Lett. 101*, 1 (2007), 26–29. 95, 96, 105, 109, 110, 120, 153

[79] Feijão, P., and Meidanis, J. a. SCJ: A breakpoint-like distance that simplifies several rearrangement problems. *IEEE/ACM Trans. Comput. Biol. Bioinformatics 8*, 5 (Sept. 2011), 1318–1329. 5, 10, 11, 143

[80] Fellows, M. R., Lokshtanov, D., Misra, N., Rosamond, F. A., and Saurabh, S. Graph layout problems parameterized by vertex cover. In *Algorithms and computation*, vol. 5369 of *Lecture Notes in Comput. Sci.* Springer, Berlin, 2008, pp. 294–305. 97, 101, 156

[81] Felsenstein, J., and Felenstein, J. *Inferring phylogenies*, vol. 2. Sinauer Associates Sunderland, 2004. 123

[82] Fertin, G., Labarre, A., Rusu, I., Tannier, É., and Vialette, S. *Combinatorics of genome rearrangements*. Computational Molecular Biology. MIT Press, Cambridge, MA, 2009. 6, 10, 11, 21

[83] Fitch, W. M. Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Biology 20*, 4 (1971), 406–416. 132, 134

[84] Fleischner, H. *Eulerian graphs and related topics. Part 1. Vol. 2*, vol. 50 of *Annals of Discrete Mathematics*. North-Holland Publishing Co., Amsterdam, 1991. 50

[85] Fontaine, M. C., et al. Extensive introgression in a malaria vector species complex revealed by phylogenomics. *Science 347*, 6217 (2015). 141

[86] Fréville, A., and Plateau, G. Sac à dos multidimensionnel en variables 0-1: encadrement de la somme des variables à l'optimum. *RAIRO Rech. Opér. 27*, 2 (1993), 169–187. 83

[87] Fulkerson, D. R., and Gross, O. A. Incidence matrices and interval graphs. *Pacific J. Math. 15* (1965), 835–855. 16, 35, 152

[88] Gagnon, Y., Blanchette, M., and El-Mabrouk, N. A flexible ancestral genome reconstruction method based on gapped adjacencies. *BMC Bioinformatics 13 Suppl 19* (2012), S4. 11, 127

[89] Gao, S., Sung, W.-K., and Nagarajan, N. Opera: Reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *Journal of Computational Biology 18*, 11 (2011), 1681–1691. 10, 126

[90] Garey, M. R., Graham, R. L., Johnson, D. S., and Knuth, D. E. Complexity results for bandwidth minimization. *SIAM J. Appl. Math. 34*, 3 (1978), 477–495. 95

[91] Garey, M. R., Johnson, D. S., and Stockmeyer, L. Some simplified $NP$-complete problems. In *Sixth Annual ACM Symposium on Theory of Computing (Seattle, Wash., 1974)*. Assoc. Comput. Mach., New York, 1974, pp. 47–63. 95

[92] Gärtner, B., and Matoušek, J. *Approximation algorithms and semidefinite programming*. Springer, Heidelberg, 2012. 184

[93] Gentleman, R. C., Carey, V. J., Bates, D. M., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome biology 5*, 10 (2004), R80. 2

[94] Gnerre, S., Lander, E. S., Lindblad-Toh, K., and Jaffe, D. B. Assisted assembly: how to improve a de novo genome assembly by using related species. *Genome Biol 10* (2009), R88. 9, 10, 126

[95] Goemans, M. X., and Williamson, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach. 42*, 6 (1995), 1115–1145. 184

[96] Goldberg, M. A., and Klipker, I. A. Minimal placing of trees on a line. *Technical report Physico-Technical Institute of Low Temperatures, Academy of Sciences of Ukranian SSR, USSR (in Russian)* (1976). 95, 100

[97] Goldberg, P. W., Golumbic, M. C., Kaplan, H., and Shamir, R. Four strikes against physical mapping of DNA. *Journal of Computational Biology 2*, 1 (1995), 139–152. 27, 94

[98] Grötschel, M., Lovász, L., and Schrijver, A. *Geometric algorithms and combinatorial optimization*, second ed., vol. 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 1993. 108, 110, 111, 183, 184

[99] Habib, M., McConnell, R., Paul, C., and Viennot, L. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoret. Comput. Sci. 234*, 1-2 (2000), 59–84. 35

[100] Haddadi, S. A note on the NP-hardness of the consecutive block minimization problem. *Int. Trans. Oper. Res. 9*, 6 (2002), 775–777. 94

[101] Haensch, S., Bianucci, R., Signoli, M., Rajerison, M., Schultz, M., et al. Distinct clones of *yersinia pestis* caused the Black Death. *PLoS Pathog 6*, 10 (10 2010), e1001134. 133

[102] Han, M. V., Thomas, G. W., Lugo-Martinez, J., and Hahn, M. W. Estimating gene gain and loss rates in the presence of error in genome assembly and annotation using cafe 3. *Molecular Biology and Evolution 30*, 8 (2013), 1987–1997. 150

[103] Handl, J., Knowles, J., and Kell, D. B. Computational cluster validation in post-genomic data analysis. *Bioinformatics 21*, 15 (2005), 3201–3212. 3

[104] Helmberg, C., Rendl, F., Mohar, B., and Poljak, S. A spectral approach to bandwidth and separator problems in graphs. *Linear and Multilinear Algebra 39*, 1-2 (1995), 73–90. 96

[105] Hilker, R., Sickinger, C., Pedersen, C. N., and Stoye, J. UniMoG- a unifying framework for genomic distance calculation and sorting based on DCJ. *Bioinformatics 28*, 19 (2012), 2509–2511. 142

[106] Husemann, P., and Stoye, J. r2cat: synteny plots and comparative assembly. *Bioinformatics 26*, 4 (2010), 570–571. 10, 127

[107] Huson, D. H., Reinert, K., and Myers, E. W. The greedy path-merging algorithm for contig scaffolding. *J. ACM 49* (2002), 603–615. 10

[108] Idury, R. M., and Waterman, W. S. A new algorithm for DNA sequence assembly. *J. Comput. Biol. 2* (1995), 291–306. 9

[109] Jahn, K., Zheng, C., Kováč, J., and Sankoff, D. A consolidation algorithm for genomes fractionated after higher order polyploidization. *BMC Bioinformatics 13 Suppl 19* (2012), S8. 11

[110] Jones, B. R., Rajaraman, A., Tannier, E., and Chauve, C. ANGES: reconstructing ANcestral GEnomeS maps. *Bioinformatics 28*, 18 (2012), 2388–2390. 3, 8, 13, 122, 127, 154

[111] Juvan, M., and Mohar, B. Optimal linear labelings and eigenvalues of graphs. *Discrete Appl. Math. 36*, 2 (1992), 153–168. 96

[112] Kaplan, H., Shamir, R., and Tarjan, R. E. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput. 28*, 5 (1999), 1906–1922. 94

[113] Karp, R. M. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*. Plenum, New York, 1972, pp. 85–103. 59

[114] Katona, G. Y. Extension of paths and cycles for hypergraphs. *Electronic Notes in Discrete Mathematics 45*, 0 (2014), 3–7. 99

[115] Katona, G. Y., and Kierstead, H. A. Hamiltonian chains in hypergraphs. *J. Graph Theory 30*, 3 (1999), 205–212. 99

[116] Kececioglu, J. D., and Myers, E. W. Combinatorial algorithms for DNA sequence assembly. *Algorithmica 13* (1995), 7–51. 9

[117] Khot, S. On the power of unique 2-prover 1-round games. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing* (2002), ACM, New York, pp. 767–775. 184

[118] Kim, J., Larkin, D., Cai, Q., Asan, Zhang, Y., Auvil, L., Capitanu, B., Zhang, G., Lewin, H., and Ma, J. Reference-assisted chromosome assembly. *Proc Natl Acad Sci 110*, 5 (2013), 1785–1790. 10, 126, 158

[119] Kohara, Y., Akiyama, K., and Isono, K. The physical map of the whole E. coli chromosome: application of a new strategy for rapid analysis and sorting of a large genomic library. *Cell 50*, 3 (Jul 1987), 495–508. 8

[120] Kolmogorov, M., Raney, B., Paten, B., and Pham, S. RagoutâĂŤa reference-assisted assembly tool for bacterial genomes. *Bioinformatics 30*, 12 (2014), i302–i309. 158

[121] Koren, S., et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nat. Biotechnol. 30* (2012), 693–700. 9

[122] Kou, L. T. Polynomial complete consecutive information retrieval problems. *SIAM J. Comput. 6*, 1 (1977), 67–75. 94

[123] Krop, E. *Enumerating matchings in regular graphs.* ProQuest LLC, Ann Arbor, MI, 2007. Thesis (Ph.D.)–University of Illinois at Chicago. 55

[124] Krzywinski, M., Schein, J., Birol, I., Connors, J., Gascoyne, R., Horsman, D., Jones, S. J., and Marra, M. A. Circos: an information aesthetic for comparative genomics. *Genome Res. 19*, 9 (Sep 2009), 1639–1645. 135

[125] Li, R., Fan, W., Tian, G., Zhu, H., He, L., Cai, J., Huang, Q., Cai, Q., Li, B., Bai, Y., et al. The sequence and de novo assembly of the giant panda genome. *Nature 463*, 7279 (Jan 2010), 311–317. 9

[126] Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., Li, S., Yang, H., Wang, J., and Wang, J. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research 20*, 2 (2010), 265–272. 3, 9

[127] Lin, H., Goldstein, S., Mendelowitz, L., Zhou, S., Wetzel, J., Schwartz, D., and Pop, M. AGORA: Assembly guided by optical restriction alignment. *BMC Bioinformatics 13* (2012), 189. 10, 126

[128] Lin, Y., Nurk, S., and Pevzner, P. A. What is the difference between the breakpoint graph and the de bruijn graph? *BMC Genomics 15*, Suppl 6 (2014), S6. 157

[129] Louis, A. Hypergraph markov operators, eigenvalues and approximation algorithms. *CoRR abs/1408.2425* (2014). 105, 155

[130] Louis, A., and Makarychev, Y. Approximation algorithms for hypergraph small set expansion and small set vertex expansion. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain* (2014), pp. 339–355. 105, 155

[131] Louis, A., Muffato, M., and Roest Crollius, H. Genomicus: five genome browsers for comparative genomics in eukaryota. *Nucleic Acids Research 41*, D1 (2013), D700–D705. 157

[132] Louis, A., Nguyen, N. T. T., Muffato, M., and Roest Crollius, H. Genomicus update 2015: Karyoview and matrixview provide a genome-wide perspective to multispecies comparative genomics. *Nucleic Acids Research* (2014). 157

[133] Lu, W. F., and Hsu, W. L. A test for the consecutive ones property on noisy data–application to physical mapping and sequence assembly. *J. Comput. Biol. 10*, 5 (2003), 709–735. 16, 27

[134] Ma, J., Ratan, A., Raney, B. J., Suh, B. B., Zhang, L., Miller, W., and Haussler, D. DUPCAR: reconstructing contiguous ancestral regions with duplications. *J. Comput. Biol. 15*, 8 (Oct 2008), 1007–1027. 127, 131

[135] Ma, J., Zhang, L., Suh, B. B., Raney, B. J., Burhans, R. C., Kent, W. J., Blanchette, M., Haussler, D., and Miller, W. Reconstructing contiguous regions of an ancestral genome. *Genome Res. 16*, 12 (Dec 2006), 1557–1565. 11, 12, 13, 14, 124, 127, 130, 136, 158

[136] Makeig, S., Jung, T.-P., Bell, A. J., Ghahremani, D., and Sejnowski, T. J. Blind separation of auditory event-related brain responses into independentâĂĽcomponents. *Proceedings of the National Academy of Sciences 94*, 20 (1997), 10979–10984. 3

[137] Maňuch, J., Patterson, M., and Chauve, C. Hardness results on the gapped consecutive-ones property problem. *Discrete Appl. Math. 160*, 18 (2012), 2760–2768. 94

[138] Maňuch, J., Patterson, M., Wittler, R., Chauve, C., and Tannier, E. Linearization of ancestral multichromosomal genomes. *BMC Bioinformatics 13*, S-19 (2012), S11. 38, 39, 57, 68, 125, 131, 146, 148, 152, 155

[139] Mardis, E. R. The impact of next-generation sequencing technology on genetics. *Trends in Genetics 24*, 3 (2008), 133–141. 2

[140] Martin, M. D., Cappellini, E., Samaniego, J. A., Zepeda, M. L., Campos, P. F., Seguin-Orlando, A., et al. Reconstructing genome evolution in historic samples of the Irish potato famine pathogen. *Nat Commun 4* (2013), 2172. 126

[141] Marx, D., and Cao, Y. Interval deletion is fixed-parameter tractable. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (2014), pp. 122–141. 38

[142] McConnell, R. M. A certifying algorithm for the consecutive-ones property. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2004), ACM, New York, pp. 768–777 (electronic). 35, 124, 125, 154

[143] McCoy, R. C., Taylor, R. W., Blauwkamp, T. A., Kelley, J. L., Kertesz, M., Pushkarev, D., Petrov, D. A., and Fiston-Lavier, A.-S. Illumina truseq synthetic long-reads empower de novo assembly and resolve complex, highly repetitive transposable elements. *bioRxiv 001834* (2014). 9

[144] McPherson, J. D., Marra, M., Hillier, L., Waterston, R. H., Chinwalla, A., Wallis, J., Sekhon, M., Wylie, K., Mardis, E. R., Wilson, R. K., et al. A physical map of the human genome. *Nature 409*, 6822 (2001), 934–941. 8

[145] Medvedev, P., Georgiou, K., Myers, E. W., and Brudno, M. Computability of models for sequence assembly. In *WABI* (2007), vol. 4645 of *Lecture Notes in Comput. Sci.*, pp. 289–301. 3, 9

[146] Micali, S. M., and Vazirani, V. V. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980* (1980), pp. 17–27. 39, 68

[147] Miklós, I., Kiss, S. Z., and Tannier, E. Counting and sampling SCJ small parsimony solutions. *Theoret. Comput. Sci. 552* (2014), 83–98. 158

[148] Miklós, I., and Tannier, E. Bayesian sampling of genomic rearrangement scenarios via double cut and join. *Bioinformatics 26*, 24 (2010), 3012–3019. 137

[149] Monien, B. The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM J. Algebraic Discrete Methods 7*, 4 (1986), 505–512. 95, 100, 101

[150] Morelli, G., Song, Y., Mazzoni, C. J., Eppinger, M., Roumagnac, P., Wagner, D. M., Feldkamp, M., Kusecek, B., Vogler, A. J., Li, Y., Cui, Y., Thomson, N. R., Jombart, T., Leblois, R., Lichtner, P., Rahalison, L., Petersen, J. M., Balloux, F., Keim, P., Wirth, T., Ravel, J., Yang, R., Carniel, E., and Achtman, M. Yersinia pestis genome sequencing identifies patterns of global phylogenetic diversity. *Nat. Genet. 42*, 12 (Dec 2010), 1140–1143. 12, 133

[151] Muffato, M., and Crollius, H. R. Paleogenomics in vertebrates, or the recovery of lost genomes from the mist of time. *BioEssays 30*, 2 (2008), 122–134. 4

[152] Muffato, M., Louis, A., Poisnel, C. E., and Roest Crollius, H. Genomicus: a database and a browser to study gene synteny in modern and ancestral genomes. *Bioinformatics 26*, 8 (Apr 2010), 1119–1121. 12, 157

[153] Müller, A. C., Bruggeman, F. J., Olivier, B. G., and Stougie, L. Fast flux module detection using matroid theory. In *Research in Computational Molecular Biology*, R. Sharan, Ed., vol. 8394 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 192–206. 3

[154] Munoz, A., Zheng, C., Zhu, Q., Albert, V. A., Rounsley, S., and Sankoff, D. Scaffold filling, contig fusion and comparative gene order inference. *BMC Bioinformatics 11* (2010), 304. 127

[155] Murat, F., Xu, J., Tannier, E., Abrouk, M., Guilhot, N., Pont, C., Messing, J., and Salse, J. Ancestral grass karyotype reconstruction unravels new mechanisms of genome shuffling as a source of plant evolution. *Genome Research 20*, 11 (2010), 1545–1557. 12

[156] Murat, F., Zhang, R., Guizard, S., Flores, R., Armero, A., Pont, C., Steinbach, D., Quesneville, H., Cooke, R., and Salse, J. Shared subgenome dominance following polyploidization explains grass genome evolutionary plasticity from a seven protochromosome ancestor with 16k protogenes. *Genome Biology and Evolution 6*, 1 (2014), 12–33. 125

[157] Myers, E. W. Toward simplifying and accurately formulating fragment assembly. *J. Comput. Biol. 2* (1995), 275–290. 9

[158] Myers, E. W. The fragment assembly string graph. *Bioinformatics 21*, suppl 2 (2005), ii79–ii85. 9

[159] Nagarajan, N., and Pop, M. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *J. Comput. Biol. 16*, 7 (Jul 2009), 897–908. 9

[160] Nagarajan, N., and Pop, M. Sequence assembly demystified. *Nature Reviews Genetics 14* (2013), 157–167. 9

[161] Narayanaswamy, N. S., and Subhasini, R. FPT algorithms for consecutive ones submatrix problems. In *Parameterized and Exact Computation*, G. Gutin and S. Szeider, Eds., vol. 8246 of *Lecture Notes in Comput. Sci.* Springer International Publishing, 2013, pp. 295–307. 38

[162] National Institutes of Health. Talking Glossary of Genetic Terms., 2006. 5

[163] Neafsey, D. E., Christophides, G. K., Collins, F. H., Emrich, S. J., Fontaine, M. C., Gelbart, W., et al. The evolution of the Anopheles 16 genomes project. *G3 (Bethesda) 3*, 7 (Jul 2013), 1191–1194. 139, 140, 141

[164] Neafsey, D. E., et al. Highly evolvable malaria vectors: The genomes of 16 anopheles mosquitoes. *Science 347*, 6217 (2015). 12, 125, 139, 140, 143, 144, 145, 154, 158

[165] Niedermeier, R. *Invitation to fixed-parameter algorithms*, vol. 31 of *Oxford Lecture Series in Mathematics and its Applications.* Oxford University Press, Oxford, 2006. 55, 56

[166] Noble, D. The rise of computational biology. *Nature Reviews Molecular Cell Biology 3*, 6 (2002), 459–463. 2

[167] Olson, M., Hood, L., Cantor, C., and Botstein, D. A common language for physical mapping of the human genome. *Science 245*, 4925 (Sep 1989), 1434–1435. 7

[168] Orlando, L., Ginolhac, A., Zhang, G., Froese, D., Albrechtsen, A., Stiller, M., Schubert, M., et al. Recalibrating Equus evolution using the genome sequence of an early Middle Pleistocene horse. *Nature 499*, 7456 (Jul 2013), 74–78. 126

[169] Otero, P., Hersh, W., and Jai Ganesh, A. U. Big Data: Are Biomedical and Health Informatics Training Programs Ready? Contribution of the IMIA Working Group for Health and Medical Informatics Education. *Yearb Med Inform 9*, 1 (2014), 177–181. 2

[170] Ouangraoua, A., Tannier, E., and Chauve, C. Reconstructing the architecture of the ancestral amniote genome. *Bioinformatics 27*, 19 (Oct 2011), 2664–2671. 10, 13, 16, 92, 94, 158

[171] Papadimitriou, C. H. The NP-completeness of the bandwidth minimization problem. *Computing 16*, 3 (1976), 263–270. 95

[172] Pell, J., Hintze, A., Canino-Koning, R., Howe, A., Tiedje, J. M., and Brown, C. T. Scaling metagenome sequence assembly with probabilistic de bruijn graphs. *Proceedings of the National Academy of Sciences 109*, 33 (2012), 13272–13277. 138

[173] Pevzner, P. A., Tang, H., and Waterman, M. S. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences 98*, 17 (2001), 9748–9753. 9

[174] Pop, M. Genome assembly reborn: recent computational challenges. *Briefings in Bioinformatics 10*, 4 (2009), 354–366. 9

[175] Raffinot, M. Consecutive ones property testing: cut or swap. In *Models of computation in context*, vol. 6735 of *Lecture Notes in Comput. Sci.* Springer, Heidelberg, 2011, pp. 239–249. 35

[176] Raghavendra, P. Optimal algorithms and inapproximability results for every CSP? [extended abstract]. In *STOC'08*. ACM, New York, 2008, pp. 245–254. 184

[177] Rajaraman, A., Tannier, E., and Chauve, C. Fpsac: fast phylogenetic scaffolding of ancient contigs. *Bioinformatics 29*, 23 (2013), 2987–2994. 12, 13, 42, 59, 125, 132, 158

[178] Rao, S., and Richa, A. W. New approximation techniques for some linear ordering problems. *SIAM J. Comput. 34*, 2 (2004/05), 388–404 (electronic). 95, 105, 107, 108, 110

[179] Renegar, J. *A mathematical view of interior-point methods in convex optimization.* MPS/SIAM Series on Optimization. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA; Mathematical Programming Society (MPS), Philadelphia, PA, 2001. 184

[180] Ribeiro, F. J., Przybylski, D., Yin, S., et al. Finished bacterial genomes from shotgun sequence data. *Genome Res 22* (2012), 2270–2277. 10, 126

[181] Rissman, A. I., Mau, B., Biehl, B. S., Darling, A. E., Glasner, J. D., and Perna, N. T. Reordering contigs of draft genomes using the mauve aligner. *Bioinformatics 25*, 16 (2009), 2071–2073. 127

[182] Rodríguez, J. A. On the Laplacian eigenvalues and metric parameters of hypergraphs. *Linear Multilinear Algebra 50*, 1 (2002), 1–14. 97

[183] Salmela, L., et al. Fast scaffolding with small independent mixed integer programs. *Bioinformatics 27* (2011), 3259–3265. 10, 126

[184] Sankoff, D. Genome rearrangement with gene families. *Bioinformatics 15*, 11 (1999), 909–917. 3, 6, 11

[185] Sankoff, D., and Zheng, C. Fractionation, rearrangement, consolidation, reconstruction. In *Models and Algorithms for Genome Evolution*. Springer, 2013, pp. 247–260. 11, 39

[186] Saxe, J. B. Dynamic-programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM J. Algebraic Discrete Methods 1*, 4 (1980), 363–369. 94

[187] Schadt, E. E., Linderman, M. D., Sorenson, J., Lee, L., and Nolan, G. P. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics 11*, 9 (2010), 647–657. 2

[188] Schmidt, T., and Stoye, J. Quadratic time algorithms for finding common intervals in two and more sequences. In *Combinatorial Pattern Matching, 15th Annual Symposium, CPM 2004, Istanbul,Turkey, July 5-7, 2004, Proceedings* (2004), pp. 347–358. 14, 124

[189] Schneeberger, K., Ossowski, S., Ott, F., Klein, J. D., Wang, X., Lanz, C., Smith, L. M., Cao, J., Fitz, J., Warthmann, N., Henz, S. R., Huson, D. H., and Weigel, D. Reference-guided assembly of four diverse arabidopsis thaliana genomes. *Proceedings of the National Academy of Sciences 108*, 25 (2011), 10249–10254. 9

[190] Schuenemann, V. J., Bos, K., DeWitte, S., Schmedesd, S., et al. Targeted enrichment of ancient pathogens yielding the pPCP1 plasmid of *Yersinia pestis* from victims of the black death. *Proc Natl Acad Sci U S A 108* (2011), E746–E752. 126

[191] Schuenemann, V. J., Singh, P., Mendum, T. A., Krause-Kyora, B., Jäger, G., Bos, K. I., Herbig, A., Economou, C., Benjak, A., Busso, P., et al. Genome-wide comparison of medieval and modern mycobacterium leprae. *Science 341*, 6142 (2013), 179–183. 126

[192] Schuster, S. C. Next-generation sequencing transforms todayâĂŹs biology. *Nature 200*, 8 (2007). 2

[193] Schuster, S. C. Next-generation sequencing transforms today's biology. *Nat. Methods 5*, 1 (Jan 2008), 16–18. 9

[194] Sebat, J., Lakshmi, B., Troge, J., Alexander, J., Young, J., Lundin, P., et al. Large-scale copy number polymorphism in the human genome. *Science 305*, 5683 (2004), 525–528. 4

[195] Shiloach, Y. A minimum linear arrangement algorithm for undirected trees. *SIAM J. Comput. 8*, 1 (1979), 15–32. 100

[196] Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J., and Birol, Ä. ABySS: A parallel assembler for short read sequence data. *Genome Research 19*, 6 (2009), 1117–1123. 3, 9

[197] Sipser, M. *Introduction to the theory of computation.* PWS Publishing Company, 1997. 180

[198] Tannier, E., Zheng, C., and Sankoff, D. Multichromosomal median and halving problems under different genomic distances. *BMC Bioinformatics 10* (2009). 11, 39

[199] Tarjan, R. E., and Yannakakis, M. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput. 13*, 3 (1984), 566–579. 99

[200] Treangen, T. J., and Salzberg, S. L. Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nature. Rev. Genet. 13* (2012), 36–46. 31

[201] Tucker, A. Matrix characterizations of circular-arc graphs. *Pacific J. Math. 39* (1971), 535–545. 28, 35

[202] Tucker, A. A structure theorem for the consecutive 1's property. *J. Combinatorial Theory Ser. B 12* (1972), 153–162. 38, 99

[203] Uno, T. A fast algorithm for enumerating non-bipartite maximal matchings. Research Report, 2001. Dept. Math. and Comp., Tokyo Institute of Technology, Japan. 55

[204] Vandenberghe, L., and Boyd, S. Semidefinite programming. *SIAM Rev. 38*, 1 (1996), 49–95. 184

[205] Vuokko, N. Consecutive ones property and spectral ordering. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA* (2010), pp. 350–360. 103, 125, 156

[206] Waterhouse, R. M., Tegenfeldt, F., Li, J., Zdobnov, E. M., and Kriventseva, E. V. OrthoDB: a hierarchical catalog of animal, fungal and bacterial orthologs. *Nucleic Acids Res. 41*, Database issue (Jan 2013), D358–365. 140

[207] Wilson, D. Insights from genomics into bacterial pathogen populations. *PLoS Pathog 8* (2012), e1002874. 126

[208] Wittler, R., Maňuch, J., Patterson, M., and Stoye, J. Consistency of sequence-based gene clusters. *J. Comput. Biol. 18*, 9 (2011), 1023–1039. 28, 35, 36, 42, 131, 152

[209] Xie, Y., Wu, G., Tang, J., Luo, R., Patterson, J., Liu, S., et al. SOAPdenovo-Trans: de novo transcriptome assembly with short rna-seq reads. *Bioinformatics* (2014). 3, 60, 152

[210] Yancopoulos, S., and Friedberg, R. Dcj path formulation for genome transformations which include insertions, deletions, and duplications. *Journal of Computational Biology 16*, 10 (2009), 1311–1338. 11

[211] Yang, Z., and Sankoff, D. Natural parameter values for generalized gene adjacency. *J. Comput. Biol. 17*, 9 (2010), 1113–1128. 94

[212] Yoshida, K., Schuenemann, V. J., Cano, L. M., Pais, M., Mishra, B., et al. The rise and fall of the phytophthora infestans lineage that triggered the irish potato famine. *eLife 2* (2013). 126

[213] Yu, J., and Buckler, E. S. Genetic association mapping and genome organization of maize. *Curr. Opin. Biotechnol. 17*, 2 (Apr 2006), 155–160. 4

[214] Zerbino, D. R., and Birney, E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res. 18*, 5 (May 2008), 821–829. 3, 9, 60, 132, 137, 152

[215] Zhang, P., Schon, E. A., Fischer, S. G., Cayanis, E., Weiss, J., Kistler, S., and Bourne, P. E. An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA. *Computer Applications in the Biosciences 10*, 3 (1994), 309–317. 94

[216] Zhang, Z., Schwartz, S., Wagner, L., and Miller, W. A greedy algorithm for aligning DNA sequences. *J. Comput. Biol. 7*, 1-2 (2000), 203–214. 133

[217] Zheng, C., and Sankoff, D. Gene order in rosid phylogeny, inferred from pairwise syntenies among extant genomes. *BMC Bioinformatics 13 Suppl 10* (2012), S9. 6, 11, 12, 158

[218] Zhu, Q., Adam, Z., Choi, V., and Sankoff, D. Generalized gene adjacencies, graph bandwidth and clusters in yeast evolution. In *Bioinformatics research and applications*, vol. 4983 of *Lecture Notes in Comput. Sci.* Springer, Berlin, 2008, pp. 134–145. 94

# Appendix A

# Pseudocode for algorithms in text

## A.1 Algorithm for Lemma 4.1

This section specifies the construction under which Lemma 4.1 holds. We reuse this constructions as a subroutine for other algorithms in the text, specifically for Theorem 4.1 and as a polynomial time construction routine for the reduction in Theorem 6.1. Let $\mathbf{deg}_H(v)$ denote the degree of the vertex in the underlying adjacency instance of $H$.

Note that the algorithm either returns an instance without repeat spanning intervals, or, if the number of occurrences of a repeat occurs in the repeat spanning intervals is greater than its multiplicity, it returns **No** to the question of if the instance can be realized in a genome model.

## A.2 Algorithm for Theorem 4.1

This is the main algorithm for Theorem 4.1. It uses Algorithm 2 as a subroutine, and then checks if the 'paths' of non-repeats between repeat clusters is realizable in the linear genome model. If so, these paths are reduced to adjacencies, and the algorithm searches for an Eulerian tour in the remaining instance. We use the term 'Eulerian tour' loosely here- while the idea is still to make sure that there is a tour on the set of vertices that uses every adjacency, depending on the genome model being used, we may have to add an auxiliary vertex and multiedges to create an augmented graph in which we check for an Eulerian tour.

## A.3 Algorithm for Theorem 5.1

This is the algorithm for Theorem 5.1, which optimizes over the set of minimal repeat spanning intervals in order to obtain an instance which is realizable in the mixed genome model. The **max_b_matching** $(\cdot)$ routine specified here is the algorithm described to get a maximum weight $b$-matching, as defined in Definition 3.3. The corresponding algorithm is provided in Section 3.4.

**Algorithm 2** **convert_rsi** $(H, \mu)$: Algorithm for replacing repeat spanning intervals by paths in an instance with repeats in repeat spanning intervals disjoint from other intervals.

```
Input Instance $(H, \mu)$, $H = (V, E)$, repeats contained in repeat spanning intervals not
    contained in any other intervals, a genome model $\mathcal{G}$.
Output An instance $(H', \mu')$, $H' = (V, E')$, without repeat spanning intervals which is
    realizable in the genome model $\mathcal{G}$ if and only if $(H, \mu)$ is realizable in $\mathcal{G}$.
    $H' = (V', E') \leftarrow H = (V, E_A)$, $\mu' \leftarrow \mu$
 2: $dict(e) = 0$ for all $e \in E_A$.
    for each repeat spanning interval $e \in E_{rsi}$, $o(e) = u.\ r_0.\ \ldots.\ r_{k-1}.\ v$ do
 4:     for $i \in [k+1]$ do
            $V' \leftarrow V' \cup \{r_{e,i}\}$                                ▷ Adding 1 copy of $r_i$
 6:         $\mu'(r_{e,i}) \leftarrow 1$, $\mu'(r_i) \leftarrow \mu'(r_i) - 1$               ▷ Decreasing multiplicity
            if ($r_i$ is unoriented and $\mathbf{deg}_{H'}(r_i) > 2\mu'(r_i)$) or
 8:           ($r_i$ is oriented and $\mathbf{deg}_{H'}(r_i) > \mu'(r_i) + 1$) then
                for all adjacencies $e' = \{u, r_i\}$, $u \neq \overline{r}_i$ if $r_i$ is oriented do
10:                 if $dict(e') = 1$ then
                        $E' \leftarrow E' \setminus \{e'\}$
12:                 end if
                end for
14:         end if
            if $\mu(r_i) < 0$ then return $(H, \mu)$ is not realizable in $\mathcal{G}$. ▷ Too many copies used
16:         end if
            if $0 < i < k$ then                      ▷ Add path corresponding to walk for $e$
18:             $E' \leftarrow E' \cup \{\{r_{e,i}, r_{e,i-1}\}\}$
                $dict(\{r_i, r_{i-1}\}) \leftarrow 1$                ▷ Keeping track of compatible adjacencies
20:         else if $i = 0$ then
                $E' \leftarrow E' \cup \{\{r_{e,0}, u\}\}$
22:             $dict(\{r_0, u\}) \leftarrow 1$
            else if $i = k - 1$ then
24:             $E' \leftarrow E' \cup \{\{r_{e,k-1}, v\}\}$
                $dict(\{r_{k-1}, v\}) \leftarrow 1$
26:         end if
        end for
28: end for
    return $(H', \mu')$
```

**Algorithm 3** Algorithm for deciding if an instance is realizable if repeats are only in repeat spanning intervals or adjacencies.

```
Input Instance $(H, \mu, w)$, $H = (V, E_A \cup E_I)$ connected, repeats only in adjacencies and
    repeat spanning intervals.
Output Is $(H, \mu)$ realizable in the genome model $\mathcal{G}$?
    $H = (V', E')$, $V' = \emptyset, E' = \emptyset$
 2: $(H', \mu') \leftarrow$ convert_rsi $(H, \mu)$
    $H'' = (V'', E'') \leftarrow H' = (V', E'), \mu'' \leftarrow \mu'$
 4: for each connected component $H'_c = (V'_c, E'_c)$ induced by non-repeats in $(H', \mu')$ do
        $V_c \leftarrow V'_c \cup \{r_{c,u} : r \in V'$ such that $\mu'(r) > 1$ and $r$ adjacent to $u \in V'_c\}$
 6:     $\phi_c(r_{c,u}) \leftarrow r \; \forall \; r_{c,u} \in V_c$
        $E_c \leftarrow E'_c \cup \{\{u, r_{c,u}\} : \forall \{u, r\} \in E', u \in V'_c\}$
 8:     $H_c = (V_c, E_c)$
        $\mu_c(v) \leftarrow 1$ for all $v \in V_c$
10:     if $((H_c, \mu_c)$ is realizable in the linear genome model$)$ then
            $V'' \leftarrow V'' \cup \{v_c\} \setminus V'_c$
12:         $\mu''(v_c) \leftarrow 1$
            $E'' \leftarrow E'' \setminus E_c$
14:         for all $r_{c,u} \in V_c \setminus V'_c$ do
                $E'' \leftarrow E'' \cup \{\phi_c(r_{c,u}), v_c\}$
16:         end for
        else
18:         return $(H, \mu, w)$ is not realizable in genome model $\mathcal{G}$
        end if
20: end for
    return check_realizability $((H'', \mu''), \mathcal{G})$    ▷ Decision algorithm for adjacencies only
```

**Algorithm 4** Algorithm for optimizing over the set of minimal repeat spanning intervals when the non-repeats contained in them are oriented.

```
Input Realizable instance $(H_A, \mu, w)$, $H_A = \left(V^h \cup V^t \cup V^u, E_A\right)$ in the mixed genome
    model, minimal repeat spanning interval set $E_I$, with oriented framing vertices.
Output Maximum weight subset $S \subseteq E_I$ such that $(H', \mu, w')$, $H' = (V, E_A \cup S)$, is real-
    izable in the mixed genome model.
    $G' = (V, E') \leftarrow H_A = (V, E_A)\,, \mu' \leftarrow \mu, w'' \leftarrow w_A$
    $S \leftarrow \emptyset$, $D \leftarrow \emptyset$
    $edge\_dictionary \leftarrow \emptyset$                    ▷ Keeps track of new edges added.
    for each repeat spanning interval $e \in E_I$, framed by non-repeats $u, v$ do      ▷ Order is
    fixed
        $E' \leftarrow E' \cup \{\{u,\ v\}\}$                         ▷ Adding representative edge
        $edge\_dictionary\,(\{u, w\}) \leftarrow e$                       ▷ Label new edge with $e$
        $w''\,(\{u, v\}) \leftarrow w\,(e)$
        if $o\,(e) = u.\ r.\ .\overline{r}.\ v$ then
            $\mu'\,(r) \leftarrow \mu\,(r) - 1$, $\mu'\,(\overline{r}) \leftarrow \mu\,(\overline{r}) - 1$                ▷ Decrease multiplicity
            $D \leftarrow D \cup \{\{u, r\}\,, \{v, \overline{r}\}\}$
        else if $o\,(e) = u.\ r.\ v$ then
            $\mu'\,(r) \leftarrow \mu\,(r) - 1$                                 ▷ Decrease multiplicity
            $D \leftarrow D \cup \{\{u, r\}\,, \{r, v\}\}$
        end if
    end for
    $E' \leftarrow E' \setminus D$                                        ▷ Remove extra edges
    for $e \in E' \cap E_A$ do
        $w''\,(e) \leftarrow 1 + \sum_{e' \in E_I} w\,(e')$        ▷ Give high weights to remaining edges from $E_A$
    end for

    $G'' = (V, Q \subseteq E') \leftarrow$ max_b_matching $(G', \mu', w'')$          ▷ b-matching algorithm
    $S \leftarrow \{edge\_dictionary\,(e) :\ \forall\ e \in Q \setminus E_A\}$
    $H' = (V, E_A \cup S)$
    return $(H', \mu, w')$
```

# Appendix B

# Some concepts and results from computational complexity

## B.1 Definitions and fixed parameter tractability

### B.1.1 Conjunctive normal form

In Chapter 6, we referred to a CNF formula. We provide the definition of the same here.

**Definition B.1.** Let $\mathcal{X} = \{x_0, \ldots, x_{n-1}\}$ be a set of $n$ boolean variables. A boolean formula $\Phi\colon \mathcal{X} \to \{0, 1\}$ is said to be specified in the *conjunctive normal form* (CNF) if it is specified in the following structure.

$$\Phi(\mathcal{X}) = C_0 \wedge C_1 \wedge \ldots \wedge C_{m-1},$$

where each $C_i = \left(l_{j_0} \vee \ldots \vee l_{j_{k_i}}\right)$, $k_i \in \mathbb{N}$, is a clause, and each $l_j$ is a literal of a variable in $\mathcal{X}$.

Determining if there is a satisfying assignment for a formula in CNF is usually NP-complete [197].

### B.1.2 Parameterized complexity

The concept of parameterized complexity was proposed by Downey and Fellows [69,70,71]. We can define a *problem* as a set $Q \subseteq \Sigma^*$, where $\Sigma$ is a finite alphabet. For any string $x \in Q$, we define $|x|$ to be the length of $x$. The question we ask is how we can recognize a string $x \in Q$, preferably using an efficient algorithm. Downey and Fellows defined the notion of a parameterized problem as follows.

**Definition B.2.** Let $Q \subseteq \Sigma^*$ be a problem.

1. A *parameterization* of $\Sigma^*$ is a polynomial time computable function $\kappa\colon \Sigma^* \to \mathbb{N}$.
2. A *parameterized problem* over $\Sigma$ is a pair $(Q, \kappa)$, where $Q \subseteq \Sigma^*$ is a set of strings (a problem) over $\Sigma$, and $\kappa$ is a parameterization of $\Sigma^*$.

Based on this definition, they gave the following definition for a *fixed parameter tractable* problem.

**Definition B.3.** [70]

1. An algorithm with an input alphabet $\Sigma$ is said to be *fixed parameter tractable* with respect to a parameterization $\kappa\colon \Sigma^* \to \mathbb{N}$ if there exists a computable function $f\colon \mathbb{N} \to \mathbb{N}$, and there exists a polynomial $p \in \mathbb{Z}[X]$, such that, for every string $x \in \Sigma^*$ in a language $Q$ given as input, the algorithm terminates in time $O\left(f\left(\kappa\left(x\right)\right)p\left(|x|\right)\right)$.

2. A problem $Q \subseteq \Sigma^*$ is said to be *fixed parameter tractable* (FPT) with respect to the parameterization $\kappa\colon \Sigma^* \to \mathbb{N}$ if there is a fixed parameter tractable algorithm with respect to $\kappa$ that decides $Q$.

As a counterpoint, Downey and Fellows also introduced the notion of *parameterized complexity* [71]. This is defined through a hierarchy of complexity classes, called the W-hierarchy of complexity classes. The hierarchy is given as follows.

$$FPT = W[0] \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[P],$$

where the inclusions are conjectured to be strict. The W-hierarchy is defined using circuit complexity, and we do not mention the exact definition here. It suffices here to know that problems in $W[1]$ itself are considered parameterized intractable.

## B.2 The hardness of 3SAT(2,2)

This section provides a proof for Lemma 6.1, attributable to Ján Maňuch. The proof follows a reduction from 3SAT.

*Proof.* First, note that 3SAT(2,2) is in NP by virtue of the fact that, given a satisfying assignment for a CNF formula $\Phi$ which is in the form described by 3SAT(2,2), it is easy to check if $\Phi$ outputs `True` for that assignment in polynomial time, by simply evaluating every clause.

Now, consider an instance $\Phi = (\mathcal{X}, \mathcal{C})$ of 3SAT, where $\mathcal{X}$ is a set of $n$ variables, and $\mathcal{C}$ is a set of $m$ 3-clauses on these variables. We will construct an instance $\Phi' = (\mathcal{X}', \mathcal{C}')$ of 3SAT(2,2) which is satisfiable if and only if $\Phi$ is satisfiable.

Initialize $\mathcal{X}' = \mathcal{C}' = \emptyset$. For each variable $x \in \mathcal{C}$, such that $x$ has $\kappa(x)$ occurrences in $\Phi$, introduce $\kappa(x)$ variables $x_0, \ldots, x_{\kappa(x)-1}$ in $\mathcal{X}'$. To create the clauses in $\mathcal{C}'$, first take all the clauses in $\mathcal{C}$, arbitrarily ordered, and replace the $i^{th}$ occurrence of a variable $x \in \mathcal{X}$ by the variable $x_i$ instead, keeping the sign of the variable the same, i.e. positive literals stay positive, and negative literals stay negative. Add the modified clauses to $\mathcal{C}'$.

The second set of clauses to add impose a constraint making sure that all $x_i \in \mathcal{X}'$ introduced for a single variable $x \in \mathcal{X}$ have the same value in a satisfying assignment. For each $x_i \in \mathcal{X}'$, add variables $f_{x,i}$ to $\mathcal{X}'$. Then, add clauses $(\neg x_i \vee x_{i+1} \vee f_{x,i})$ for $i \in [\kappa(x)]$ to $\mathcal{C}'$, with addition in the indices being done modulo $\kappa(x)$. Finally, for each $f_{x,i}$, add a variable $p_{x,i}$ to $\mathcal{X}'$, and add two clauses $(p_{x,i} \vee p_{x,i} \vee \neg f_{x,i})$ and $(\neg p_{x,i} \vee \neg p_{x,i} \vee \neg f_{x_i})$ to $\mathcal{C}'$.

Note that, in the previous construction, the chain of clauses $\bigwedge_{i \in [\kappa(x)]} (\neg x_i \vee x_{i+1} \vee f_{x,i})$ is satisfiable if $f_{x,i} = 0$ if and only if $x_i = x_{i+1}$ for all $i \in [\kappa(x)]$. Also, the clause pair $(p_{x,i} \vee p_{x,i} \vee \neg f_{x,i}) \wedge (\neg p_{x,i} \vee \neg p_{x,i} \vee \neg f_{x,i})$ both evaluate to `True` if and only if $f_{x,i} = 0$. Thus, all the $x_i$ are guaranteed to have the same truth assignment.

In order to get an instance of 3SAT(2,2), we need to make sure that every variable has 4 occurrences, twice as a positive literal, and twice as a negative literal. Each $x_i$ currently has 3 occurrences, so it is missing either a positive occurrence or a negative occurrence. Each $f_{x,i}$ is missing a positive occurrence. For each $x_i$, add a variable $q_{x,i}$ to $\mathcal{X}'$. Then, if $x_i$ is missing a positive occurrence, add the clause $(q_{x,i} \vee \neg q_{x,i} \vee x_i)$ to $\mathcal{C}'$. Conversely, if it is missing a negative occurrence, add the clause $(q_{x,i} \vee \neg q_{x,i} \vee \neg x_i)$. Finally, add the clauses $(q_{x,i} \vee \neg q_{x,i} \vee f_{x,i})$ to $\mathcal{C}'$. The total number of variables introduced is $12|\mathcal{C}|$, and the total number of clauses introduced is $16|\mathcal{C}|$.

To prove the claim, note that by the construction given above, all $f_{x,i}$ must have negative value in order for $\Phi'$ to be satisfied, and that the values of the $p_{x,i}$'s do not matter, since the $f_{x,i}$'s are all 0. Similarly, the values of the $q_{x,i}$'s do not matter, since both literals of $q_{x,i}$ appear in a clause. So, a satisfying assignment for $\Phi'$ fixes the values of the $x_i$'s, which means it fixes a value for each $x \in \mathcal{X}$. Since the clauses in $\mathcal{C}$ are precisely a subset of those in $\mathcal{C}'$, with the literals indexed, these clauses must also be satisfied by the values fixed for each $x \in \mathcal{C}$, giving a satisfying assignment of $\Phi$.

In the opposite direction, by fixing a satisfying assignment of $\Phi$, we assign each $x_i \in \mathcal{X}'$ the value of $x \in \mathcal{X}$ in this satisfying assignment. Then, setting all $f_{x,i}$ to 0 gives us a satisfying assignment for $\Phi'$. This completes the proof.

$\square$

# Appendix C

# Primer on semidefinite programming

This appendix is a short introduction to semidefinite programming and its applications in combinatorial optimization. For a more thorough coverage of the topics discussed here, the reader is referred to Grötschel, Lovász and Schrijver [98], which is also the source text for this exposition.

Let $M$ be an $n \times n$ real, symmetric matrix. It is a well-known fact in linear algebra that the eigenvalues of $M$ are also real. We say $M$ is *positive semidefinite* if, for all eigenvalues $\lambda_0, \ldots, \lambda_{n-1}$ of $M$, we have $\lambda_i \geq 0$. This is equivalent to the following property on $M$.

$$\mathbf{x}^T M \mathbf{x} \geq 0 \quad \forall\, \mathbf{x} \in \mathbb{R}^n.$$

We denote a positive semidefinite matrix $M$ by stating that $M \succeq 0$. Positive semidefinite matrices are often used as the matrix analogs of non-negative real numbers. One such analogy is the extension of linear programming to *semidefinite programming.* Consider the following inner product on two $n \times n$ real matrices $A$ and $B$.

$$A \bullet B = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{ij}.$$

Then, we can define a semidefinite program as follows.

**Definition C.1.** Let $C$ be a symmetric matrix in $\mathbb{R}^{n \times n}$. Let each $A_0, \ldots, A_{m-1}$ be $m$ other $n \times n$ symmetric real matrices, and let $\mathbf{b}$ be a vector in $\mathbb{R}^n$. A *semidefinite program* is defined as the following constraint satisfaction problem.

$$\begin{aligned} &\min_{X \in \mathbb{R}^{n \times n}} C \bullet X \\ &\text{subject to} \\ &\qquad A_i \bullet X = b_i \quad \forall\, i \in [m]\,, \\ &\qquad X \succeq 0. \end{aligned}$$

As in the case of linear programming, the equality in the constraints can be replaced by inequalities. This is one definition of semidefinite programs; there are many equivalent definitions based on the different characterizations of a positive semidefinite matrix. One of these definitions follows from the fact that an $n \times n$ positive semidefinite matrix can be expressed as the Gram matrix of $n$ vectors in some complex vector space. So, each entry $x_{ij}$ can be expressed as $\mathbf{x}_i^T \mathbf{x}_j$ for two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. Using this definition, we can reformulate a semidefinite program as a constraint satisfaction problem in which we need to find $n$ vectors $\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}$.

Semidefinite programs are natural generalizations of linear programs. As they are generalizations, they capture a larger class of constraint satisfaction problems which can be efficiently solved. Indeed, methods for solving linear programs in polynomial time, such as the ellipsoid method and interior point methods, can also be used to solve semidefinite programs. The ellipsoid method, in particular, is well-suited to handling a large number of structured constraints, such as those we deal with. However, while these algorithms can run in polynomial time, they may only solve semidefinite programs up to an arbitrary additive error, on which the complexity depends [98]. Improvements in interior point methods [179] promise extendability to semidefinite programs as well [204], allowing polynomial time solvability with an exponential number of structured constraints.

Semidefinite programs are also important as a useful tool in combinatorial optimization and in designing approximation algorithms. Many problems in optimization may be formulated as non-convex programs, such as quadratic programs.

**Definition C.2.** Let $C, A_0, \ldots, A_{m-1}$ be real symmetric $n \times n$ matrices. A quadratic program is defined as the following constraint satisfaction problem.

$$\begin{aligned} &\min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{x}^T C \mathbf{x} \\ \text{subject to} & \\ &\mathbf{x}^T A_i \mathbf{x} = b_i \quad \forall i \in [m] \,. \end{aligned}$$

Note that $\mathbf{x}^T C \mathbf{x}$ can be rewritten as $Tr\left(C\mathbf{x}\mathbf{x}^T\right)$, where $Tr\left(\cdot\right)$ is the trace of the matrix. From the discussion before, $\mathbf{x}\mathbf{x}^T$ is a positive semidefinite matrix, the Gramian matrix of $n$ 1-dimensional vectors, and $Tr\left(C\mathbf{x}\mathbf{x}^T\right) = \sum_{i=1}^n \sum_j^n c_{ij}.x_i.x_j$, which is precisely the matrix inner product we defined before. So, a quadratic program can be visualized as a semidefinite program, with the positive semidefinite matrix being the Gramian matrix of the $n$ 1-dimensional vectors specified by entries in $\mathbf{x}$. However, this program is non-convex, and cannot be solved in polynomial time.

If we relax the constraint on $\mathbf{x}$, and allow it to be a set of $n$-dimensional vectors, we get a semidefinite program which is solvable in polynomial time. This is the concept behind *semidefinite relaxations*. An especially useful fact is that we can efficiently recover the vectors $\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}$ of which the solution is a Gramian matrix of, using Cholesky decomposition [98]. Thus, instead of getting a vector $\mathbf{x}$, we obtain a set of vectors from the semidefinite relaxation.

The use of semidefinite relaxations in the theory of approximation algorithms gained a lot of support following the work of Goemans and Williamson, who used it to get a constant factor approximation algorithm for the MAX-CUT problem on graphs [95]. Later, it was proved that this is the best possible algorithm, assuming the Unique Games Conjecture [117], and indeed, assuming the conjecture, the best approximation for all constraint satisfaction problems is achieved by a certain simple semidefinite program [176]. As a result, it is one of the standard techniques used to find approximation algorithms [92].

# Appendix D

# Polynomial time algorithms for minimum cumulative stretch

In the following section, we show a simple algorithm for calculating the minimum cumulative stretch on a highly structured hypergraph. The result presented was proved in the course of an ongoing project to study polynomial time algorithms for finding a minimum cumulative stretch order of certain restricted hypergraphs, particularly hypertrees. The project is a collaboration with Cedric Chauve, Ján Maňuch and Arash Rafiey.

**Lemma D.1.** *Let $H = (V, E)$ be a hypergraph such that there is exactly* 1 *vertex $v$ adjacent to all hyperedges $e_0, \ldots, e_{\kappa-1}$, and all other vertices are contained in exactly one edge. The cumulative stretch problem for this hypergraph can be solved in polynomial time.*

We call such an instance a *hyperstar*, or simply a *star*. The vertex $v$ is called the *central vertex* of the star. Note that stars avoid all Tucker patterns except for $G_{III,1}$.

*Proof.* Assume that $|e_i| \leq |e_{i+1}|$ for all $i \in [\kappa - 1]$. If not, we can always relabel the hyperedges. We are going to prove the following claim.

**Claim.** *Let $\sigma(e_i)$ denote the consecutive series of vertices belonging to $e_i$, other than $v$, in*

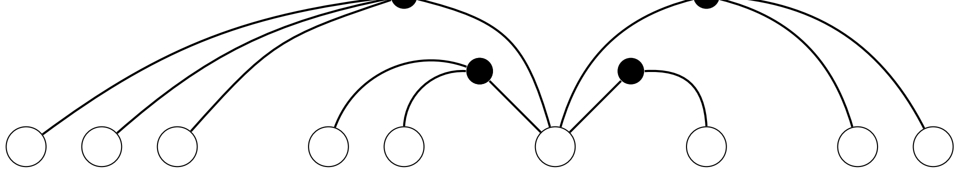


Figure D.1: A hyperstar in its optimal ordering. The white nodes represent vertices, and the black nodes represent hyperedges.

*an order $\sigma$. An optimal ordering $\pi$ has the following structure.*

$$\pi\left(e_{\kappa-1}\right).\pi\left(e_{\kappa-3}\right)\ldots\pi\left(e_0\right).v.\left(e_1\right)\ldots\pi\left(e_{\kappa-4}\right).\pi\left(e_{\kappa-2}\right)\ \ \textit{if}\ \ \kappa\equiv 1 \mod 2, \tag{D.1}$$
$$\pi\left(e_{\kappa-2}\right).\pi\left(e_{\kappa-4}\right)\ldots\pi\left(e_0\right).v.\left(e_1\right)\ldots\pi\left(e_{\kappa-3}\right).\pi\left(e_{\kappa-1}\right)\ \ \textit{if}\ \ \kappa\equiv 0 \mod 2. \tag{D.2}$$

Consider the cost of an order $\pi$ which looks as follows.

$$\pi\left(e_{\sigma^{-1}(0)}\right).\pi\left(e_{\sigma^{-1}(1)}\right)\ldots\pi\left(e_{\sigma^{-1}(i-1)}\right).v.\left(e_{\sigma^{-1}(i)}\right)\ldots\pi\left(e_{\sigma^{-1}(\kappa-2)}\right).\pi\left(e_{\sigma^{-1}(\kappa-1)}\right),$$

where $\sigma$ is a permutation on $[\kappa]$ elements. A hyperedge $e_{\sigma^{-1}(k)}$ to the left of $v$ will contribute to the stretch of all $e_{\sigma^{-1}(j)}$, where $j \leq k$. Thus, its vertices, other than $v$, contribute to the minimum cumulative stretch $k$ times. Extending this argument for both the edges to the left of $v$, and those to the right of $v$, we can calculate the cost of the order as follows.

$$Cost_\pi\left(H\right) = \sum_{j=0}^{i-1}\left(j+1\right)\left(\left|e_{\sigma^{-1}(j)}\right|-1\right) + \sum_{j=i}^{\kappa-1}\left(\kappa-j\right)\left(\left|e_{\sigma^{-1}(j)}\right|-1\right). \tag{D.3}$$

In order to minimize this, we need to allocate the largest edges to the smallest coefficients $j+1$ and $(\kappa-j)$ in the sums, and so on. Note that if the vertices in a single hyperedge (apart from $v$) were not kept together in an optimal vertex ordering, these vertices would contribute to the stretch of other hyperedges, and thus increase the cost.

To see that $i = \lfloor\kappa/2\rfloor$, which would imply the orders specified in (D.1) and (D.2), assume that the size of the hyperedges is a concave continuous function $f(x)$ in the domain $[0,\kappa]$. Then, the *continuous cost* can be computed as follows.

$$C\left(x\right) = \int_0^x \left(u+1\right)f\left(u\right)\ du + \int_x^\kappa \left(\kappa-u\right)f\left(u\right)\ du, \tag{D.4}$$

If we optimize (D.4) with respect to $x$, we find that $x = \left(\kappa-1\right)/2$, which establishes that the hyperedges are equally distributed on the two sides of $v$.

Since this ordering can be found in polynomial time by simply keeping track of the size of the hyperedges $e_i$, this completes the proof of the lemma. □

Assuming $e_0 \leq \ldots \leq e_{\kappa-1}$ are the edges involved in the star, Lemma D.1 also implies the following observations.

1. For any $e_i$, $i$ being an odd integer, $\sum_{t\leq(i+1)/2}\left(|e_{2t-1}|-1\right) > \sum_{t\leq(i-1)/2}\left(|e_{2t}|-1\right)$.

2. For any $e_i$, $i$ being an even integer, $\sum_{t\leq i/2}\left(|e_{2t-1}|-1\right) < \sum_{t\leq i/2}\left(|e_{2t}|-1\right)$.

Furthermore, if we take the substar on the edges $|e_0| \leq \ldots \leq |e_p|$, $p < \kappa - 1$, the order induced on the vertices in these edges by an optimal solution for the minimum cumulative stretch ordering of the star must also be optimal. This is merely a consequence of the iterative method by which we build the optimal star ordering, as indicated in (D.3).

This result is intended to serve as a stepping stone to a polynomial time algorithm for finding a minimum cumulative stretch order of a larger class of hypergraphs, called *hypercaterpillars*.

**Definition D.1.** A hypercaterpillar $H = (V,E)$ is a hypergraph which consists of the following set of hyperedges and vertices.

1. $\{v_0, \ldots, v_n\}$ is a set of $n+1$ *backbone vertices.*
2. $\{e_1, \ldots, e_n\}$ is a set of $n$ *backbone hyperedges*, such that $v_i, v_{i+1} \in e_{i+1}$ for $i \in [n]$, and no other vertices in $e_i$ have degree more than 1 (i.e. they are only contained in that particular hyperedge).
3. For each $i \in [n+1]$, there are hyperedges $\{e_{i,0}, \ldots, e_{i,k_i-1}\}$ where $k_i \in \mathbb{Z}_{\geq 0}$, which contain $v_i$, and the only other vertices in each $e_{i,j}$ have degree 1.

The number of backbone vertices is called the *length* of the hypercaterpillar. The maximum number of non-backbone hyperedges containing a backbone vertex is said to be the *width* of the caterpillar.

A hypercaterpillar avoids all Tucker patterns except for $G_{III,1}$. It may be visualized as a chain of stars, with two stars being connected at the central vertex by a hyperedge. An algorithm, or the absence of one, for the minimum cumulative stretch problem on hypercaterpillars, in turn, may provide some insight into the tractability of the problem on hypertrees.