Algorithmic Methods
for
Reconstruction
of
Biological Sequences, Gene Orders and Maps

THESIS SUBMITTED FOR THE DEGREE OF
"DOCTOR OF PHILOSOPHY"

by
Itsik Pe'er

The work on this thesis has been carried out
under the supervision of **Prof. Ron Shamir**

Submitted to the Senate of Tel-Aviv University
January 2002

Contents

Acknowledgements

The PhD is more than research work. It is a chapter in one's life. While summarizing the scientific contributions in this thesis, I would like to take this opportunity to acknowledge all who accompanied me through this period of learning, helping in ideas, spirit, data and budgets. The list is long, but you all deserve my deepest gratitude.

This work is dedicated to my better half, Dana Pe'er, without which it would have never been completed. Thank you for pushing me through the frustrating periods, and for loving me in my worst moments. Thank you also for countless insightful comments on virtually all my research projects.

I thank my daughter Inbar, that has been not only a source of strength and encouragement but also inspired some of the ideas and starred in my presentations. I wish to thank my canine sister-soul, Bat-Sheva, my favorite distraction in the long nights of work. I thank my parents, wise people who gave me the chance they never had to study more and more. I know how important this degree has always been to you, but I think you don't appreciate how essential to its completion was the motivation to excel and to wisen, a drive that you have given me. Dad, I wish you could read this.

I am in great debth to my mentor, Ron Shamir. Ron, in a decade of working under your guidance, you have never ceased to surprise me with the marvelous things you can do. Thank you for all the skills, ideas, and support you have given me.

Much of this work would not have been done without Tal Pupko, a true friend and a close collaborator. Tal, your scientific insight and fruitful thought have never failed to drive us to new horizons, and your company made this so much fun, too. I deeply thank Roded Sharan, my research-mate during all this period. Roded, I admire your ability, your stamina, and your wonderful approach to life. I thank you for sharing these and will always cherish the hours we spent brainstorming at the white-board.

I would like to thank all my colleagues and co-authors: Prof. Richard Karp,

a great scientist with whom I'd been priveledged to collaborate and find out he is even more impressive in work than in titles; Danny Graur, the funniest person on campus, whose scientific vision is the reason this work may be of any value to biology; Nir Friedman, a role model for scientific dedication, and source of infinite ideas; Amir Ben-Dor, with whom both scientific and social discussions have been a pleasure; Matan Ninio, and Naama Arbili, who is doing a great work, not only in research, but also in bringing a fresh spirit to our lab.

Lots of thanks to all my lab mates at the Computational Genomics group, and to my fellow research students in the School of Computer Science. Special thanks to Zipi Fligelman, a true friend I have acquired during this PhD, sharing our thoughts on research and life in general. I am leaving the Department of Computer Science at Tel Aviv after three degrees and fifteen years, which is the adult half of my life. This place have been a second home for me. I thank the department staff, especially Prof. Haim Wolfson, my surrogate supervisor, and Hagit Hen, whose administration made bureaucracy workable.

I thank Thomas Anantharaman, Bud Mishra and David Schwartz from N.Y. University for providing us with experimental data of Optical Mapping, and analyzing our results. Thanks also to Richard Durbin from the Sanger Centre, for a wonderful scienntific visit there.

I wish to thank all the funding agencies, that enabled this work. I greatly thank the Clore foundation, and Mrs. Vivian Clore-Duffield, for the generous scholarship they awarded me. This support allowed me to focus on research, providing me with optimal working conditions during the last three years. I thank the Ministry of Science, Israel, for the Eshkol fellowship given to me at initial stages of the research. I thank the Planning and Budgeting Committee of the Higher Education Council in Israel, for additional financing, and the Deutch foundation for their travel support.

Chapter 1

Introduction

In this chapter we first describe basic notions in biology that are used in this thesis. Then we describe probabilistic approaches to modeling sequences. The last section summarizes the thesis results.

1.1 A Primer on Molecular Sequences

This work involves application of algorithmic techniques to the domain of molecular biology. While the computational methods developed constitute novel algorithms, their target material are biological problems, and full appreciation of their contribution requires understanding this target domain. This introduction is meant to make this work accessible to a general computer science audience. We will thus try to explain the essentials of molecular genetics in brief. Inevitably, there are many omissions and oversimplifications in this presentation. The interested reader is referred to basic literature in this field, c.f. Lewin [1997].

Organisms and cells carry molecular information from generation to generation. The molecule responsible for storing this information is called *DNA*. Generally speaking, DNA serves as a program for the cell to construct and operate molecular machinery. The DNA macro-molecule is a *polymer*, that is, a long linear chain of small molecular building blocks. These units of DNA are called nucleotides.

There are of four kinds of such DNA units, which are usually denoted A,C,G, and T. The DNA strand can therefore be represented as a sequence of these letters. In computer science we model this sequence as a string over this four-letter alphabet which contains the genetic information.

In its usual state the DNA is *double stranded*. That is, it comprises of two DNA strands that run anti-parallel to each other. Each nucleotide in one strand is chemically bounded, or *hybridized*, to a corresponding nucleotide on the other strand. These bonds follow a strict pairing rule: A pairs with T while G pairs with C. The second strand is thus a complementary template of the first one. Nature exploits this structure to copy the information embedded in the DNA, both in order to replicate it for future cellular generations, and to read its instructions for constructing cellular machinery. Furthermore, hybridization pairing of DNA molecules is an essential principle used in many molecular experiments in biology.

The DNA sequence of all organisms of a certain species is roughly the same. The master DNA sequence of a species is termed its *genome*. Determination of the genome sequence, termed *sequencing*, is one of the major tasks and accomplishments in contemporary molecular biology. The crown jewel is the recent completion of the 3×10^9 base-pair long human genome, one of the greatest feats of science [Lander et al., 2001, Venter et al., 2001].

Current sequencing machinery can only read short DNA molecules, of few hundred base-pairs. To read the sequence of genomic target segments that are much longer, the DNA is shreaded into short pieces. Each piece is read separately, and the short sequences are then assembled together computationally. This fragmentation and assembly process is greatly aided by the ability to locate molecular landmarks along the target DNA, i.e., having a *map* of the genomic target. Therefore, mapping is often a primary stage in sequencing a long genomic target.

Most of the DNA of higher organisms has no known function. It is sometimes called *non-coding*, or *"junk" DNA*. In contrast, coding sequence segments are known to encode instructions that directly control cellular machinery. Specifically, they encode proteins, which are polymers responsible for virtually all organic activity. These polymers are composed of building blocks called *amino acids*. The encoding of proteins by DNA is known as the *genetic code*. This code is a

simple mapping, that assigns one of the 20 existing *amino-acids* to each of the $4^3 = 64$ possible triplets of nucleotides. Translation of a gene entails reading non-overlapping triplets, called *codons*, and mapping each to an amino-acid. The resulting linear chain of amino-acids is the protein. In simple life forms genes are continuous stretches of DNA, while in higher organisms a typical gene is composed of several continuous segments, called *exons*, interleaved by *introns*, sequences which do not code for protein.

DNA is replicated and inherited when cells divide. This replication, however, is not perfect. The errors in replication are called *mutations*. Such mutations are usually local, i.e., involve a change in sequence only in a specific position along the genome. The change may be replacement of a specific nucleotide by another, deletion, or insertion of nucleotides. Mutations which are not local include deletion, insertion, inversion, or duplication of long genomic fragments.

All these differences in sequence are the molecular source of hereditary variation between different individuals of the same species. In fact, mutations are also responsible for practically all genetic diversity in nature: The origin of all species is thought to be unique, and thus all contemporary genomes have a common ancestor sequence. Furthermore, even within the same genome one finds duplicated segments that gradually varied by mutation-prone replication.

1.2 Probabilistic Approach to Mutations

Due to the importance of mutations in shaping genomic landscape, their analysis has attracted extensive research. Mutations occur randomly. Detailed characterization of the stochastic process of their occurance has broad applications in comparison of diverged sequences and identification of their origin.

Basic models of sequence change consider only the most common kind of mutations: substitution of a single nucleotide. The simplest such model assumes that at each point in time, each nucleotide along the sequence has a fixed probability of being replaced by any other nucleotide [Jukes and Cantor, 1969]. A more elaborate model of DNA evolution [Kimura, 1980a] distinguishes between replacements of a

nucleotide by a nucleotide of roughly the same shape (A by G, or C by T, and vice versa), versus replacements by nucleotides of a different shape. The biochemical procedure through which mutations occur dictates that the former replacements will take place more frequently. A general framework for describing the stochastic process of nucleotide replacement [Felsenstein, 1981] uses a 4×4 matrix, that contains the probabilities of change of each nucleotide into each nucleotide.

When a mutation occurs in an individual member of some population, we say that the mutated position is now *polymorphic*. Specifically, the mutations discussed thus far are *Single Nucleotide Polymorphisms*. After many generations, the mutations may either become extinct from the population, or become inherited by all living members. In the latter case, we say that the mutation is *fixed*.

When a mutation is incorporated into a gene, it may change the protein encoded by that gene, replacing one amino-acid by another, and in some cases truncating the protein. This is the major effect of mutations on cellular processes. A mutation that dramatically changes the protein is usually harmful to the organism, and thus its chances of fixation are very small. These chances therefore depend on the difference between the replaced amino-acids, and suggest models for evolution of coding sequences that consider replacements of amino-acids rather than nucleotides. The simplest models of this kind assign probabilities to the replacements of each amino-acid by another, and register them in a 20×20 matrix [Dayhoff et al., 1978, Jones et al., 1992, Adachi and Hasegawa, 1996]. Different sequence positions are assumed to change independently according to the matrix probabilities. A natural refinement considers sequences over an alphabet of codons, and models them using a 64×64 matrix of probabilities [Goldman and Yang, 1994].

All mutation models mentioned thus far discuss pairwise substitution of characters in some alphabet. However, in addition to substitutions, insertions or deletions also occur in biological sequences. When a deletion takes place, a character, or a sequence of characters, in the original sequence is replaced by a *gap* in the mutated one. The converse is true for insertions. A naive approach to mutations would simply extend the substitution matrix to the "gap" character. However, in this case the assumption of independence among sequence positions miserably fails: the gap characters tend to occur consecutively. In other words, the proba-

bility of a gap being *extended* is much larger than its probability of being *opened*. The log-probability of observing a gap is thus an affine function of the gap length. Differentiating these two probabilities has become standard in biological sequence comparison [Smith and Waterman, 1981].

The affine gap model can be discussed in a probabilistic framework for generation of related sequences. Suppose that such sequences are generated by a randomized state-automaton. For every sequence position, this automaton randomly outputs a character (possibly the gap character) for each of the generated sequences. The randomized output, or *emission* function, depends solely on the current automaton state. Subsequent to emitting output, the automaton changes its state, or performs a *state transition*. The transition function is also randomized and state-dependent. The strings composed by the automaton throughout its operation constitute the related sequences. Such state-machine models are collectively called *Hidden Markov Models (HMMs)*. An example of a simple such HMM can model sequence similarity without any gaps. This similarity is modeled by single "match/mismatch" state, that emits pairs of related characters. The similarity of characters in these pairs is described by a fixed substitution matrix. To model affine gaps between two compared sequences, Durbin et al. [1998a] present a 3-state HMM, which is more expressive than the single state example. In addition to the match/mismatch state previously described, this model has two symmetric insertion/deletion states, each modeling a gap in one of the compared sequences. Such states emit a pair of characters one of which is the gap character. The log-probability of observing two sequences as output of this automaton is the affine-gap measure for comparing these sequences.

A collection of proteins that have similar function and similar phylogenetic origin is called a *protein family*. Sequences of proteins from the same family are related. When trying to characterize such sets of sequences, there is a need for using state-machine models, which are more elaborate than discussed thus far. Recent such models try to capture dependencies between several sequence characters at nearby positions [Bejerano and Yona, 1999]. Another important class is *profile HMMs*. These introduce a match/mismatch state, a deletion state, and an insertion state for each position along the representative sequence of the protein

family [Krogh et al., 1994]. This kind of models have become standard in protein classification.

1.3 Summary of the Thesis Results

This thesis studies various algorithmic problems in genomics. They arise when trying to reconstruct biological sequences, maps, or gene orders. Each chapter deals with a different sub-field of genomics, and thus can be read independently of the others. We now outline the organization of the chapters and summarize the main results in each chapter. In this section we cite only publications of thesis results. Complementary references are cited in the chapters themselves.

Chapter 2 concerns evolution of related biological sequences from their common ancestor. During evolution, ancestral sequences diverge into separate lineages in a tree-like fashion. Branches of this tree represent periods of time during which sequences underwent mutations. Nodes stand for points in history where lineages were split. The ancestral sequences that were present at such points have special importance in understanding biological function and sequence evolution. Unfortunately, such sequences are usually not directly available to us. Rather, they need to be inferred, or *reconstructed* from the set of their contemporary descendant sequences. We study this task, called *ancestral reconstruction*.

Our input data consists of a set of contemporary sequences, the tree of evolution, and, loosely speaking, a model for the evolutionary process. Our goal is to find the most likely set of ancestral sequences. The reconstruction of these ancestral sequences naturally depends on the assumptions made regarding the process of evolution. Simpler models of evolution assume a fixed rate of mutations in all sequence positions. For such models we devise and implement a polynomial algorithm for ancestral reconstruction. These results have been published in [Pupko et al., 2000]. We also develop a branch and bound algorithm for ancestral reconstruction when the rate of evolution varies. The implementation of our algorithms and their application to real data have been performed in collaboration with Tal Pupko, then at the Department of Zoology in Tel Aviv. We present experimental data for both algorithms.

In Chapter 3 we study algorithmic aspects of sequencing. We improve an existing method, called Sequencing By Hybridization (SBH), which we now outline. SBH is based on interrogating a target DNA molecule for all of its k-long subsequences, called k-*mers*. This is made possible by recent technologies, that immobilize many thousands of short single-stranded molecules of DNA, or *oligonucleotides*, on a small surface, called a *microarray*. Each oligonucleotide probes the target sequence for presence of its complementary counterpart. SBH calls for employing all possible k-long oligonucleotides, thereby obtaining the k-mer contents of the sequence. Subsequently, the target sequence is computationally reconstructed.

Unfortunately, SBH has very limited practicality. In Chapter 3, we introduce ways to enhance SBH by computationally using additional information.

One kind of information we use is additional data regarding the position of k-mers along the target molecule. These data can be obtained by novel microarray reactions. The input to our computational problem is a set of possible occurrence positions for each k-mer. The output is a target sequence all of whose k-mers satisfy the positional constraints. We provide a polynomial algorithm for this problem when no k-mer has more than two possible positions. We prove NP-hardness of the problem, even when no k-mers has more than three possible positions. This work has been published in [Ben-Dor et al., 1999]. We further study a biologically motivated restriction of this problem, where all sets of possible positions are intervals. We provide a parameterized algorithm to solve this restricted problem, which imeroves over a prior algorithm. This part has been published in [Ben-Dor et al., 2001a], with a complete version in [Ben-Dor et al., 2001b].

We study another method to enhance SBH. This method uses additional data which is usually available, and it can therefore be widely applied. Most contemporary tasks of sequence determination involve resequencing rather than sequencing. Hence, one has prior information regarding the target sequence. One knows in advance, that the sequencing target highly resembles an already sequenced molecule. In fact, there are only a few small differences between these two sequences. It is these small differences that are of interest. We formalize this challange as a computational problem, and provide a polynomial algorithm for it.

Our framework is probabilistic. We describe the resemblance to a known sequence as a profile HMM. We further describe results of the SBH reaction in probabilistic terms, and incorporate them into a graph whose edges correspond to probes and vertices to probe prefixes/suffixes of length $k-1$. The probabilistic formulation allows maximum likelihood analysis. It gives rise to an interesting optimization problem: finding the most likely sequence, which corresponds to a pair of paths: one path in the HMM transition graph, and another in the SBH graph. We can therefore solve an analogous optimization problem, finding a pair of paths with maximum weight. We provide several dynamic programming algorithms to solve this problem and some of its interesting restrictions. We provide a practical solution, although it is only guaranteed to be an approximate maximum-likelihood sequence. Simulations results are also provided. This work has been published in [Pe'er and Shamir, 2000b].

Chapter 4 concerns *Optical Mapping*: a revolutionary method for obtaining maps of enzymatic cleavage sites along large DNA molecules. The method is based on immobilization of many copies of the target molecule in elongated form. Enzymes that perform site-specific cleavage (*restriction*) are then applied, and each cleaved, fluorescent molecule is then photographed, to detect cleavage sites. Unfortunately, the orientation of each photographed molecule is unknown. Furthermore, the reported list of cleavage sites suffers from inaccurate locations, false positives and false negatives. The computational problem is to reconstruct the true map of cleavage sites from this noisy data. We define a maximum-likelihood formalization of this problem, and devise an appropriate algorithm. We implement our methods and demonstrate performance on a blind test using real life data. This study has been published in [Karp et al., 1999, 2000].

In Chapter 5 we discuss changes in gene order between genomes during evolution. These changes can be measured by the *breakpoint distance* between gene orders: the number of gene pairs adjacent in one but not the other. For three contemporary gene orders, the ancestral gene order in their evolutionary divergence point is called their *median*. A fundamental task is to find this median given the contemporary gene orders, so that its sum of (breakpoint) distances to the leaves is minimum. We prove this problem to be NP-hard [Pe'er and Shamir, 1998]. We

further provide polynomial time approximation algorithms that guarantee a $\frac{7}{6}$ approximation. This study has been published in [Pe'er and Shamir, 2000a].

Chapter 2

Reconstruction of Evolutionary Ancestral Sequences

This chapter concerns evolution of related biological sequences from their common ancestor. During evolution, ancestral sequences diverge into separate lineages in a tree-like fashion. Branches of this tree represent periods of time during which sequences underwent mutations. Nodes stand for points in history where lineages were split. The goal is to infer, or *reconstruct*, the ancestral sequences from their contemporary descendant sequences. We study this task, called *ancestral reconstruction*.

Our input data consists of a set of contemporary sequences, the tree of evolution, and, loosely speaking, a model for the evolutionary process. Our goal is to find the most likely set of ancestral sequences. The reconstruction of these ancestral sequences naturally depends on the assumptions made regarding the process of evolution. Simpler models of evolution assume a fixed rate of mutations in all sequence positions. We devise and implement a polynomial algorithm for ancestral reconstruction under this model. These results have been published in [Pupko et al., 2000]. We also describe a branch and bound algorithm for ancestral reconstruction when the rate of evolution varies. The implementation of our algorithms and their application to real data have been performed in collaboration with Tal Pupko, then at the Department of Zoology in Tel Aviv. We present experimental

data for both algorithms.

2.1 Introduction

The understanding that many biological sequences share a single common origin is fundamental to biology. Such a set of contemporary sequences diverged from their ancestor sequence in a tree-like fashion. In this work, we assume the topology of this *phylogenetic tree* is known in advance. Branches, or edges, of this tree represent periods in which a sequence accumulated mutations, while tree leaves correspond to contemporary sequences. Internal nodes represent events of sequence divergence. At these points, a single ancestral sequence separated into two independent lineages. It is these ancestral sequences that we wish to study.

Ancestral sequences attract attention because of their position in evolutionary crossroads: Divergent sequences often have differentiated functions, and thus the identity and function of the ancestral sequences are interesting. Several authors used such sequences to provide insights regarding evolutionary processes. For example, Zhang et al. [1998] inferred ancestral sequences of mammalian ribonuclease proteins, in order to detect evidence for positive Darwinian selection in the primate lineage. In [Pupko, 2000], sequences of ancestral vertebrates were analyzed in order to study parallel evolution in lysozymes.

The ancestral sequences are usually not explicitly known. Rather, they need to be inferred from contemporary sequences. The methods for evaluating these sequences vary in complexity, and depend on one's assumptions regarding the process of evolution. The simplest such method is parsimony. It seeks the set of ancestral sequences, that imply a minimum number of mutations over the entire tree. Sankoff [1975] and Fitch and Farris [1974] provided a polynomial algorithm for finding this most parsimonious set of sequences.

Parsimony handles all mutations equally. Taking a statistical viewpoint, this would have been appropriate only if mutations along each branch of the tree were equiprobable. However, some branches and lineages exhibit more mutations than others, and thus a more educated model of evolution takes into account a *length* of

each branch, which corresponds to the rate of mutations along it [Jukes and Cantor, 1969].

Even along the same branch, different mutations may occur with different frequencies. The probability of an amino acid x being replaced by an amino acid y greatly depends on x and y. For a unit-long branch, these probabilities are arranged in a 20×20 matrix [Dayhoff et al., 1978, Jones et al., 1992, Adachi and Hasegawa, 1996]. For other branch lengths, the appropriate probabilities can be calculated by exponentiation of the matrix [Felsenstein, 1981]. When DNA sequences are considered, similar arguments motivate 4×4 matrices that register the nucleotide substitution frequencies [Kimura, 1980b].

The description of evolution in terms of a stochastic process gives rise to maximum-likelihood methods [Felsenstein, 1981]. In relation to ancestral reconstruction, two problems arise [Yang et al., 1995]:

1. **Marginal Reconstruction:** Given a specific tree node, what is the most likely sequence at this node?
2. **Joint Reconstruction:** What is the set of most likely ancestral sequences, in all ancestral nodes?

Marginal Reconstruction concerns one distinct ancestral node. It is motivated by study of protein function at a particular divergence event. A polynomial-time algorithm for this problem is was given by Yang et al. [1995].

We study the problem of Joint Reconstruction. This problem is motivated by the need to research the collection of all ancestral sequences to understand phenomena across the whole phylogenetic tree. Studies of positive Darwinian selection, and research of parallel evolution are examples for such whole-tree effects.

In Section 2.3 we provide an efficient dynamic programming algorithm for Joint Reconstruction of maximum-likelihood ancestral sequences. The memory requirement of our algorithm scales linearly with the number of sequences, while its running time is proportional to the sum of their lengths.

For certain purposes, the evolutionary models discussed so far are still not accurate enough. They assume that the rate of mutation is the same for all positions

along the sequence. This simplifying assumption is invalid. It is statistically superior to assume that the mutation rate varies among sites, that is, for each position along the sequence this rate is randomly chosen according to a prescribed prior distribution. The Γ-distribution is used as a standard for such prior [Yang, 1993a].

Intuitively, ancestral reconstruction seems harder in the Γ-rate model. In this case, the likelihood of a single sequence position can no longer be presented as product of independent terms, one per branch. The substitution rate is common to all branches at the same position, and thus their corresponding terms are not independent. This prevents decomposition of the reconstruction task to separate, local tasks that involve a distinct part of the tree. Since a simple decompositional approach seems impossible, we need to directly solve the complete problem, which makes the task much harder.

Fortunately, even in the Γ-rate model, the Marginal Reconstruction problem can be formulated as maximizing a sum of terms, each of which is a fixed-rate likelihood. Thus, a polynomial-time algorithm for this problem is available [Yang, 1999]. In contrast, the Joint Reconstruction problem which we study, has no polynomial-time exact solution. Exhaustive (exponential-time) solution is implemented in [Yang, 1994], for trees of up to 5 sequences. Pupko [2000] applied a hill-climbing heuristic to this problem, analyzing phylogenetic trees for more than 30 species. A polynomial-time heuristic for Joint Γ-rate Reconstruction is to use Marginal Reconstruction for each internal node. Yang [1999] argued that typically, the resulting solution does not differ by much from the true optimum.

In section 2.4 we provide an exact Branch-and-Bound algorithm for reconstructing ancestral sequences, assuming a Γ-distributed rate of evolution. The algorithm was empirically tested and shown to be capable of handling data-sets of more than 70 sequences in reasonable time. The obtained reconstruction is guaranteed to have maximum likelihood. We apply our algorithm to real biological data. This allows, for the first time, comparing the exact Joint Reconstruction to the heuristic of using the Marginal Reconstruction instead. Our analysis of the results proves that indeed, in these cases, the Marginal Reconstruction heuristic made only a few errors, although the exact solution is statistically superior.

The chapter is organized as follows. In Section 2.2 we review the homogeneous

rate and variable rate probabilistic models of sequence evolution, and introduce terminology. In Sections 2.3 and 2.4 we present our algorithms for reconstructing ancestral sequences, assuming homogeneous and variable rate, respectively. In Section 2.5 we apply our methods to simulated data and to real biological datasets. We conclude with discussion in Section 2.6.

2.2 Preliminaries

2.2.1 Notation

Let $T = (V, E)$ by a binary, unrooted phylogenetic tree over the set of nodes $V = 1, 2, \ldots, 2n-2$, with $1, \ldots, n$ being the leaves. The nodes $n+1, \ldots, 2n-2$ are called *ancestral*. T is assumed to be undirected. A *rooting* of T at an ancestral node r, denoted $T^r = (V, E^r)$, is a directed version of T, with all edges oriented away from the *root* r. We reserve the term *edges* for unoriented trees and *branches* for oriented trees. For a branch $vu \in E^r$, we say that v is the *parent* of u, and that u is the *child* of v. We denote $v = p(u)$. Note, that the rooted tree T^r is almost binary: except the root, that has three children, all other internal nodes have two children each. Let u, v be two nodes in T^r. We say that u is a *descendant* of v if u is reachable from v by a directed path in T^r. For a node $u \neq r$, we denote the subtree of T^r induced by u and its descendants by $T^{r,u}$.

In order to describe different lengths of evolutionary time associated with distinct edges, we assign to each edge e a known duration τ_e, and denote $\tau = \{\tau_e\}_{e \in E}$. We also associate a possibly different rate of evolution, ρ_i, with every sequence position $1 \leq i \leq l$.

We associate l-long sequences $S^1, \ldots, S^{2n-2}$ over an alphabet Σ with the tree nodes $1, \ldots, 2n-2$. For the rest of this chapter, we refer to the characters in Σ as amino-acids, unless explicitly mentioned otherwise, but all results apply to a general alphabet. We denote the characters in sequence S^k by $s_1^k, \ldots, s_l^k$. We denote by $\mathfrak{S} = \{S^1, \ldots, S^n\}$ the set of sequences associated with leaves, while $\hat{\mathfrak{S}} = \{S^{n+1}, \ldots, S^{2n-2}\}$ denotes the set of sequences associated with ancestral nodes.

We denote by π the $|\Sigma|$-long vector such that $\pi[x]$ is the probability of a-priori observing an amino acid x. We denote the probability of an amino acid x mutating into an amino acid y during one time unit, with a unit rate of evolution, by $m[x, y]$, and arrange these values in a matrix M of order $|\Sigma| \times |\Sigma|$, setting $M_{xy} = m[x, y]$.

We denote by $a_{e,\rho_i}[x, y]$ the probability of an amino acid x mutating into an amino acid y along the edge e with evolutionary rate ρ_i. We arrange the values $a_{e,\rho_i}[x, y]$ for all x, y in a matrix A_{e,ρ_i} of order $|\Sigma| \times |\Sigma|$.

We denote the Gamma-distribution density function with shape parameter α and scale parameter β by:

$$Pr(y < r) = \Gamma[\alpha, \beta](r) = \frac{\beta^\alpha}{\Gamma(\alpha)} e^{-\beta r} r^{\alpha-1} \tag{2.1}$$

where $\Gamma(\alpha)$ is the Gamma function:

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} dx \tag{2.2}$$

2.2.2 Probabilistic Model

We assume that the sequences in $\mathfrak{S} \cup \hat{\mathfrak{S}}$ evolved according a stochastic process [Felsenstein, 1981, Yang, 1993a], which we now detail.

- For each position $i = 1, \ldots, l$:
    1. **[For the variable rate model:]** Randomly choose ρ_i, according to the $\Gamma[\alpha, \beta]$ distribution.
    2. **[For the homogeneous rate model:]** Set $\rho_i = 1$.
    3. Pick an arbitrary ancestral node v of T and set it to be the root. Randomly assign a value to s_i^v: For each amino-acid x, the probability of s_i^v to be x is $\pi[x]$.
    4. Traverse T^v top-down. Upon visiting a node w, whose parent is u, randomly assign a value to s_i^w, depending on the value x already assigned to s_i^u: For each amino-acid y, the probability of s_i^w to be y is $A_{uw,\rho_i}[x, y]$.

This model is justified by the following assumptions regarding evolution:

1. The stochastic process is independent and identically distributed at different positions along the tree, and at different sites.

2. The process is time-reversible, i.e., when considering evolution of a single sequence position along a lineage between two nodes, the chances of observing two amino acids at these nodes is invariant to the direction of the lineage between them:

$$\pi[x]a_{e,\rho_i}[x,y] = \pi[y]a_{e,\rho_i}[y,x] \tag{2.3}$$

   This reversibility assumption implies that the tree can be rooted arbitrarily for our purposes [Felsenstein, 1981].

3. ρ_i represents rate of evolution and τ_e stands for duration of evolutionary time. Formally, this implies that the matrix A_{e,ρ_i} is M exponentiated to the power of $\rho_i \times \tau_e$.

4. The rate of evolution is constant in the homogeneous-rate model. This is the simplest possible assumption. However, some sites along a protein may be highly conserved, while others may be mutation hot-spots that are hypervariable. We do not know in advance which positions are which. This is reflected in the variable-rate model, which assumes that the rate for each position is randomly chosen. The rate distribution is assumed to be of the Gamma family, which was found to be conveniently flexible.

The parameters for this model are assumed to be known in advance:

- The phylogenetic tree $T = (V, E)$, along with the set τ of branch lengths. These may be known from previous studies of other proteins in the same group of species. Otherwise they can be computed from the given dataset of contemporary sequences $\mathfrak{S}$ using standard phylogenetic techniques, e.g., the neighbor-joining algorithm [Saitou and Nei, 1987].

- The matrix M governing the evolutionary process, along with the vector π of a-priori amino-acid frequencies, are usually assumed to be fixed for

essentially all proteins. They have been empirically computed by several authors [Dayhoff et al., 1978, Jones et al., 1992, Adachi and Hasegawa, 1996]. The vector π is sometimes computed from the given data instead.

- **[In the variable rate model:]** The Γ-distribution parameters, α and β. Observe, that the scale parameter β is just a multiplicative factor that scales the branch lengths. Without loss of generality, we set $\beta \equiv 1$. The α shape parameter is usually computed to best fit the given dataset.

2.2.3 Problem Statement

The stochastic process detailed in Section 2.2.2 defines a probability space over the collection of all possible outputs $\mathfrak{S} \cup \hat{\mathfrak{S}}$. Given $\mathfrak{S}$, this induces a likelihood function over all candidate sets of ancestral sequences. Our goal is to infer a set $\hat{\mathfrak{S}}$ maximizing this likelihood.

We can now state this likelihood function explicitly. For each position i, we say that the vector $(s_i^1, \ldots, s_i^n)$ of contemporary amino acids at this position is the i-th *contemporary vector* and denote it by S_i. Similarly, we say that the vector $\{s_i^{n+1}, \ldots, s_i^{2n-2}\}$ of ancestral amino acids at this position is the i-th *ancestral vector* and denote it by $\hat{S}_i$. Assuming a rate ρ_i, we consider the likelihood f of $\hat{S}_i$:

$$f(\hat{S}_i|M, \pi, \rho_i, S_i, T^v, \tau) = \pi[s_i^v] \prod_{uw \in E^v} A_{uw,\rho_i}[s_i^u, s_i^v] \tag{2.4}$$

where T^v is a rooting of T at an arbitrary ancestral node v. The time-reversibility assumption on M and π implies that this expression is indeed independent of the choice of v [Felsenstein, 1981, Yang et al., 1995]. The likelihood of $\hat{S}_i$ is therefore:

1. In the homogeneous rate case:

$$L(\hat{S}_i|M, \pi, S_i, T, \tau) = f(\hat{S}_i|M, \pi, 1, S_i, T^v, \tau) \tag{2.5}$$

2. In the variable rate case:

$$L(\hat{S}_i|M, \pi, S_i, \alpha, \beta, T, \tau) = \int_{\rho_i=0}^{\infty} f(\hat{S}_i|M, \pi, \rho_i, S_i, T^v, \tau)\Gamma[\alpha, \beta](\rho_i)d\rho_i \tag{2.6}$$

In practice, following Yang [1993a], we discretize the Γ distribution. We partition the range $(0, \infty)$ of possible rates into a constant number k of equiprobable intervals. Let ρ_j be the average rate along the j-th such interval. Instead of the accurate, continuous distribution $\Gamma[\alpha, \beta]$, we use an approximate, discrete distribution, where each rate ρ_j has probability $\frac{1}{k}$. The likelihood of $\hat{S}_i$ is rewritten as follows:

$$L(\hat{S}_i|M, \pi, S_i, \alpha, \beta, T, \tau) = \sum_{j=1}^{k} \frac{1}{k} f(\hat{S}_i|M, \pi, \rho_j, S_i, T^v, \tau) \qquad (2.7)$$

Finally, the likelihood of $\hat{\mathfrak{S}}$ is:

$$\mathcal{L}(\hat{\mathfrak{S}}|M, \pi, \mathfrak{S}, \alpha, \beta, T, \tau) = \prod_{i=1}^{l} L(\hat{S}_i|M, \pi, S_i, \alpha, \beta, T, \tau) \qquad (2.8)$$

In the sequel, we omit the parameters $M, \pi, S_i, \alpha, \beta, T, \tau, \rho_i, \mathfrak{S}, T^v$ from the functions f, L and $\mathcal{L}$ whenever they are clear from context.

We are now ready to formally define the problem of *ancestral reconstruction*:

Input:

- The set $\mathfrak{S}$ of sequences.
- Model parameters:
  - The phylogenetic tree $T = (V, E)$, with the set τ of branch lengths.
  - The matrix M governing the evolutionary process, with the vector π of amino-acid frequencies.
  - **[In the variable rate model:]** The Γ-distribution parameters, α and β.

Output: The set $\hat{\mathfrak{S}}$ of ancestral sequences maximizing $\mathcal{L}(\hat{\mathfrak{S}})$.

2.3 Joint Reconstruction with Homogeneous Rate

Our algorithm for the homogeneous model is based on decomposability of the likelihood score. In this model, given the reconstructed character at some internal node, one can consider the subtrees obtained by deletion of that node, and reconstruct each of them independently of the others. Observe, that if the reconstructed node is adjacent to one or more leaves, then some of these subtrees are trivially reconstructed. This calls for a dynamic programming approach, working our way from the leaves inward. We root the tree arbitrarily at the node r. At each node u traversed, we consider each of the characters $x \in \Sigma$, for the reconstruction of the node. For each such character, we compute and record the most likely reconstructions of each of the subtrees rooted at the sons of u. Once the root is reached, we trace our steps back towards the leaves to recover the entries of the most likely ancestral vector.

We now formally present this algorithm. We focus on a single position i as the reconstruction of each ancestral vector $(s_i^{n+1}, \ldots, s_i^{2n-2})$ is independent for each i. We omit the subscript i for convenience. Let $e = uv$ be a branch of T^r, and let x be a character. Let $V^{r,v}(u, x)$ be the most likely ancestral vector for the subtree $T^{r,v}$ given that the character at u is x. Note, that the entries of this vector only concern a subset of the ancestral nodes in T, those in $T^{r,v}$. Let $L_e(x)$ be the likelihood of this vector, and let $C_e(x)$ be the character it assigns to v. Furthermore, let $\Phi_v(x)$ be the likelihood of $V^{r,v}(v, x)$. Obviously, for each node v and character x, $\Phi_v(x)$ is a product of likelihoods associated with immediate subtrees of $T^{r,v}$: $\Phi_v(x) = \prod_{e=vw} L_e(x)$, The dynamic program is defined as follows:

1. **Initialize:** For each leaf v, and character x, set:

$$\Phi_v(x) = \begin{cases} 1 & \text{if } x \text{ is the contemporary character at } v \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

2. **Recurse:**

   - For each non-root node v, let $e = uv$ be the branch leading to v. Once $\Phi_v(y)$ values have been computed for each character y, we recursively

compute

$$\begin{aligned} L_e(x) &\leftarrow max_y A_{uv,1}[x,y] \times \Phi_v(y) \\ C_e(x) &\leftarrow argmax_y A_{uv,1}[x,y] \times \Phi_v(y) \end{aligned}$$

- For each node u, once $L_e(x)$ values have been computed for all branches $e = uw$, we recursively compute

$$\Phi_u(x) \leftarrow \prod_{e=uw} L_e(x)$$

3. **Conclude:** Once all $L_e(x)$ and $C_e(x)$ entries have been computed, the ancestral root character can be reconstructed according to the formula:

$$s^r \leftarrow argmax_y \pi[y] \times \Phi_r(y)$$

4. **Trace-back:** For each node v whose immediate ancestor u has been reconstructed, set $s^v \leftarrow C_e(s^u)$, where $e = uv$.

The algorithm takes time and space $O(n|\Sigma|)$. We note that in practice, instead of multiplying the probabilities themselves, it is more convenient to sum their logarithms.

2.4 Joint Reconstruction with Variable Rate

2.4.1 Ancestral Vectors with Wildcards

The first observation is that each ancestral vector for a particular position i can be reconstructed independently. We therefore consider only the optimization of $L(\hat{S}_i)$, hereafter. We omit the subscript i for convenience.

We augment the alphabet with a wildcard: $\underline{\Sigma} = \Sigma \cup \{*\}$. An ancestral vector $\hat{S} = (\hat{s^{n+1}}, \ldots, \hat{s^{2n-2}})$ over Σ is said to be an *instance* of a vector $\underline{\hat{S}} = (\underline{\hat{s}^{n+1}}, \ldots, \underline{\hat{s}^{2n-2}})$ over $\underline{\Sigma}$ if $\underline{\Sigma}$ and $\underline{\hat{S}}$ are identical to one another in all non-wildcard vector coordinates: $\forall u : \underline{\hat{s}^u} \in \{\hat{s^u}, *\}$. We also denote in that case

$\hat{S} \in \underline{\hat{S}}$. The relation $''\in''$ between vectors over $\underline{\Sigma}$ is defined in the same way. We identify $\underline{\hat{S}}$ with the set of all of its instances.

Let $\underline{\hat{S}} = (\underline{\hat{s}^{n+1}}, \ldots, \underline{\hat{s}^{2n-2}})$ be a vector over $\underline{\Sigma}$, with $\underline{\hat{s}^u} = *$, and let $x \in \Sigma$ be a character. We define $\underline{\hat{S}}[u \leftarrow x]$ as the vector obtained from $\underline{\hat{S}}$ by setting the u-th entry to x.

2.4.2 Outline of the Branch-and-Bound Algorithm

For each ancestral vector $\underline{\hat{S}}$ over $\underline{\Sigma}$, we define:

$$MAXL(\underline{\hat{S}}) = \max_{\hat{S} \text{ over } \Sigma, \hat{S} \in \underline{\hat{S}}} \{L(\hat{S})\} \tag{2.10}$$

The output sought is therefore $MAXL(\underline{\hat{S}}^{init})$, where $\underline{\hat{S}}^{init}$ is an all-* vector. One can compute this value of $MAXL$ recursively, by the following recursion rule:

$$MAXL(\underline{\hat{S}}) = \begin{cases} L(\underline{\hat{S}}) & \text{if } \underline{\hat{S}} \text{ does not contain any *.} \\ \max_{x \in \Sigma} MAXL\left(\underline{\hat{s}^u}[u \leftarrow x]\right) & \text{if } \underline{\hat{s}^u} = * \text{ for some } u \end{cases} \tag{2.11}$$

Naively, this recursion leads to an exhaustive search, in $O(|\Sigma|^n)$ time. However, we can use the Branch-and-Bound paradigm to greatly reduce the search time in practice, as we now explain. In exhaustive search, each call to the $MAXL$ evaluation procedure recursively branches to $|\Sigma|$ nested calls. Instead, we can eliminate an unfruitful recursive call using an upper bound on the value of possible solutions explored during this call: If we already have a different candidate solution, whose likelihood exceeds this upper bound, there is no need to explore this recursive branching of the search tree. To avoid confusion, we distinguish between a *branch* of the phylogenetic tree, and *recursive branching* of the tree of recursive calls. More formally, Figure 2.4.2 contains the pseudo-code for our algorithm.

2.4.3 Bounding the Likelihood

To complete the description of the algorithm outlined in Section 2.4.2, it remains to present an upper bound on $MAXL(\underline{\hat{S}})$, which can be computed rapidly. We now describe two bounds. The minimum of them will be our upper bound.

```
Vector ComputeMAXL(  Vector current_wildcard_vec,
                     float best_likelihood_so_far )
{
    if ( UpperBound(current_wildcard_vec)> best_likelihood_so_far) then {
        if (no * characters in current_wildcard_vec) then {
            return current_wildcard_vec
        }
        else {
            u ← some position along current_wildcard_vec with *
            most_likely_vec_so_far ← UNDEFINED_VECTOR
            best_likelihood_so_far ← 0
            foreach x ∈ Σ do {
                sub_vec ← current_wildcard_vec[u ← x]
                best_vec_in_branch ← ComputeMAXL(sub_vec, best_likelihood_so_far)
                if (L(best_vec_in_branch) > best_likelihood_so_far) then {
                    most_likely_vec_so_far ← best_vec_in_branch
                    best_likelihood_so_far ← L(best_vec_in_branch)
                }
            }
            return most_likely_vec_so_far;
        }
    }
    else { /* UpperBound(current_wildcard_vec)≤ best_likelihood_so_far */
        return UNDEFINED_VECTOR
    }
}
```

Figure 2.1: Pseudo-code for our Branch-and-Bound algorithm for ancestral sequence reconstruction. This `ComputeMAXL` function is initially called with a $current\_wildcard\_vec$ parameter which is all-wildcard, and a $best\_likelihood\_so\_far$ parameter which is zero. It returns the most likely ancestral vector.

Bound Max-Likelihood by Sum-of-Likelihoods

The first bound is based on the following observation:

$$SUML(\underline{\hat{S}}) \equiv \sum_{\hat{S} \in \underline{\hat{S}}} L(\hat{S}) \geq MAXL(\underline{\hat{S}}) \tag{2.12}$$

$SUML$ can be computed efficiently. In fact, for a vector $\underline{\hat{S}}$ with exactly one non-* character entry $\underline{\hat{s}^u} = x$, the quantity $SUML(\underline{\hat{S}})$ is exactly the *marginal likelihood* of that character: the probability of the observed sequences assuming the ancestral character at node u is x. The marginal likelihood can be computed using the algorithm of Koshi and Goldstein [1996]. For an arbitrary vector $\underline{\hat{S}} = (\underline{\hat{s}^{n+1}}, \ldots, \underline{\hat{s}^{2n-2}})$, we compute $SUML$ using dynamic programming, as we now describe.

For each tree node $u \in \{1, \ldots, 2n-2\}$, for each character $x \in \Sigma$, and for each rate $j = 1, \ldots, k$, we define a dynamic programming cell, $D[j, u, x]$. This cell will record the probability of observing the leaves of $T^{r,u}$ assuming the rate of evolution is ρ_j, and the ancestral character at node u is x. The order of computing entries in D is as follows: We root the tree arbitrarily at some ancestral node r. For each rate ρ_j, we traverse the tree from the leaves to the root. Upon visiting a node u, we compute, for all characters x, the values of $D[j, u, x]$.

For each leaf u, with the input contemporary vector having value of y at u, we set $D[j, u, x] \leftarrow 1$ if $x = y$, and $D[j, u, x] \leftarrow 0$ otherwise. For an ancestral node u, if $\underline{\hat{s}^u} \notin \{x, *\}$, we set $D[j, u, x] \leftarrow 0$. Otherwise, we assume $D[j, w, y]$ entries have already been computed for each character y and child w of u, and set:

$$D[j, u, x] \leftarrow \prod_{uw \in E} \left(\sum_{y \in \Sigma} A_{uw, \rho_j}[x, y] D[j, w, y] \right) \tag{2.13}$$

Finally, once entries of the D-table are known, we return:

$$SUML(\underline{\hat{S}}) = \sum_{j=1}^{k} \frac{1}{k} \sum_{x \in \Sigma} \pi[x] D[j, x, r] \tag{2.14}$$

The algorithm requires $O(nlk|\Sigma|^2) = O(nl)$ time. To analyze space complexity, observe that only the D entries at r are needed for computing the final result. Also, the computation of entries for each specific ρ_j does not depend on those of ρ_i, $i \neq j$. Hence, one can separately compute all D entries for a specific ρ_j. This implies a space complexity of $O(n|\Sigma|) = O(n)$.

Bound Maximum-of-Sums by Sum-of-Maxima

Recall that

$$MAXL(\underline{\hat{S}}) = \max_{\hat{S}} \left\{ \sum_{j=1}^{k} \frac{1}{k} f(\underline{\hat{S}}|\rho_j) \right\} \tag{2.15}$$

Therefore:

$$MAXL(\underline{\hat{S}}) \leq \sum_{j=1}^{k} \frac{1}{k} \max_{\hat{S}} \left\{ f(\underline{\hat{S}}|\rho_j) \right\} \tag{2.16}$$

Observe, that f is the maximum likelihood of an ancestral reconstruction with a constant rate of evolution, ρ_j. As discussed in Section 2.3, f can be computed efficiently in $O(nl)$ time, and so is the bound on $MAXL$.

2.4.4 Enhancements

The Branch-and-Bound optimization paradigm is guaranteed to produce the optimal solution, but the search may be exhaustive in the worst case. To achieve manageable running times, one must enhance the basic method by heuristic means. One important enhancement concerns the order of enumeration. In Branch-and-Bound, it is desirable to obtain a good lower bound in an early stage of the search, so that as much of the recursive branchings as possible will be pruned from that point onward. To this end, one seeks a very good candidate solution in the first stages of the search. We would therefore like to search the most promising recursive branchings first.

In our problem, we use the upper bound to estimate how promising each recursive branching is. At each point during our search, we have $|\Sigma|$ options for recursively branching, one per character. We choose the option whose upper bound is highest.

In fact, we do more than that. Consider a recursive call to our `ComputeMAXL` function, and denote the $current\_wildcard\_vec$ parameter by $\underline{\hat{S}}$. At each such call the search can recursively branch to $\underline{\hat{S}}[u \leftarrow x]$. Therefore, not only do we have to rank the options for each character x, but we also have to decide what is the node u to be assigned a character. Again, we use the upper bound to choose the most promising option. We first recursively compute $MAXL(\underline{\hat{S}}[u \leftarrow x])$ for the vector $\underline{\hat{S}}[u \leftarrow x]$ whose upper bound is maximum (over all possible values for u and x. Only then do we compute $MAXL(\underline{\hat{S}}[u \leftarrow x'])$ for all other characters x'.

2.5 Results

We have implemented our algorithms in a C++ application called FastML. The software is publicly available as `http://evolu3.ism.ac.jp/~tal/fastml.htm` We report benchmarks on datasets of mammalian protein sequences.

2.5.1 Results with Homogeneous Rate Reconstruction

Aligned *cytochrome b* amino-acid sequences from a sample of mammals were taken from a data-set by Takezaki and Gojobori [1999]. A phylogenetic tree with 21 amino acid sequences of length 382 (see Figure 2.2) was prepared by the neighbor-joining algorithm [Saitou and Nei, 1987]. The matrix of Adachi [1995], suitable for mitocondrial proteins was used to calculate replacement probabilities. We applied our algorithm to this data set, and reconstructed the joint ancestral sequences. Reconstruction was completed within less than a minute on a Pentium 450MHz machine. Note, that existing algorithms for Joint Reconstruction Zhang and Nei [1997], Yang [1999] are of exponential complexity, and cannot be applied to a data-set of similar size.

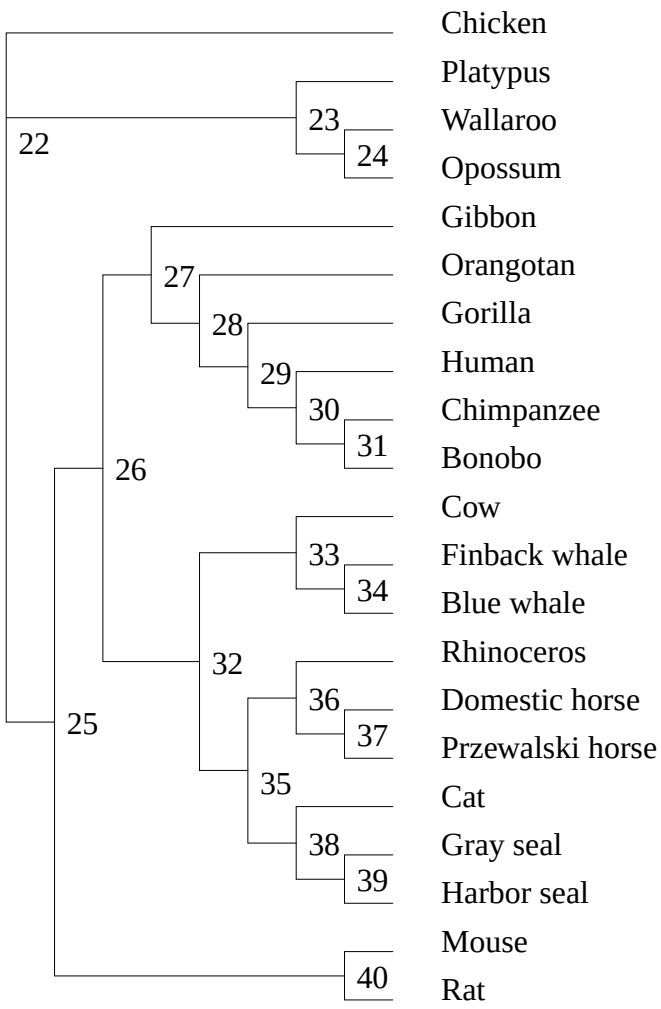


Figure 2.2: Tree for 21 *cytbtree b* sequences used for reconstruction using the homogeneous rate model. Sequences were taken from Takezaki and Gojobori [1999]. Branch lengths are not according to scale.

node	position	reconstruction	
		Joint	Marginal
23	209	T	S
24	209	T	S
25	263	S	N
26	263	S	N
27	263	S	N
28	263	S	N
29	263	S	N
32	238	I	V
33	241	M	L
35	23	A	T
36	23	A	T
37	23	A	T
38	23	A	T

Table 2.1: Difference between Joint ML Reconstruction and Marginal ML Reconstruction. The numbers in the first column refer to the internal-node labels in Figure 2.2. Amino acids are shown as one-letter abbreviations. In all other nodes and positions, the two methods yield the same ancestral amino-acid reconstruction.

We compared our reconstruction versus the marginal reconstruction on this data-set. Differences are summarized in Table 2.1.

2.5.2 Results with Variable Rate Reconstruction

We analyzed the ancestral sequence reconstruction of the *lysozyme c* gene family. This family is of particular interest to variable-rate analysis, because of complex evolutionary processes operating on different sequence loci [Wen and Irwin, 1999]. 71 vertebrate representative 148-long sequences were chosen. Ancestral sequences were constructed based on the neighbor-joining tree [Saitou and Nei, 1987], presented in Figure 2.3. We used the matrix of amino-acid substitution probabilities by [Jones et al., 1992].

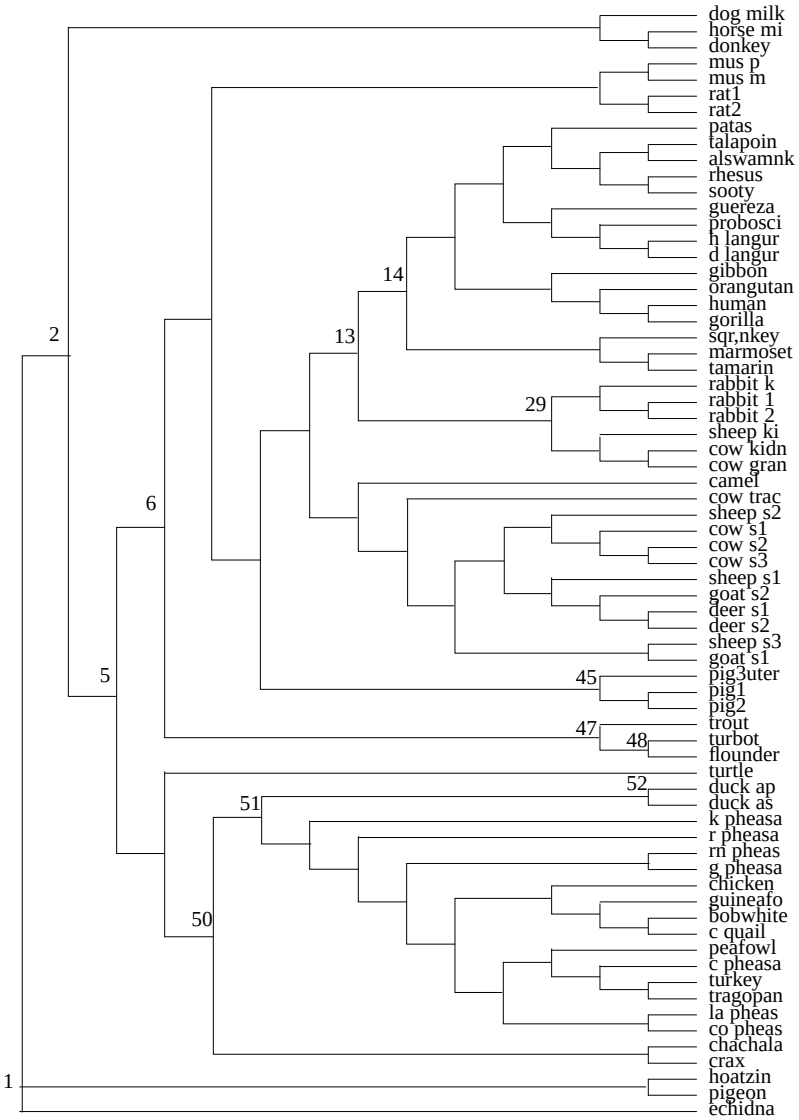


Figure 2.3: Tree for 71 lysozyme sequences used for reconstruction using the variable rate model. Sequences were taken from Wen and Irwin [1999]. Nodes listed in Table 2.2 are identified in the sketch. Branch lengths are not according to scale.

The likelihood of the tree is significantly improved by the assumption of rate variation among sites (log-likelihood of -3668.58 versus -3808.72), further motivating our analysis. The alpha parameter of the gamma distribution was computed to best fit the data, i.e., to maximize the likelihood of the observed sequences. The maximum-likelihood alpha was found to be 0.9335. To infer the ancestral sequences the discrete gamma distribution with $k = 4$ categories was used [Yang, 1994]. The log-likelihood of the reconstruction (rather than the tree itself), which is the function optimized by the ancestral reconstruction, is -3760.64. This is compared to the maximum likelihood of -3880.87 for reconstruction assuming homogeneous rate. The differences between these two reconstructions are summarized in Table 2.2.

2.6 Discussion

2.6.1 Biological Analysis of Results

We have chosen to study the lysozyme sequences because of an intriguing evolutionary phenomenon they display, as we now discuss. *Parallel evolution* is the process of a character arising along two or more independent evolutionary lineages. Foregut fermentation is an example for such parallel evolution. It has arisen at least twice in the evolution of placental mammals (in ruminants and colobine monkeys) and at least once in birds (in the hoatzin). Lysozyme is known to be recruited by the digestive system in foregut fermenters. This protein is therefore a natural target for searching molecular evidence related to the parallel evolution of foregut fermentation.

At the sequence level, parallel mutations that change the amino-acids at the same site to the same amino-acid are called *homoplasious*. In any molecular phylogenetic tree, a certain number of homoplasious changes is expected to be observed purely by chance. However, if the number of homoplasious events exceeds its chance expectation, then it is unlikely that these events have occurred by random genetic drift. The excess in homoplacy is then attributed to positive Darwinian selection.

position	node	rate model	
		fixed	Gamma
4	1	S	E
4	2	S	E
11	1	N	T
11	2	N	T
11	5	A	T
11	6	A	T
11	47	A	T
14	29	K	R
33	48	Q	K
37	13	N	D
37	14	N	D
37	29	N	D
37	47	S	N
37	48	S	N
43	1	F	T
43	2	F	T
94	51	R	K
94	52	R	K
113	13	R	K
113	14	R	K
113	45	R	K
123	50	R	K

Table 2.2: Differences between ancestral amino-acid reconstructions inferred with and without the assumption of rate variation among sites. The numbers in the second column refer to the internal-node labels in Figure 2.3.

Based on the above rationale, Zhang and Kumar [1997] suggested eight amino acid sites in lysozyme that have evolved by positive selection in foregut fermenters. Unfortunately, previous such studies of homoplasious replacements were based on a small number of sequences. Our methods are the first to allow analysis of large data-sets. Such analysis indeed requires Joint Reconstruction of the ancestral sequences. Furthermore, the lysozyme phylogenetic tree is significantly more likely under the assumption of variable rate of evolution. This motivates analysis of ancestral sequences in the same model. We thus revisit the proposition of positive selection in the lysozyme, using a large data-set.

The ability to reconstruct ancestral sequences for a large number of extant sequences has led to better mapping of the homoplasious replacements than in previous studies. Our analysis suggests that some of the homoplasious replacements that were previously mapped strictly to the lineages leading to foregut fermentation taxa are now mapped to lineages either inside the foregut fermenting clades, or to lineages preceding them. For example, the R to K homoplacy in position 14 was previously mapped to the lineage leading to cow, and with the addition of the camel sequence, this homoplacy is now mapped to the lineage before the divergence of the camel.

2.6.2 Computational Directions for Subsequent Research

We presented algorithms for ancestral reconstruction, both with and without the simplifying assumption that the rate of sequence evolution is homogeneous among sites. This work has consequences beyond the task of ancestral reconstruction. The framework we present can help infer not only the ancestral vector, but also probabilities of observing a particular pair of characters at any pair of nodes. This can be applied, for example, to carefully infer positive selection in the tree. Moreover, our framework can greatly aid in inferring the phylogenetic tree itself, in cases it is not known in advance. In a subsequent work, we used this probabilistic reconstruction of ancestral sequences as a building block in a novel algorithm for learning phylogenetic trees [Friedman et al., 2001]. This Structural-EM algorithm is both faster and more accurate than previous Maximum-Likelihood methods for phylogenetic

inference. Incorporating variable rate into this new algorithm is still to be done.

Another important generalization concerns sequence gaps. In molecular phylogenetics, usually the available sequences are aligned with some gaps in certain positions for some of the sequences. Alignment columns containing gap characters are discarded in current studies (as was done also in the results reported here), as they behave differently from the model. Efficient reconstruction of the ancestral sequences in gapped positions is an open problem.

Chapter 3

Reconstructing Nucleotide Sequences from Hybridization Experiments

In this chapter we study algorithmic aspects of sequencing. We improve an existing method, called Sequencing By Hybridization (SBH), which we now outline. SBH is based on interrogating a target DNA molecule for all of its k-long sub-sequences, called *k-mers*. This is made possible by recent technologies, that immobilize many thousands of short single-stranded molecules of DNA, or *oligonucleotides*, on a small surface, called a *microarray*. Each oligonucleotide probes the target sequence for presence of its complementary counterpart. SBH calls for employing all possible k-long oligonucleotides, thereby obtaining the k-mer contents of the sequence. Subsequently, the target sequence is computationally reconstructed. Section 3.1 we introduce SBH.

Unfortunately, SBH has very limited practicality. In this chapter we introduce ways to enhance SBH by computationally using additional information.

One kind of information we use is additional data regarding the position of k-mers along the target molecule. These data can be obtained by novel microarray reactions. We study the arising computational problem in Section 3.2. The input to

this problem is a set of possible occurrence positions for each k-mer. The output is a target sequence all of whose k-mers satisfy the positional constraints. We provide a polynomial algorithm for this problem when no k-mer has more than two possible positions. We prove NP-hardness of the problem, even when no k-mers has more than three possible positions. This work has been published in [Ben-Dor et al., 1999]. We further study a biologically motivated restriction of this problem, to all sets of possible positions being intervals. We provide a parameterized algorithm to solve this restricted problem. This part has been published in [Ben-Dor et al., 2001a], with a complete version in [Ben-Dor et al., 2001b].

In Section 3.3 we study another method to enhance SBH. This method uses additional data which is usually available, and it can therefore be widely applied. Most contemporary tasks of sequence determination involve resequencing rather than sequencing, as one has prior information regarding the target sequence. One knows in advance, that the sequencing target highly resembles an already sequenced molecule. In fact, there are only a few small differences between these two sequences. It is these small differences that are of interest. We formalize this challenge as a computational problem, and provide a polynomial algorithm for it.

Our framework is probabilistic. We describe the resemblance to a known sequence as a profile HMM. We further describe results of the SBH reaction in probabilistic terms, and incorporate them into a graph whose edges correspond to probes and vertices to probe prefixes/suffixes of length $k-1$. The probabilistic formulation allows maximum likelihood analysis. It gives rise to an interesting optimization problem: finding the most likely sequence, which corresponds to a pair of paths: one path in the HMM transition graph, and another in the SBH graph. We can therefore solve an analogous optimization problem, finding a pair of paths with maximum weight. We provide several dynamic programming algorithms to solve this problem and some of its interesting restrictions. We provide a practical solution, although it is only guaranteed to be an approximate maximum-likelihood sequence. Simulations results are also provided. This work has been published in [Pe'er and Shamir, 2000b].

3.1 Introduction to Sequencing by Hybridization

In sequencing by hybridization (SBH), one has to reconstruct a sequence from its l-long substrings. SBH was proposed as an alternative to gel-based DNA sequencing approaches, but in its original form the method is not competitive. Positional SBH (PSBH) is a recently proposed enhancement of SBH in which one has additional information about the possible positions of each substring along the target sequence. We give a linear time algorithm for solving PSBH when each substring has at most two possible positions. On the other hand, we prove that the problem is NP-complete if each substring has at most three possible positions. We also show that PSBH is NP-complete if the set of allowed positions for each substring is an interval of length k, and provide a fast algorithm for the latter problem when k is bounded.

Sequencing by hybridization (SBH) was proposed in the late eighties as an alternative approach to gel-based DNA sequencing [Bains and Smith, 1988, Lysov et al., 1988, Southern, 1988, Drmanac and Crkvenjakov, 1987, Macevics, 1989]. Using DNA chips, cf. [Southern, 1996], one can in principle determine exactly which l-mers (l-tuples) appear as substrings in a target that needs sequencing, and try to infer its sequence. Practical values of l are 8 to 10.

The fundamental computational problem in SBH is the reconstruction of a sequence from its *spectrum* - the list of all l-mers that are included in the sequence along with their multiplicities. In Pevzner [1989] it was shown that the reconstruction problem can be solved efficiently by a reduction to finding an Eulerian path in the following graph: Vertices correspond to $(l-1)$-tuples, and for each l-tuple in the spectrum, an edge connects the vertices corresponding to its $(l-1)$-long prefix and suffix.

The main handicap of SBH is ambiguity of the solution. Alternative solutions are manifested as *branches* in the graph (i.e., two or more edges leaving the same vertex), and unless the number of branches is very small, there is no good way to determine the correct sequence. Theoretical analysis and simulations [Southern et al., 1992, Pevzner and Lipshutz, 1994, Arratia et al., 1997, Dyer et al., 1994] have shown that the average length of a uniquely reconstructible sequence using

an 8-mer chip is only about two hundred, way below a single read length on a commercial gel-lane machine.

Due to the centrality of the sequencing problem in biotechnology and in the Human Genome Project, and due to its mathematical elegance, SBH continues to draw a lot of attention. Many authors have suggested ways to improve the basic method. Alternative chip designs [Bains and Smith, 1988, Khrapko et al., 1989, Pevzner et al., 1991, Preparata et al., 1999] as well as interactive protocols [Skiena and Sundaram, 1995] were suggested. An effective and competitive sequencing solution using SBH has yet to be demonstrated.

3.2 On the Complexity of Positional Sequencing by Hybridization

Recently, several authors have suggested enhancements of SBH based on adding location information to the spectrum [Adleman, 1998, Broude et al., 1994, Hannenhalli et al., 1996, Gusfield et al., 1998]. In *positional sequencing by hybridization* (PSBH), additional information is gathered concerning the position of the l-mers in the target sequence. More precisely, for each l-mer in the spectrum its allowed positions along the target are registered. The reduction to the Eulerian path problem still applies, but for each edge in Pevzner's graph we now have constraints restricting its position in the Eulerian path. Mathematically, this gives rise to the *positional Eulerian path* problem (PEP): Given a directed graph with a list of allowed positions on each edge, decide if there exists an Eulerian path in which each edge appears in one of its allowed positions. In Hannenhalli et al. [1996] it was shown that PEP is NP-complete, even if all the lists of allowed positions are intervals of equal length. Note that this leaves open the complexity of PSBH in this case. They also gave a polynomial algorithm for the problem when the length of the intervals is bounded.

In this chapter we address the positional sequencing by hybridization problem in the case that the number of allowed positions per l-mer is bounded, and the positions need not be consecutive. We give a linear time algorithm for solving the

positional Eulerian path problem, and hence, the PSBH problem, in the case that each edge is allowed at most two positions. On the negative side, we show that the PSBH problem is NP-complete, even if each l-mer has at most three allowed positions and multiplicity one. We use in our hardness proof a reduction from the PEP problem restricted to the case where each edge is allowed at most three positions. The latter problem is shown to be NP-complete as well. We also study the complexity of PSBH in the case that the set of allowed positions for each substring is an interval of bounded length. We strengthen the results of Hannenhalli et al. [1996] with respect to this problem in two ways. First, we show that PSBH is NP-complete, even if all sets of allowed positions are k-long intervals. Second, we give a faster parameterized algorithm for the problem, where the parameter k is an upper bound on the size of the intervals. Our algorithm requires $O(mk^{1.5}4^k)$ time, compared to the $O(mk^3 \log k 4^k)$ bound of Hannenhalli et al. [1996].

The chapter is organized as follows: In Section 3.2.1 we define the PSBH and the PEP problems. In Section 3.2.2 we describe a linear time algorithm for the PEP problem when each edge has at most two allowed positions. In Section 3.2.3 we prove that the PEP problem is NP-complete if each edge has at most three allowed positions. In Section 3.2.4 we show that the PSBH problem is NP-complete when each l-mer is allowed at most three positions. Furthermore, in Section 3.2.5 we prove that the PSBH problem is NP-complete when all sets of allowed positions are intervals of equal length. Finally, in Section 3.2.6 we give a parameterized algorithm for the latter problem.

3.2.1 Preliminaries

All graphs in this chapter are finite and directed. Let $D = (V, E)$ be a graph. We denote $m = |E|$ throughout. For a vertex $v \in V$, we define its *in-neighbors* to be the set of all vertices from which there is an edge directed into v. We denote this set by $N_{in}(v) = \{u : (u, v) \in E\}$. We define the *in-degree* of v to be $|N_{in}(v)|$. The *out-neighbors* $N_{out}(v)$ and *out-degree* are similarly defined.

Let $E = \{e_1, \ldots, e_m\}$ and let P be a function mapping each edge of D to a non-empty set of integer labels from $\{1, \ldots, m\}$. The set $P(e)$ is called the set of

allowed positions of edge e. The pair (D, P) is called a *positional graph*. If for all e, $|P(e)| \leq k$, then (D, P) is called a *k-positional graph*. Let $\pi = \pi(1), \ldots, \pi(m)$ be a permutation of the edges in E. If π defines a (directed) path, i.e., for each $1 \leq i < m$, $\pi(i) = (u, v)$ and $\pi(i+1) = (v, w)$, for some $u, v, w \in V$, then we say that π is an *Eulerian path* in D.

An Eulerian path π in D is said to be *compliant* with the positional graph (D, P), if $\pi^{-1}(e) \in P(e)$ for every $e \in E$, that is, each edge in π occupies an allowed position. The k-positional Eulerian path problem is defined as follows:

Problem 1 (k-PEP)
Instance: *A k-positional graph (D, P).*
Question: *Is there an Eulerian path compliant with (D, P)?*

Let $\Sigma = \{A, C, G, T\}$. The *p-spectrum* of a string $X \in \Sigma^*$ is the multi-set of all p-long substrings of X. The problem of sequencing by hybridization is defined as follows:

Problem 2 (SBH)
Instance: *A multi-set S of p-long strings.*
Question: *Is S the p-spectrum of some string X?*

For simplicity, we shall call the input multi-set a spectrum, even if it does not correspond to a sequence. The SBH problem is solvable in polynomial time by a reduction to finding an Eulerian path in Pevzner's graph [Pevzner and Lipshutz, 1994]. More specifically, construct a graph D whose vertices correspond to $(p-1)$-long substrings of strings in S, and in which edges are directed from $\sigma_1 \cdots \sigma_{p-1}$ to $\sigma_2 \cdots \sigma_p$ for each $\sigma_1 \cdots \sigma_p \in S$. Then every solution $\sigma_1 \cdots \sigma_{m+p-1}$ to the SBH instance naturally corresponds to an Eulerian path $\sigma_1 \cdots \sigma_{p-1}, \ldots, \sigma_{m+1} \cdots \sigma_{m+p-1}$ in D.

The *positional SBH* problem is defined as follows:

Problem 3 (PSBH)
Instance: *A multi-set S of p-long strings. For each $s \in S$, a set $P(s) \subseteq \{0, \ldots, |S|-$*

$1\}$.
Question: *Is S the p-spectrum of some string X, such that for each $s \in S$ its position along X is in $P(s)$?*

If the set of allowed positions for each string is of size at most k, then the corresponding problem is called k-positional SBH, or k-PSBH. k-PSBH is linearly reducible to k-PEP in an obvious manner.

The Interval PEP and Interval PSBH problems are defined as follows:

Problem 4 (Interval PEP)
Instance: *A positional graph (D, P), such that for each edge e, the set $P(e)$ is a sub-interval of $[1, m]$ (i.e., an interval in the linear order $1, \ldots, m$).*
Question: *Is there an Eulerian path compliant with (D, P)?*

Problem 5 (Interval PSBH)
Instance: *A multi-set S of p-long strings. For each $s \in S$, a set $P(s)$ which is a sub-interval of $[0, |S| - 1]$.*
Question: *Is S the p-spectrum of some string X, such that for each $s \in S$ its position along X is in $P(s)$?*

3.2.2 A Linear Algorithm for 2-Positional Eulerian Path

In this section we provide a linear time algorithm for solving the 2-positional Eulerian path problem. A key element in our algorithm is a reduction to 2-SAT. Before the reduction can be applied, the input must be preprocessed, discarding unrealizable edge labels (positions).

Let $(D = (V, E), P)$ be the input 2-positional graph. For every $1 \leq t \leq m$ define $\Delta(t)$ to be the set of edges allowed at position t, i.e., $\Delta(t) \equiv \{e \in E : t \in P(e)\}$. Let us call a position t for which $\Delta(t) = \{e\}$ and $|P(e)| > 1$, a *resolvable* position.

The first phase of the algorithm applies the following preprocessing step:

While there exists a resolvable position t, **do**:
Suppose $\Delta(t) = \{e\}$ and $P(e) = \{t, t'\}$.
$\Delta(t') \leftarrow \Delta(t') \setminus \{e\}$.
$P(e) \leftarrow \{t\}$.

If at any stage we discover that some set $\Delta(t)$ is empty, then we output *False* and halt, since no edge can be labeled t.

Lemma 3.2.1 *The preprocessing step does not change the set of Eulerian paths compliant with* (D, P).

Proof: Let S be the set of Eulerian paths compliant with (D, P) before the preprocessing step. It is obvious that the preprocessing step can only reduce S. It thus suffices to prove that if π is compliant with (D, P) then π remains compliant with the new (D, P) obtained by a single iteration of the while loop.

Suppose there exists a resolvable position $1 \leq t \leq m$ such that $\Delta(t) = \{e\}$ and $P(e) = \{t, t'\}$. Then by definition of $\Delta(t)$, any Eulerian path $\pi \in S$ satisfies $\pi(t) = e$. Hence, $\pi(t') \neq e$ and upon updating $P(e) \leftarrow \{t\}$, π remains compliant with the new (D, P). ■

Lemma 3.2.2 *The preprocessing step can be implemented in linear time.*

Proof: We maintain current $P(e)$ for each e, and $\Delta(t)$ for each t. Initialization of $\Delta(t)$ for all t can be done in linear time. We further maintain a set Z of resolvable positions, which can be initialized in linear time. The while loop can be implemented by repeatedly handling the positions in Z. Let t be an arbitrary position in Z, where $\Delta(t) = \{e\}$ and $P(e) = \{t, t'\}$. Upon handling t, we delete it from Z, updating $P(e)$ and $\Delta(t')$ in constant time. If t' becomes resolvable, we add it to Z. ■

In the following, (D, P) and Δ refer to the positional graph obtained after the preprocessing phase.

Lemma 3.2.3 *In* (D, P) *each position is allowed for at most two edges.*

Proof: The preprocessing ensures that if for some position t, $|\Delta(t)| = 1$, then $e \in \Delta(t)$ satisfies $|P(e)| = 1$. Let R be the set of positions t with $|\Delta(t)| = 1$, and let $r = |R|$. Then there are $m - r$ positions t for which $|\Delta(t)| \geq 2$, and $r' \geq r$ edges e with $|P(e)| = 1$. Thus,

$$2(m-r) \leq \sum_{t \notin R} |\Delta(t)| = \sum_{t} |\Delta(t)| - r = \sum_{e} |P(e)| - r = 2m - r' - r \leq 2(m-r) \; .$$

Hence, $r = r'$ and each label $t \notin R$ occurs exactly twice, implying that $|\Delta(t)| \in \{1, 2\}$ for all t. ■

For every vertex $v \in V$ define $In(v,t)$ as the set of t-labeled edges entering v, i.e.,

$$In(v,t) \equiv \{(u,v) : (u,v) \in \Delta(t)\} \; .$$

Similarly define

$$Out(v,t) \equiv \{(v,u) : (v,u) \in \Delta(t)\} \; .$$

We say that a vertex v is *fixed to position* t in (D,P) if $In(v,t) = \Delta(t)$ or $Out(v,t+1) = \Delta(t+1)$. In other words, any Eulerian path compliant with (D,P) must have v as the $(t+1)$-st vertex in the path. Define Boolean variables X_e^t for every e, t such that $t \in P(e)$ ($2m - r$ variables in total). Examine the following sets of Boolean clauses:

$$X_e^t \qquad \text{for every } e,t \text{ such that } P(e) = \{t\} \; . \tag{3.1}$$

$$X_{e_1}^t \oplus X_{e_2}^t \qquad \text{for every } e_1, e_2, t \text{ such that } \Delta(t) = \{e_1, e_2\} \; . \tag{3.2}$$

$$X_e^{t_1} \oplus X_e^{t_2} \qquad \text{for every } e, t_1, t_2 \text{ such that } P(e) = \{t_1, t_2\} \; . \tag{3.3}$$

$$X_{(a,b)}^t \Leftrightarrow X_{(b,c)}^{t+1} \qquad \text{for every } t \in P((a,b)), t+1 \in P((b,c)) \text{ such that } b \text{ is not fixed to position } t \; . \tag{3.4}$$

$$\overline{X}_{(u,v)}^t \qquad \text{for every } t \in P((u,v)), t < m \text{ such that } Out(v,t+1) = \emptyset \; . \tag{3.5}$$

$$\overline{X}_{(u,v)}^t \qquad \text{for every } t \in P((u,v)), t > 1 \text{ such that } In(u,t-1) = \emptyset \; . \tag{3.6}$$

Lemma 3.2.4 *There is a positional Eulerian path compliant with (D, P) if and only if the set of clauses (3.1)-(3.6) is satisfiable.*

Proof:

$\Leftarrow$ Suppose that a satisfying truth assignment Φ exists. Let us assign an edge e to position t if and only if $\Phi(X_e^t) =$*True*. Clauses (3.1) and (3.2) guarantee that exactly one edge is assigned to each position. Clauses (3.1) and (3.3) guarantee that each edge is assigned to exactly one position, and that this position is allowed for it.

It remains to show that the above assignment of edges to positions yields a path in D. Suppose to the contrary that both $X_{(a,b)}^t$ and $X_{(b',c')}^{t+1}$ are assigned *True*, with $b \neq b'$. Then clauses (3.5) guarantee the existence of an edge $(b, c) \in \Delta(t + 1)$, while clauses (3.6) guarantee the existence of an edge $(a', b') \in \Delta(t)$. Therefore, b is not fixed to t, and a contradiction follows from clauses (3.4). Hence, Φ defines an Eulerian path compliant with (D, P).

$\Rightarrow$ Suppose that π is an Eulerian path compliant with (D, P). A truth assignment Φ satisfying clauses (3.1)-(3.6) can be defined as follows: Assign $\Phi(X_e^t) =$*True* if and only if $\pi(t) = e$. As π is a permutation, Φ satisfies clauses (3.1)-(3.3). Since π is a path, Φ satisfies clauses (3.5) and (3.6). It remains to prove that Φ satisfies clauses (3.4). Consider the clause $X_{(a,b)}^t \Leftrightarrow X_{(b,c)}^{t+1}$ in (3.4). Since b is not fixed to t, $\Delta(t) = \{(a, b), (a', b')\}$ and $\Delta(t + 1) = \{(b, c), (b'', c'')\}$ where $b', b'' \neq b$. There are two possible cases. Either $\pi(t) = (a, b)$ and then $\pi(t + 1) = (b, c)$, or $\pi(t) \neq (a, b)$ and then $\pi(t + 1) \neq (b, c)$. In both cases the clause is satisfied.

■

Lemma 3.2.5 *Clauses (3.1)-(3.6) can be generated in linear time.*

Proof: By definition and Lemma 3.2.3, each of the sets $P(e)$ and $\Delta(t)$ is of size at most 2. Clauses (3.1)-(3.3) can be trivially generated by scanning the sets

$P(e)$ for all e and $\Delta(t)$ for all t. This computation takes $O(m)$ time. To construct clauses (3.5) we examine each edge $e = (u, v)$ and each $t \in P(e)$ (there are at most $2m$ such pairs (e, t)). If $\Delta(t+1)$ contains no edge of the form (v, w), we include the clause $\bar{X}^t_{(u,v)}$. Construction of these clauses takes $O(m)$ time. Clauses (3.6) are constructed in a similar manner. Finally, clauses (3.4) can be constructed in linear time by examining $\Delta(t)$ and $\Delta(t+1)$ for all $t < m$. If we find a pair of edges $(a, b) \in \Delta(t)$ and $(b, c) \in \Delta(t+1)$, such that there exist edges $(u, v) \in \Delta(t)$, $v \neq b$ and $(w, z) \in \Delta(t+1)$, $w \neq b$, we include the appropriate clause. ■

Theorem 3.2.6 *2-PEP is solvable in linear time.*

Proof: The preprocessing phase is linear by Lemma 3.2.2. By Lemma 3.2.5, the number of clauses (3.1)-(3.6) is $O(m)$. Each exclusive OR clause in (3.2)-(3.3) and each equivalence clause in (3.4) can be written as two OR clauses, and therefore, by Lemma 3.2.5 one can generate all clauses in linear time. By Lemma 3.2.4, the problem is reduced to an instance of 2-SAT which is solvable in linear time [Apsvall et al., 1979]. ■

Corollary 3.2.7 *2-PSBH is solvable in linear time.*

All clauses (3.1)-(3.6) can take the form of $A \Leftrightarrow B$, with A, B being constants or literals. Hence, the 2-SAT instance can in fact be solved by a linear-time algorithm simpler than Apsvall et al. [1979] as follows: Let S be the set of all clauses, written as equivalence clauses. Construct an undirected graph $G = (U, F)$ such that $U = \{X^t_e, \bar{X}^t_e : e \in E, t \in P(e)\} \cup \{0, 1\}$. Include (A, B) and $(\bar{A}, \bar{B})$ in F whenever $A \Leftrightarrow B$ is a clause in S. It is easily seen that the 2-SAT instance is satisfiable if and only if no connected component of G contains two vertices corresponding to a variable (or a constant) and its negation.

3.2.3 3-Positional Eulerian Path is NP-Complete

In this section we prove that the 3-PEP problem is NP-complete by reduction from 3-SAT.

Theorem 3.2.8 *The 3-PEP problem is NP-complete.*

Proof: Membership in NP is trivial. We prove NP-hardness by reduction from *3-SAT*. We first provide a sketch of the construction. For each occurrence of a literal in the formula, a *special vertex* is introduced. Special vertices corresponding to the same literal are connected serially to form a *literal path*. Two literal paths of a variable and its negation are connected in parallel to form a *variable subgraph*. For each clause in the formula, the corresponding special vertices are connected by three edges to form a *clause triangle*. Finally, for each special vertex we introduce a triangle incident on it, called its *bypass triangle* (see Figure 3.1).

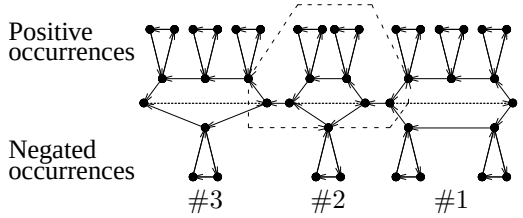


Figure 3.1: A schematic sketch of the main elements in our construction. The figure includes three variable subgraphs. The first variable, whose subgraph is the rightmost (#1), has three positive occurrences (top) and two negated occurrences (bottom), etc. One of the clause triangles is also drawn, using dashed line.

The sets of allowed positions are chosen so that they force every compliant Eulerian path to visit the literal paths one by one. A compliant Eulerian path corresponds to a satisfying truth assignment. When a special vertex is visited, either its clause triangle, or its bypass triangle are traversed. Traversing the clause triangle while passing through a certain literal's path, corresponds to this literal satisfying the clause. Eventually, we enable visiting all unvisited bypass triangles.

We now give the construction in detail. Let F be a 3-CNF formula with N variables $x_1, \ldots, x_N$, and M clauses $C_1, \ldots, C_M$. We assume, w.l.o.g., that each clause contains three distinct variables, and that all $2N$ literals occur in F. Denote $X_i = \{x_i\} \cup \{\overline{x}_i\}$. For a literal $L \in X_i$, let a_L denote the number of its occurrences in F. For $1 \leq j \leq a_L$ define $L(j) \equiv (L, j)$. Thus, $L(1), \ldots, L(a_L)$ is an enumeration of L's occurrences in F. For a clause $C = L \vee L' \vee L''$ introducing the j-th (j', j'') occurrence of L (L', L'', respectively), we write $C = L(j) \vee L'(j') \vee$

$L''(j'')$. We shall construct a directed graph $D = (V, E)$ and a map P from E to integer sets of size at most 3, such that F is satisfiable if and only if (D, P) has a compliant Eulerian path.

We introduce the following vertices:

- $u_i, \hat{u}_i$ for each variable x_i, $1 \leq i \leq N$.
- $v_{L(j)}, \hat{v}_{L(j)}$ for each occurrence $L(j)$ of the literal L. $v_{L(j)}$ is called *special*. For $L \in X_i$, we shall denote u_i also by $v_{L(0)}$, and $\hat{u}_i$ also by $v_{L(a_L+1)}$.
- $r(C_c)$ for each clause C_c, $1 \leq c \leq M$, identifying $\hat{u}_N$ as $r(C_0)$ and u_1 as $r(C_{M+1})$.

We introduce the following edges:

- For each clause $C = L(j) \vee L'(j') \vee L''(j'')$, a clause triangle consisting of the edges $\{(v_{L(j)}, v_{L'(j')})$, $(v_{L'(j')}, v_{L''(j'')})$, $(v_{L''(j'')}, v_{L(j)})\}$.
- For each occurrence $L(j)$ of a literal L in a clause C, a bypass triangle with the edges $\{(v_{L(j)}, \hat{v}_{L(j)})$, $(\hat{v}_{L(j)}, r(C))$, $(r(C), v_{L(j)})\}$.
- A literal path $\{(u_i, v_{L(1)})$, $(v_{L(1)}, v_{L(2)})$, $(v_{L(2)}, v_{L(3)}), \ldots, (v_{L(a_L)}, \hat{u}_i)\}$, denoted $lpath(L)$ for each literal $L \in X_i$.
- For $i = 1, \ldots, N$, *back edges* $(\hat{u}_i, u_i)$. For $i = 1, \ldots, N-1$, *forward edges* $(\hat{u}_i, u_{i+1})$.
- The path $\{(\hat{u}_N, r(C_1))$, $(r(C_1), r(C_2))$, $(r(C_2), r(C_3)), \ldots, (r(C_M), u_1)\}$, which is called the *finishing path*

Figure 3.2 shows an example of the constructed graph. The motivation for this construction is the following: Using the position sets, we intend to force the literal paths of the different variables to be traversed in the natural order, where the only degree of freedom is switching the order between $lpath(x_i)$ and $lpath(\overline{x}_i)$. This switch will correspond to a truth assignment for the variable x_i, by assigning *True* to the literal in X_i whose $lpath$ was visited first. After visiting a special vertex

along this first path, we either visit its clause triangle, or its bypass triangle. Along the other path (the one of the literal assigned *False*) only a bypass triangle can be visited.

Eventually, the finishing path is traversed. Its vertices are visited in the natural order. Upon visiting a vertex $r(C)$, we visit only one bypass triangle - the yet unvisited triangle among those corresponding to the literals of clause C. The truth assignment will satisfy that literal.

We now describe the sets $P(e)$. We use the following notation:

$$\begin{aligned} b_i &= a_{x_i} + a_{\overline{x}_i} \quad \text{for } i = 1, \ldots, N \ . \\ B_i &= \sum_{j=1}^{i} b_j \quad \text{for } i = 0, \ldots, N \ (B_0 = 0) \ . \\ Base_L = Base_{\overline{L}} = Base_i &= 4B_{i-1} + 4(i-1) \quad \text{for } L \in X_i \ . \\ Alternate_L &= Base_i + 4a_{\overline{L}} + 2 \quad \text{for } L \in X_i \ . \\ ClauseBase_c &= Base_{N+1} + 4c \quad 0 \leq c \leq M \ . \end{aligned}$$

- For each forward edge $e = (\hat{u}_{i-1}, u_i)$, $2 \leq i \leq N$, we set $P(e) = \{Base_i\}$. This is intended to ensure that the literal paths are traversed in a constrained order: $lpath(x_i)$ and $lpath(\overline{x}_i)$ are allocated a time interval $[Base_i+1, Base_{i+1}-1]$ of length $4b_i + 3$, during which they must be traversed.

- For each back edge $e = (\hat{u}_i, u_i)$ we set $P(e) = \{Alternate_{\overline{x}_i}, Alternate_{x_i}\}$. This enables either visiting $lpath(x_i)$ first, then e and $lpath(\overline{x}_i)$; or visiting $lpath(\overline{x}_i)$ first, followed by e and $lpath(x_i)$.

- For each literal path edge $e = (v_{L(j)}, v_{L(j+1)})$, with $L \in X_i$, $0 \leq j \leq a_L$, we set $P(e) = \{Base_i + 4j + 1, Alternate_L + 4j + 1\}$. Consecutive edges in a literal path are thus positioned 4 time units apart (allowing a triangle in-between).

- For each clause $C = L_1(j_1) \vee L_2(j_2) \vee L_3(j_3)$ with the clause triangle $\{e_1 = (v_{L_1(j_1)}, v_{L_2(j_2)}), e_2 = (v_{L_2(j_2)}, v_{L_3(j_3)}), e_3 = (v_{L_3(j_3)}, v_{L_1(j_1)})\}$

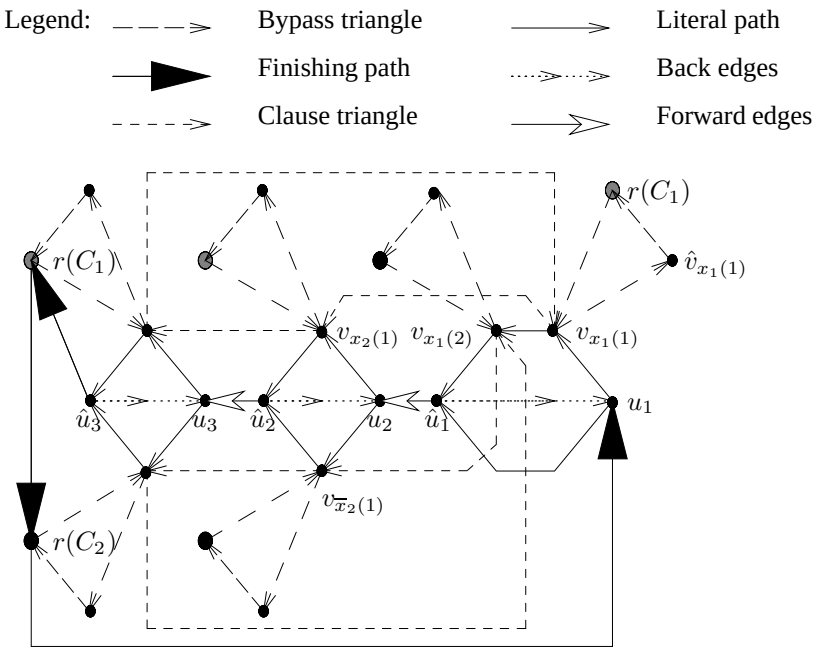


Figure 3.2: An example of the construction for the formula $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3)$. All large grey (black) vertices are actually the same vertex $r(C_1)$ ($r(C_2)$).

such that $L_k \in X_{i_k}$, define $t_k \equiv Base_{i_k} + 4j_k - 2$ and set

$$\begin{aligned} P(e_1) &= \{t_1, t_3 + 1, t_2 + 2\} , \\ P(e_2) &= \{t_2, t_1 + 1, t_3 + 2\} , \\ P(e_3) &= \{t_3, t_2 + 1, t_1 + 2\} . \end{aligned}$$

(See Figure 3.3.) This means that the edges of a clause triangle must be visited consecutively during the traversal of $lpath(L_k)$, for some k. Furthermore, note that this may happen only if $lpath(L_k)$ is traversed immediately after time $Base_{L_k}$, that is, only if it precedes $lpath(\overline{L}_k)$.

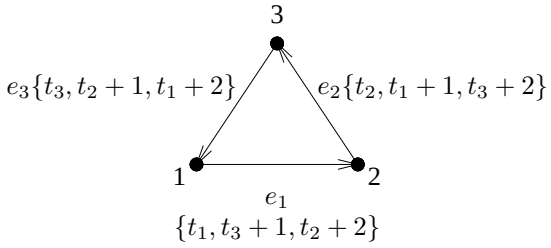


Figure 3.3: A clause triangle, with vertex $v_{L_k(j_k)}$ denoted by k. The allowed positions for each edge appear in braces.

- For each finishing edge $e = (r(C_c), r(C_{c+1}))$, $0 \leq c \leq M$, we set $P(e) = \{ClauseBase_c\}$. This determines the order in which the vertices of the finishing path are traversed. Furthermore, this allows a time slot of length 3, the interval $[ClauseBase_c + 1, ClauseBase_{c+1} - 1]$ for the bypass triangle visited while traversing $r(C_c)$.

- For a bypass triangle with the edges $\{e = (v_{L(j)}, \hat{v}_{L(j)})$, $e' = (\hat{v}_{L(j)}, r(C_c))$, $e'' = (r(C_c), v_{L(j)})\}$, we set:

$$\begin{aligned} P(e) &= \{Base_L + 4j - 2, Alternate_L + 4j - 2, ClauseBase_c - 2\} , \\ P(e') &= \{Base_L + 4j - 1, Alternate_L + 4j - 1, ClauseBase_c - 1\} , \\ P(e'') &= \{Base_L + 4j, Alternate_L + 4j, ClauseBase_c - 3\} . \end{aligned}$$

This means that the bypass triangle edges must be visited consecutively, and there are three possible time slots for that:

- While traversing $lpath(L)$, before traversing $lpath(\overline{L})$.
- While traversing $lpath(L)$, after traversing $lpath(\overline{L})$.
- While traversing $r(C_c)$ along the finishing path.

The reduction is obviously polynomial. We now prove validity of the construction.

$\Leftarrow$ Suppose that F is satisfiable. We will show that (D, P) is a "yes" instance of the 3-positional Eulerian path problem. Let ϕ be a truth assignment satisfying F. For each clause C_c, let $L_c(j_c)$ be a specific literal occurrence satisfying C_c.

We describe an Eulerian path π in D. For $c = 0, \ldots, M$, set $\pi(ClauseBase_c) = (r(C_c), r(C_{c+1}))$. Set $\pi(Base_i) = (\hat{u}_{i-1}, u_i)$, for $i = 2, \ldots, N$. For all i, if $\phi(x_i) =$*True*, set $\pi(Alternate_{\overline{x}_i}) = (\hat{u}_i, u_i)$. Otherwise, set $\pi(Alternate_{x_i}) = (\hat{u}_i, u_i)$.

For each literal $L \in X_i$:

- If $\phi(L) =$*True*: For each $0 \leq j \leq a_L$, set $\pi(Base_i + 4j + 1) = (v_{L(j)}, v_{L(j+1)})$ (see Figure 3.4, top).

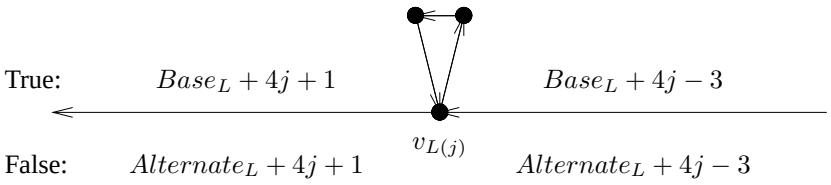


Figure 3.4: Either a clause triangle or a bypass triangle must be traversed upon visiting a special vertex $v_{L(j)}$, due to time constraints. Edge positions in case L is assigned *True* (*False*) are shown at the top (bottom).

We further distinguish between two cases:

* If $L(j) = L_c(j_c)$ for the clause $C_c = L(j) \vee L'(j') \vee L''(j'')$ in which $L(j)$ occurs, then set π to visit the edges of the clause

triangle of C_c as follows:

$$\begin{aligned}\pi(Base_L + 4j - 2) &= (v_{L(j)}, v_{L'(j')}) \ , \\ \pi(Base_L + 4j - 1) &= (v_{L'(j')}, v_{L''(j'')}) \ , \\ \pi(Base_L + 4j) &= (v_{L''(j'')}, v_{L(j)}) \ .\end{aligned}$$

Furthermore, in this case we set π to visit the edges of the bypass triangle of $L(j)$ as follows:

$$\begin{aligned}\pi(ClauseBase_c - 3) &= (r(C_c), v_{L(j)}) \ , \\ \pi(ClauseBase_c - 2) &= (v_{L(j)}, \hat{v}_{L(j)}) \ , \\ \pi(ClauseBase_c - 1) &= (\hat{v}_{L(j)}, r(C_c)) \ .\end{aligned}$$

$*$ Otherwise, $L(j) \neq L_c(j_c)$ for the clause C_c in which $L(j)$ occurs. In this case we set π to visit the edges of the bypass triangle of $L(j)$ as follows:

$$\begin{aligned}\pi(Base_L + 4j - 2) &= (v_{L(j)}, \hat{v}_{L(j)}) \ , \\ \pi(Base_L + 4j - 1) &= (\hat{v}_{L(j)}, r(C_c)) \ , \\ \pi(Base_L + 4j) &= (r(C_c), v_{L(j)}) \ .\end{aligned}$$

– If $\phi(L) =$*False*: For each $0 \leq j \leq a_L$, set $\pi(Alternate_L + 4j + 1) = (v_{L(j)}, v_{L(j+1)})$. Furthermore, in this case we set π to visit the edges of the bypass triangle of $L(j)$ as follows (see Figure 3.4, bottom):

$$\begin{aligned}\pi(Alternate_L + 4j - 2) &= (v_{L(j)}, \hat{v}_{L(j)}) \ , \\ \pi(Alternate_L + 4j - 1) &= (\hat{v}_{L(j)}, r(C_c)) \ , \\ \pi(Alternate_L + 4j) &= (r(C_c), v_{L(j)}) \ .\end{aligned}$$

It is easy to see that π is a permutation of the edges, and if $\pi(k) = (u, v)$,$\pi(k+1) = (u', v')$ then $v = u'$. Hence, π is an Eulerian path. Furthermore, by our construction π is compliant with (D, P), proving that (D, P) is a "yes" instance.

$\Rightarrow$ Let π be an Eulerian path compliant with (D, P). We shall construct an assignment ϕ satisfying F. In order to determine $\phi(x_i)$ we consider the edge

$\pi(Base_i + 1)$. By construction, $\pi(Base_i + 1) = (u_i, v_{L(1)})$ for $L \in X_i$. We therefore set $\phi(L)$ =*True* (and of course $\phi(\overline{L})$ =*False*). We observe that for any other edge $e' = (v_{L(j)}, v_{L(j+1)})$ along $lpath(L)$, we must have $\pi(Base_i + 4j + 1) = e'$ if and only if $\phi(L)$ =*True*.

We now prove that ϕ satisfies each clause $C_c = L_1(j_1) \vee L_2(j_2) \vee L_3(j_3)$ of F. Consider the clause triangle of C_c: $\{e_1 = (v_{L_1(j_1)}, v_{L_2(j_2)})$, $e_2 = (v_{L_2(j_2)}, v_{L_3(j_3)})$, $e_3 = (v_{L_3(j_3)}, v_{L_1(j_1)})\}$. Denote $t_k = Base_{L_k} + 4j_k - 2$. By the positional constraints, there exists some $1 \leq k \leq 3$ for which $\pi(t_k) = e_k$. The edge e preceding e_k in π must have $t_k - 1 = Base_{L_k} + 4(j_k - 1) + 1 \in P(e)$. The only such edge entering $v_{L_k(j_k)}$ is the literal path edge $(v_{L_k(j_k-1)}, v_{L_k(j_k)})$. Therefore, $\phi(L_k)$ =*True*, satisfying C_c.

This proves that F is satisfiable if and only if (D, P) is a "yes" instance, completing the proof of Theorem 3.2.8. ■

Observe that the graph constructed in the proof of Theorem 3.2.8 has in-degree and out-degree bounded by 4, giving rise to the following result:

Corollary 3.2.9 *3-PEP is NP-complete, even when restricted to graphs with in-degree and out-degree bounded by 4 .*

Henceforth, we call this restricted problem *(3,4)-PEP*. We comment that a slight modification of the construction results in a graph whose in-degree and out-degree are bounded by 2.

3.2.4 3-Positional SBH is NP-Complete

We show in this section that the problem of sequencing by hybridization with at most 3 positions per spectrum element is NP-complete, even if each element in the spectrum is unique. The proof is by reduction from (3,4)-PEP.

Theorem 3.2.10 *The 3-PSBH problem is NP-complete, even if all spectrum elements are of multiplicity one.*

Proof: It is easy to see that the problem is in NP. We reduce (3,4)-PEP to 3-PSBH. Let $(D = (V, E), P)$ be an instance of (3,4)-PEP. Let $k = \lceil \log_4 |V| \rceil + 2$, $p = 3k + 1$ and $c = p + 1$. In order to construct an instance of 3-PSBH we first encode the edges and vertices of D. In the following, we denote string concatenation by $|$. We let σ_1 ='A', σ_2 ='C', σ_3 ='G' and σ_4 ='T'.

To each $v \in V$ we assign a distinct string in Σ^{k-2}. We add a leading 'T' symbol and a trailing 'T' symbol to this string, and call the resulting k-long string the *name* of v. We also assign the string 'A ... A' of length k to encode a *space*. Each vertex is encoded by a $3k$-long string containing two copies of its name separated by a space. We denote the encoding of v by $en(v)$. Each edge $(u, v) \in E$ is encoded by two symbols chosen as follows: Let $N_{out}(u) = \{v_1, \ldots, v_l\}$, where $v = v_i$ for some i, and $l \leq 4$. Let $N_{in}(v) = \{u_1, \ldots, u_r\}$, where $u = u_j$ for some j, and $r \leq 4$. Then (u, v) is encoded by $\sigma_i|\sigma_j$, and we denote its encoding by $en(u, v)$. We call $EN(u, v) \equiv en(u)|en(u, v)|en(v)$ the *representative string* of (u, v) (see Figure 3.5).

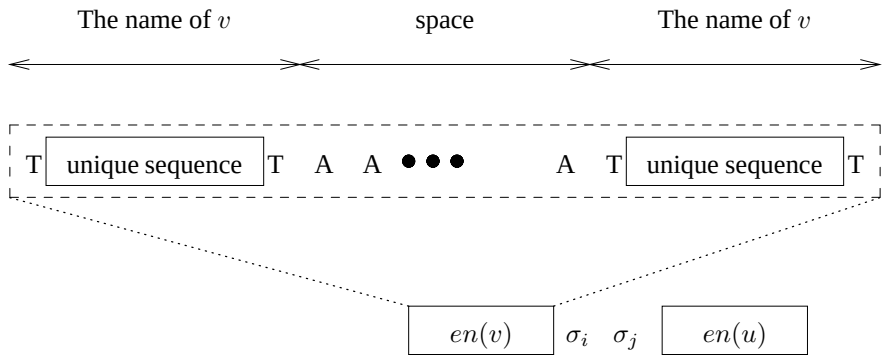


Figure 3.5: The encoding of vertices and edges into representative strings.

We now construct a 3-PSBH instance, i.e., a spectrum S with position constraints T, as follows: For every edge $(u, v) \in E$ the set S contains all p-long substrings of the $2p$-long string $EN(u, v)$ (c substrings in total). Let $s^i_{(u,v)}$ denote the i-th such substring, $i = 0, \ldots, p$. Let $P((u, v)) = \{t_1, \ldots, t_l\}$, $1 \leq l \leq 3$, be the set of allowed positions for (u, v). Then we set $T(s^i_{(u,v)}) = \{c(t_1 - 1) + i, \ldots, c(t_l - 1) + i\}$ for all i (note that substring positions are numbered starting at zero).

Lemma 3.2.11 *Each of the p-long substrings in S is unique.*

Proof: Suppose that $s^i_{(u,v)} = s^j_{(w,z)}$. By construction, either $s^i_{(u,v)}$ contains a complete space, or begins with a (possibly empty) suffix of a space. The position of the space in the first case, or the length of the suffix in the second case, uniquely determines i, which implies that $i = j$. Furthermore, $s^i_{(u,v)}$ contains a name of a vertex plus a unique symbol identifying an edge entering or leaving that vertex, implying that $(u, v) = (w, z)$. ■

We now show validity of the reduction.

$\Leftarrow$ Suppose that $\pi = (v_0, v_1), (v_1, v_2), \ldots, (v_{m-1}, v_m)$ is a solution of the (3,4)-PEP instance. We claim that the 3-PSBH instance is also solved by $X = en(v_0)|en(v_0, v_1)|en(v_1)|en(v_1, v_2)|\ldots|en(v_{m-1}, v_m)|en(v_m)$. By Lemma 3.2.11, each p-long substring of X occurs exactly once in X. As π visits all edges in D, we have that S is the p-spectrum of X. The fact that position constraints are satisfied follows directly from the construction.

$\Rightarrow$ Let X be a solution of the 3-PSBH instance. Consider the m substrings of length p, whose starting positions in X are integer multiples of c. By the position constraints, the r-th such substring is an encoding of some vertex v_r, followed by a symbol σ_{i_r}. Denote by w_r the i_r-th out-neighbor of v_r. We prove that $\pi = (v_1, w_1), \ldots, (v_m, w_m)$ is an Eulerian path compliant with (D, P).

Since each string in the p-spectrum of X is unique, π is a permutation of the edges in D. To prove that π is a path in D we have to show that $w_r = v_{r+1}$ for $r = 1, \ldots, m-1$. Let x be the p-long substring of X starting at position $(r-1)c + 2k$. We observe that x must begin with the last k symbols of $en(v_r)$, which compose $name(v_r)$, followed by σ_{i_r}, some symbol, and the first $2k-1$ symbols of $en(v_{r+1})$, which contain $name(v_{r+1})$. The uniqueness of $name(v_r)$, $name(v_{r+1})$ and the index i_r among the out-neighbors of v_r, implies that $w_r = v_{r+1}$. The claim now follows, since position constraints are trivially satisfied by π.

■

3.2.5 Interval PSBH is NP-Complete

In this section we study the complexity of Interval PSBH. In Hannenhalli et al. [1996] it was proved that the related Interval PEP problem is NP-complete, but the complexity of Interval PSBH was left open. We show below that this problem is NP-complete.

Theorem 3.2.12 *The Interval PSBH problem is NP-complete, even if all sets of allowed positions are intervals of equal length.*

Proof: The Interval PEP problem is NP-complete, even if each vertex has in-degree and out-degree at most two, and the sets of allowed positions are intervals of equal length [Hannenhalli et al., 1996]. We reduce this restriction of Interval PEP to Interval PSBH, analogously to the reduction used in the proof of Theorem 3.2.10. In the following, we use the same notation as in the proof of Theorem 3.2.10.

Let $(D = (V, E), P)$ be an instance of Interval PEP, with each vertex having in-degree and out-degree at most two. Recall, that $EN(u, v)$ denotes a $2p$-long representative string of an edge $(u, v) \in E$ (see Figure 3.5). We now construct an Interval PSBH instance (S, P'). The spectrum S is the same as in Theorem 3.2.10, containing the p-long substrings $s^i_{(u,v)}$ of $EN(u, v)$, for each $(u, v) \in E$. Note, that this construction of S requires that the in-degree and out-degree of each vertex in D will be at most four, a condition which is satisfied by the above restriction of Interval PEP. The set of allowed positions for each substring is defined as follows. Let (u, v) be an edge of E, and suppose that $P((u, v)) = [t_{low}, t_{high}]$. Then, for all i, we set the interval of allowed positions $P'(s^i_{(u,v)})$ to $[c(t_{low} - 1) + i, c(t_{high} - 1) + i]$.

It is enough to show that if X solves the Interval PSBH instance (S, P'), then there is an Eulerian path compliant with (D, P). The proof of the other direction follows immediately from that of Theorem 3.2.10.

We first claim that for any $(u, v) \in E$ the string $EN(u, v)$ occurs exactly once along X. Let r be a $(p-1)$-long substring of $EN(u, v)$, which is neither a prefix, nor a suffix of $EN(u, v)$. By construction of $EN(u, v)$, either r contains the name

of u and an identification of the outgoing edge to v, or r contains the name of v and an identification of the incoming edge from u. Therefore, similarly to the proof of Lemma 3.2.11, we have that r occurs only once in $EN(u,v)$, and does not occur in any other representative string. As $s^0_{(u,v)}, \ldots, s^p_{(u,v)}$ are the p-long substrings of $EN(u,v)$, there exists a unique index $i \in \{0, \ldots, p-1\}$ such that r is a suffix of $s^i_{(u,v)}$ and a prefix of $s^{i+1}_{(u,v)}$. Since X is a valid solution to the Interval PSBH instance, it contains exactly one copy $X_l \ldots X_{l+p-1}$ of $s^i_{(u,v)}$. Since $i < p$, the positional constraints P' assure that X is not terminated at X_{l+p-1}. Hence, $X_{l+1} \ldots X_{l+p}$ is a spectrum element whose prefix is r, and therefore, must be a copy of $s^{i+1}_{(u,v)}$. We conclude, that for each $0 \leq i \leq p-1$, the occurrence of $s^i_{(u,v)}$ is immediately followed by the occurrence of $s^{i+1}_{(u,v)}$. The claim follows.

Let $EN(u_1,v_1), \ldots, EN(u_m,v_m)$ be the order of occurrence of representative strings along X. For every $1 \leq j < m$, $EN(u_j,v_j)$ and $EN(u_{j+1},v_{j+1})$ do not share any p-long substring (see Lemma 3.2.11). Consequently, their overlap along X is of length at most $p-1$. But $|X| = m(p+1)+p-1$, implying that for all $1 \leq j < m$, $EN(u_j,v_j)$ and $EN(u_{j+1},v_{j+1})$ have a $(p-1)$-long overlap. Hence, $en(v_j) = en(u_{j+1})$ and $v_j = u_{j+1}$. Therefore, denoting $v_0 = u_1$, the string X has the form

$$X = en(v_0)|en(v_0,v_1)|en(v_1)|\ldots|en(v_{m-1})|en(v_{m-1},v_m)|en(v_m)$$

It is easy to see, that $\pi = (v_0,v_1), \ldots, (v_{m-1},v_m)$ is an Eulerian path compliant with (D,P).

■

3.2.6 A Parameterized Algorithm for Interval PSBH

Hannenhalli et al. have given a linear-time algorithm to solve the parametric version of the Interval PEP (and hence, the Interval PSBH) problem, where the parameter k is an upper bound on the sizes of the intervals of allowed positions for each edge [Hannenhalli et al., 1996]. We provide a faster algorithm for the problem, which uses the same basic idea as in Hannenhalli et al. [1996], and runs in $O(mk^{1.5}4^k)$ time, improving upon the $O(mk^3 \log k 4^k)$ time bound of Hannenhalli et al. [1996].

For simplicity, we shall describe our algorithm when all allowed intervals have length exactly k. Let $(D = (V, E), P)$ be the input positional graph, where $P(e) = [l_e, l_e + k - 1]$ for every $e \in E$. For every $1 \leq i \leq m$ define $\underline{i} \equiv \max\{i - k + 1, 1\}$ and $\overline{i} \equiv \min\{i + k - 1, m\}$. For every $1 \leq i \leq m$ define $\delta_i \equiv \{e \in E : l_e = i\}$ and $\Delta_i \equiv \{e \in E : i \in P(e)\}$. That is, Δ_i is the set of edges for which position i is allowed.

The naive approach to the problem would be to try all possible Eulerian paths in D, and test for each one whether it is compliant with P. However, the number of Eulerian paths in D might be exponential (in m). Instead, we iteratively construct for every $i = 1, \ldots, m + 1$, a list Φ_i of pairs (v, S), such that v is the last vertex of some path of length $i - 1$, and S is the list of edges that may extend the path from v. The size of each list Φ_i can be at most exponential in k, and this leads to the improved bound. The algorithm is summarized in Figure 3.6.

Compute δ_i for all i.
If for some i, $|\delta_i| > k$ **then** return *False*.
Initialize $\Phi_1 \leftarrow \{(v, \delta_1) : \exists w, (v, w) \in \delta_1\}$, $\Phi_{m+1} \leftarrow \emptyset$.
Initialize $\Delta_1 \leftarrow \delta_1$, $i \leftarrow 1$.
While $\Phi_i \neq \emptyset$ and $i \leq m$ **do**:
    $i \leftarrow i + 1$.
    $\Delta_i \leftarrow \Delta_{i-1} \cup \delta_i \setminus \delta_{\underline{i}-1}$.
    If $|\Delta_i| \geq 2k$ **then** return *False*.
    $$\Phi_i \leftarrow \left\{ (w, S \cup \delta_i \setminus \{(v, w)\}) : \begin{array}{l} (v, S) \in \Phi_{i-1}, (v, w) \in S, \\ S \cap \delta_{\underline{i}-1} \subseteq \{(v, w)\} \end{array} \right\}.$$
If $\Phi_{m+1} \neq \emptyset$ **then** return *True*.
Else return *False*.

Figure 3.6: An algorithm for Interval PEP.

The following lemma, which is mentioned in Hannenhalli et al. [1996], is crucial for the analysis of the algorithm. The subsequent theorem proves the correctness of the algorithm.

Lemma 3.2.13 *If (D, P) has a compliant Eulerian path, then $|\Delta_i| \leq 2k-1$ for all i.*

Proof: Let π be an Eulerian path compliant with (D, P). Then for all i: $\Delta_i \subseteq \{\pi(\underline{i}), \ldots, \pi(\bar{i})\}$. ■

Theorem 3.2.14 *The algorithm returns* True *if and only if there is an Eulerian path compliant with (D, P).*

Proof: Let us call a pair (w, S) *i-valid* if there exists a path $\{w_1, \ldots, w_i = w\}$ in D such that:

1. $j \in P((w_j, w_{j+1}))$ for all $1 \leq j < i$.
2. $S = \Delta_i \setminus \{(w_1, w_2), \ldots, (w_{i-1}, w_i)\}$.
3. $\bigcup_{j=1}^{i-1} \delta_j \subseteq \{(w_1, w_2), \ldots, (w_{i-1}, w_i)\}$.

Intuitively, (1) ensures that all edges in the path occupy allowed positions, (2) ensures that the next (i-th) edge is both allowed for position i, and was not used already, and finally, (3) ensures that any edge that had to be used before position i, was indeed used. We first prove by induction on i, that Φ_i is the set of all i-valid pairs. The case $i = 1$ is obvious. Suppose the claim holds for $i-1$. By the induction hypothesis, Φ_{i-1} is the set of all $(i-1)$-valid pairs. By definition, $\Phi_i = \{(w, S \cup \delta_i \setminus \{(v, w)\}) : (v, S) \in \Phi_{i-1}, (v, w) \in S, S \cap \delta_{\underline{i}-1} \subseteq \{(v, w)\}\}$. The claim follows.

If π is an Eulerian path compliant with (D, P), then $\pi(i) \in \Delta_i$ for all $1 \leq i \leq m$, and therefore, $(v, \emptyset)$ is an $(m+1)$-valid pair for the vertex v at which π ends. On the other hand, if $(v, \emptyset)$ is an $(m+1)$-valid pair, then there exists a path in D which traverses all edges and satisfies the positional constraints. ■

We now analyze the complexity of the algorithm. To this end, we define the sets $\Psi_i \equiv \{S : \exists w, (w, S) \in \Phi_i\}$ and $V_i \equiv \{v : \exists S, (v, S) \in \Phi_i\}$.

Lemma 3.2.15 *For each i there exists a constant s_i, such that all sets $S \in \Psi_i$ satisfy $|S| = s_i$.*

Proof: The proof is by induction on i. Correctness for $i = 1$ is obvious. Suppose the claim holds for $i - 1$. Then by definition, for each $S \in \Psi_i$, there exists a set $S' \in \Psi_{i-1}$ such that $|S| = |S'| + |\delta_i| - 1$. Hence, $s_i = s_{i-1} + |\delta_i| - 1$, which proves the claim for i. ■

Corollary 3.2.16 *For all i, $|\Psi_i| = O(\frac{4^k}{\sqrt{k}})$.*

Proof: Every set $S \in \Psi_i$ satisfies $S \subseteq \Delta_i$. Hence, using Lemma 3.2.13 $|\Psi_i| \leq \binom{|\Delta_i|}{s_i} \leq \binom{2k-1}{k} = O(\frac{4^k}{\sqrt{k}})$. ■

Theorem 3.2.17 *The algorithm can be implemented in $O(mk^{1.5}4^k)$ time and $O(m+k^{1.5}4^k)$ space.*

Proof: The initialization of δ_i for all i takes $O(m)$ time and space, since $\sum_{i=1}^{m} |\delta_i| = m$. We sort all edges in E according to their first allowed position, breaking ties arbitrarily, and renumber the edges $e_1, \ldots, e_m$ in this order. This takes $O(m)$ time using bucket sort. We also compute in $O(m)$ time the numbers $n(e_i) \equiv i \pmod{2k-1}$, for each $e_i \in E$. Hereafter, we assume that $|\Delta_i| < 2k$ for all i, as otherwise the algorithm aborts returning a negative answer. A key property of the numbering $n(\cdot)$ is that for each Δ_i, if $e, e' \in \Delta_i$ then $n(e) \neq n(e')$. This follows since the edges in Δ_i must appear contiguously in the order $e_1, \ldots, e_m$, and $|\Delta_i| \leq 2k - 1$.

We use the following data structure to keep Δ_i and Φ_i:

- Let $v_1, \ldots, v_n$ be an arbitrary ordering $\prec$ of the vertices in V. Δ_i is kept as a linked list of edges, sorted lexicographically such that $(x_1, x_2) < (y_1, y_2)$ if and only if $x_2 \prec y_2$, or $x_2 = y_2$ and $x_1 \prec y_1$.
- For each $v \in V_i$ we keep a linked list $L_i(v)$ containing all sets $S \in \Psi_i$ such that $(v, S) \in \Phi_i$. We represent a subset $S \subseteq \Delta_i = \{e^i_1, \ldots, e^i_{|\Delta_i|}\}$ by a $(2k-1)$-bit vector $c_i(S)$, whose $n(e^i_j)$-th bit is one if and only if $e^i_j \in S$.

The initialization of Φ_1 requires constructing $O(k)$ linked lists, each containing the $(2k-1)$-bit vector $c_1(\Delta_1)$. This takes $O(k^2)$ time. The computation (and sorting) of Δ_{i+1} from Δ_i takes $O(k \log k)$ time.

The construction of Φ_{i+1} from Φ_i can be done as follows: We consider the edges in Δ_i one at a time according to their lexicographic order. Recall, that these edges are sorted so that edges ending at the same vertex occur consecutively. Thus, for each $w \in V_{i+1}$ we can consecutively traverse all edges in Δ_i that enter w. Note, that we do not know V_{i+1} exactly, but only use the simple fact that $V_{i+1} \subseteq \{w : \exists v, (v, w) \in \Delta_i\}$.

We now construct the linked lists $L_{i+1}(w)$ for every $w \in V_{i+1}$. For each $(v, w) \in \Delta_i$ and for each $S' \in L_i(v)$ we add $S = S' \cup \delta_{i+1} \setminus \{(v, w)\}$ to $L_{i+1}(w)$ if and only if $(v, w) \in S'$ and $S' \cap \delta_{\underline{i}} \subseteq \{(v, w)\}$. In order to add S, we compute the bit vector $c_{i+1}(S)$ in $O(k)$ time. To prevent generating duplicate pairs (w, S) in $L_{i+1}(w)$, we keep an auxiliary 2^{2k-1}-bit vector B which is initialized to zero in the beginning of the algorithm (in $O(4^k)$ time). We add $c_{i+1}(S)$ to $L_{i+1}(w)$ if and only if $B_{c_{i+1}(S)} = 0$, and in this case we also set $B_{c_{i+1}(S)} = 1$. When we finish constructing $L_{i+1}(w)$, we traverse its elements and reset the corresponding bits in B to zero. If no set was added to any linked list during the iteration, we declare that the graph has no compliant Eulerian path and halt. Since $|L_i(v)| = O(|\Psi_i|)$ for each $v \in V_i$, Φ_{i+1} can be computed in $O(|\Delta_i| \cdot |\Psi_i| \cdot k)$ time. By Lemma 3.2.13 and Corollary 3.2.16, this amounts to $O(k^{1.5}4^k)$ time.

We conclude that the time complexity of the algorithm is $O(m + k^2 + 4^k + mk \log k + mk^{1.5}4^k) = O(mk^{1.5}4^k)$. To save space, we reuse the storage area of Φ_i and Δ_i once Φ_{i+1} and Δ_{i+1} have been computed. Hence, the space complexity of the algorithm is $O(m + k^{1.5}4^k)$. ■

We note, that the algorithm can be modified to generate also one (or all) of the compliant Eulerian paths, e.g., by keeping a record of the last edge(s) leading to each i-valid pair. We also note, that our algorithm can be extended to handle the PEP problem even when the sets of allowed positions for each edge need only be subsets of fixed-length intervals.

3.3 Spectrum Alignment: Efficient Resequencing by Hybridization

Recent high-density microarray technologies allow, in principle, the determination of all k-mers that appear along a DNA sequence, for $k = 8 - 10$ in a single experiment on a standard chip. The k-mer contents, also called the *spectrum* of the sequence, is not sufficient to uniquely reconstruct a sequence longer than a few hundred bases. We have devised a polynomial algorithm that reconstructs the sequence, given the spectrum and a homologous sequence. This situation occurs, for example, in the identification of single nucleotide polymorphisms (SNPs), and whenever a homologue of the target sequence is known. The algorithm is robust, can handle errors in the spectrum and assumes no knowledge of the k-mer multiplicities. Our simulations show that with realistic levels of SNPs, the algorithm correctly reconstructs a target sequence of length up to 2000 nucleotides when a polymorphic sequence is known. The technique is generalized to handle profiles and HMMs as input instead of a single homologous sequence.

3.3.1 Introduction

In this chapter we propose a new method to reconstruct a DNA target sequence. The method combines spectrum data (obtainable from a universal DNA chip), a known homologous reference sequence, and novel computational techniques, similar to those used for analysis of sequence homology. Section 3.1 provided introduction to universal DNA arrays and SBH. This section presents additional necessary background, surveying sequence homology and its computational formulation.

Similar Sequences are Ubiquitous

The understanding that many DNA sequences resemble each other is fundamental to biology. This similarity is due to the process of evolution: numerous contemporary sequences have evolved by mutations from a single ancestral molecule. Such related (*homologous*) sequences exhibit similarity to their unique ancestor, and thus

to each other. This general scenario is routinely encountered in genome analysis:

- The genomic sequences of all individual organisms of the same species are almost identical. Current estimates of the variability rate among all humans, for instance, amount to one *Single Nucleotide Polymorphism (SNP)* per $100 - 300$ base pairs [National Center for Biotechnology Information, 2000]. Of course, when comparing DNA of a specific individual to the genomic reference, these two sequences would share the same allele in many of the polymorphic sites. Hence, the fraction of their base pairs on which they differ would be considerably smaller.

- Much of the eukaryotic genome is composed of *repetitive elements* - sequences which recur in thousands or millions of copies. Different repeats are usually $90 - 95\%$ identical.

- Various duplications of large genomic segments occurred during the course of evolution. In some cases, the whole genome was duplicated. This process created many homology relationships. In particular, duplications which include coding regions gave rise to several gene copies. At a later stage, these genes diverged and mutated, forming a gene family.

- The genomes of different, phylogenetically related species have homologous segments. The rate of sequence identity may approach 100% identity when comparing highly conserved genomic regions of closely related species, but may also drop to the twilight zone of near-random expected resemblance, for more divergent segments, and more distant taxa.

As sequence data accumulates in an accelerated rate, an increasing number of sequencing targets have a known homologous sequence. This motivates the development of new sequencing strategies which utilize homology information. Hybridization has been successfully used for sequencing SNPs given the reference genomic sequence, using custom made microarrays [Cargill et al., 1999, Hacia et al., 1999, Hacia, 1999]. To the best of our knowledge, this study is the first proposal to use standard, universal chips and homology information for sequencing.

Exploiting Homology Computationally

Sequence similarity is perhaps the most studied issue in bio-informatics (cf. Durbin et al. [1998b]). The evolution of homologous sequences from a common ancestral origin is mainly due to nucleotide substitution: a stochastic process which can be described by a nucleotide substitution matrix [Jukes and Cantor, 1969, Kimura, 1980a]. This description facilitates calculating the likelihood that one sequences is homologous to another.

Insertions and deletions of nucleotides also occur during evolution of homologous sequences, though at lower rates. This calls for *aligning* the two sequences, i.e., matching pairs of their loci according to the common origin of the paired nucleotides. The well known Smith-Waterman dynamic programming algorithm [Smith and Waterman, 1981] computes the alignment score with affine gap penalties. Such a score can be formulated as the log-likelihood of the data using Hidden Markov Models (HMMs) [Durbin et al., 1998b, chapter 4]. The latter are often explicitly used to generalize the homology concept, and to model alignment against a family of sequences [Krogh et al., 1994, Eddy, 1996].

Our Contribution

We describe here a new method for reconstructing the target sequence, by combining information on a reference sequence with experimental spectrum data obtainable from a standard chip. We call the technique resequencing by hybridization, or *spectrum alignment* since the algorithm attempts to find the best ”alignment” of the reference sequence with the spectrum. Technically, this is done by developing a dynamic programming algorithm which runs on the de-Bruijn graph and the reference sequence simultaneously. The algorithm is polynomial, and can handle inexact, probabilistic information on the spectrum, which is common in hybridization results. Unlike other algorithms proposed for SBH and its extensions, it does not assume knowledge of the multiplicities of the k-mers in the sequence. We also show how to extend our results to handle profiles and HMMs as homology information, instead of a particular reference sequence,

The section is organized as follows: Section 3.3.2 gives background on SBH and on homology, and necessary terminology is presented. In Section 3.3.3 we provide an algorithm for resequencing by hybridization, allowing mismatches but no insertions or deletions in the target with respect the reference sequence. Section 3.3.4 extends the methods to deal with the general case allowing insertions, deletions and substitutions . Finally, in Section 3.3.5, we present preliminary results on simulated data,. and discuss future directions in Section 3.3.6.

3.3.2 Preliminaries

Scoring by the Hybridization Data

Let $\Sigma = \{A, C, G, T\}$ be our alphabet, with $M = 4$ being the alphabet size. We denote sequences by a string over Σ between angle brackets ($\langle\rangle$). A *k-spectrum* of a sequence $\mathcal{T} = \langle t_1 t_2 \cdots t_L \rangle$ is the set of all k-long substrings (k-mers) of $\mathcal{T}$. For each k-mer $\vec{x} = \langle x_1 x_2 \cdots x_k \rangle \in \Sigma^k$, we define $\mathcal{T}(\vec{x})$ to be 1 if $\vec{x}$ is a substring of $\mathcal{T}$, and 0 otherwise. We denote $K = M^k$.

A hybridization experiment measures, for each k-mer $\vec{x} \in \Sigma^k$, the intensity of ots hybridization signal. For our puspose, the relevant information in such a signal is the probabilities $P_0(\vec{x}), P_1(\vec{x})$ of this observed intensity assuming $\mathcal{T}(\vec{x}) = 0$, and $\mathcal{T}(\vec{x}) = 1$, respectively. We therefore define a *probabilistic spectrum* (PS) to be a pair (P_0, P_1) of functions $P_i : \Sigma^k \mapsto [0, 1]$. PS is assumed to be the result of the hybridization experiment, which we analyse. If the experiment were *perfect*, i.e., if the probabilities were all zero or one (with $P_0(\vec{x}) + P_1(\vec{x}) \equiv 1$), then the hybridization data would be translated into a k-spectrum. In such a case we could perfectly determine the occurrence of a k-mer in the sequence by examining the hybridization signal. In practice, though, both $P_0(\vec{x}), P_1(\vec{x})$ are positive, and any deterministic binarization of the hybridization signal will contain errors. Our algorithms will therefore use the probabilistic data. We assume knowledge of these error parameters even prior to sequence reconstruction.

Many suggested combinatorial models for SBH assume, for theoretical convenience, that the multiplicities of k-mer occurrences are known [Pevzner, 1989,

Preparata et al., 1999]. This assumption is impractical using current technology and our algorithm does not rely on it (In fact, when all multiplicities are 1 our performance improves.)

The *de-Bruijn graph* is a directed graph $G_k(V, E)$ whose vertices are labeled by all the $(k-1)$-mers $V = \Sigma^{k-1}$, and its edges are labeled by k-mers, $E = \Sigma^k$. The edge labeled $\langle x_1 x_2 \cdots x_k \rangle$ connects the vertex $\langle x_1 x_2 \cdots x_{k-1} \rangle$ to the vertex $\langle x_2 \cdots x_k \rangle$. Whenever k is clear from context we omit it, referring to the De-Bruijn graph as G. There is a 1-1 correspondence between candidate L-long target sequences and $(L - k + 1)$-long paths in G, whose edge labels comprise the target spectrum. In case the spectrum data-set is perfect and the multiplicities are known, omitting all zero probability edges from G one gets one gets Pevzner's formulation, i.e., every solution is an Eulerian path [Pevzner, 1989]. For our more general formulation we devise a scoring scheme for paths, and search for the best scoring path in G. Hereafter, we interchangeably refer to edges and their labels, and also to sequences and their corresponding paths. Observe that since k-mers may re-occur, paths do not have to be simple.

We assume that hybridization results of different oligos are mutually independent (see discussion). Hence, the *experimental likelihood* $L^e(\hat{\mathcal{T}})$ of a candidate target sequence $\hat{\mathcal{T}}$ is

$$L^e(\hat{\mathcal{T}}) = Prob(\text{PS}|\hat{\mathcal{T}}) = \prod_{\vec{x} \in \Sigma^k} P_{\hat{\mathcal{T}}(\vec{x})}(\vec{x}) \tag{3.7}$$

Taking (base 2) logarithm, define $w(\vec{x}) = \log \frac{P_1(\vec{x})}{P_0(\vec{x})}$. Throughout, when handling probabilities, some of which are perfect, problems of division by zero might occur. We get around those by implicitly perturbing perfect probabilities to δ and $1 - \delta$. We can thus write:

$$\log L^e(\hat{\mathcal{T}}) = \sum_{\vec{x} \in \Sigma^k} \log P_0(\vec{x}) + \sum_{\hat{\mathcal{T}}(\vec{x})=1} w(\vec{x}) \tag{3.8}$$

The first term is a constant, independent of $\hat{\mathcal{T}}$, and is omitted hereafter.

Let $p = e_0, \ldots, e_{L-k}$ be the path in G corresponding to $\hat{\mathcal{T}}$. Then

$$\log \widetilde{L^e}(\hat{\mathcal{T}}) = \sum_{i=0}^{L-k} w(e_i) \tag{3.9}$$

is an approximate likelihood score, deviating from the true likelihood whenever an edge is revisited along p. $\widetilde{L^e}(\hat{\mathcal{T}})$ has the advantage of being easily computable in a recursive manner:

$$\log \widetilde{L^e}(e_0, \ldots, e_l) = \log \widetilde{L^e}(e_0, \ldots, e_{l-1}) + w(e_l) \tag{3.10}$$

Pevzner assumes perfect hybridization data [Pevzner, 1989]. In this case, every path in G whose likelihood score equals one is a possible solution to the SBH problem, while all other paths have probability zero. Indeed, Pevzner simply discards improbable k-mers from G. One can handle imperfect data in an analogous manner, by discarding edges with probability smaller than ϵ. After this procedure, isolated vertices correspond to highly improbable $(k-1)$-mers, and can be discarded as well. We denote the resulting graph $[G]_\epsilon = ([V]_\epsilon, [E]_\epsilon)$, and call its size, $[K]_\epsilon$, the *effective size* of the data-set. Observe that $|[V]_\epsilon| = O([K]_\epsilon), [E]_\epsilon = O([K]_\epsilon)$. Of course, $[K]_0 = K$, but for $\epsilon > 0$, usually $[K]_\epsilon \ll K$, so working with $[G]_\epsilon$ considerably reduces complexity. We omit the ϵ subscript in the sequel.

Scoring by the Homology Information

In this section we show how to use homology information in order to obtain a prior distribution on the space of candidate target sequences. Assume that the unknown target sequence $\mathcal{T} = \langle t_1 \cdots t_l \rangle$ has a known, homologous reference $\mathcal{H} = \langle h_1 \cdots h_l \rangle$, without insertions or deletions (*indels*). This is the case, for instance, when the target $\mathcal{T}$ is a specimen from a population whose reference wild type $\mathcal{H}$ has already been sequenced, and one expects that SNPs will be the only cause of difference between $\mathcal{H}$ and $\mathcal{T}$ (statistically, SNPs are much more prevalent than indels [Wang et al., 1998]). We assume a set of $M \times M$ position specific substitution matrices $\mathcal{M}^{(1)}, \ldots, \mathcal{M}^{(l)}$ are known, where for each position j along the sequence:

$$\mathcal{M}^{(j)}[i, i'] = Prob(t_j = i | h_j = i') \tag{3.11}$$

This is a very general setting. Standard literature discussing nucleotide substitution matrices [Jukes and Cantor, 1969, Kimura, 1980a] assumes all substitution matrices to be the same, i.e., $\mathcal{M}^{(j)} = \mathcal{M}$ for all j. More recent studies support difference between sites for DNA [Yang, 1993b] and protein [Eddy, 1996] sequences.

The setting just presented implies a distribution on the space of possible target sequences. This *prior distribution for ungapped homology*, D^u, is explicitly given for each candidate target sequence $\hat{\mathcal{T}}$ by:

$$D^u(\hat{\mathcal{T}}) = Prob(\hat{\mathcal{T}}|\mathcal{H}) = \prod_{j=1}^{l} \mathcal{M}^{(j)}[t_j, h_j] \tag{3.12}$$

One may recursively compute:

$$D^u(\langle t_1 \cdots t_j \rangle) = D^u(\langle t_1 \cdots t_{j-1} \rangle) \cdot \mathcal{M}^{(j)}[t_j, h_j] \tag{3.13}$$

We denote $\mathcal{L}^{(j)}[x, y] \equiv \log \mathcal{M}^{(j)}[x, y]$.

3.3.3 Spectrum Alignment

In this section we show how to combine our two sources of information on the target sequence, i.e., the result, PS , of the hybridization experiment, and the reference sequence $\mathcal{H}$. We formalize a Bayesian score, which is a composition of the scores discussed in the previous sections, and present a fast dynamic programming algorithm to compute this score.

The probability of a candidate solution sequence $\hat{\mathcal{T}}$, given the information we have is:

$$Prob(\hat{\mathcal{T}}|\mathcal{H}, \text{PS}) = \frac{Prob(\mathcal{H}) \cdot Prob(\hat{\mathcal{T}}|\mathcal{H}) \cdot Prob(\text{PS}|\mathcal{H}, \hat{\mathcal{T}})}{Prob(\mathcal{H}, \text{PS})} \tag{3.14}$$

Given $\hat{\mathcal{T}}$, the hybridization signal is independent of $\mathcal{H}$:

$$Prob(\text{PS}|\mathcal{H}, \hat{\mathcal{T}}) = Prob(\text{PS}|\hat{\mathcal{T}})$$

Thus, omitting the constant $\frac{Prob(\mathcal{H})}{Prob(\mathcal{H}, \text{PS})}$ we can write:

$$Prob(\hat{\mathcal{T}}|\mathcal{H}, \text{PS}) \cong D^u(\hat{\mathcal{T}}) \cdot L^e(\hat{\mathcal{T}}) \tag{3.15}$$

We shall use the approximated likelihood, $\widetilde{L^e}(\hat{\mathcal{T}})$, and after taking logarithms we obtain the following *ungapped score* of a candidate target:

$$Score^u(\hat{\mathcal{T}}) = \log \widetilde{L^e}(\hat{\mathcal{T}}) + \log D^u(\hat{\mathcal{T}}) \tag{3.16}$$

We can compute the highest scoring target sequence by dynamic programming. For each vertex $\vec{y} = \langle y_1 \cdots y_{k-1} \rangle \in \Sigma^{k-1}$, and integer $j = k-1, k, k+1, \ldots, l$, let $S^u[\vec{y}, j]$ be the maximum score of a j-long sequence ending with $\vec{y}$ aligned to $\langle h_1 \cdots h_j \rangle$. Initialize, for each $\vec{y}$:

$$S^u[\vec{y}, k-1] = \sum_{j=1}^{k-1} \mathcal{L}^{(j)}[y_j, h_j] \tag{3.17}$$

Loop over $j = k, \ldots, l$, and for each vertex $\vec{y} = \langle y_1 \cdots y_{k-1} \rangle$ recursively update:

$$S^u[\vec{y}, j] = L^{(j)}[y_{k-1}, h_j] + \max_{e=(\vec{z},\vec{y}) \in E} \{S^u[\vec{z}, j-1] + w(e)\} \tag{3.18}$$

Finally, return:

$$MAXScore^u = \max_{\vec{y} \in V} S^u[\vec{y}, l] \tag{3.19}$$

As in the Smith-Waterman algorithm [Smith and Waterman, 1981], a sequence $\mathcal{T}^*$ attaining the optimal score can be reconstructed by standard means from the matrix S^u, saving trace-back pointers to follow the optimally scoring path. The time complexity is $O(lK)$, since the maximization in (3.18) is over a set of constant size 4. Note that although the complexity is exponential in k, it is constant for a given microarray (currently feasible values are $k = 8, 9$). Working with $[G]$, with high probability the correct solution will not be missed, but the time complexity will drop sharply to $O(l[K])$.

A crucial issue for the practicality of this algorithm is its space requirement. Computing the optimal score alone requires space which is linear in the (effective) size of the hybridization experimental data, that is $O([K])$ space. However, in order to reconstruct the optimal path, we need to record trace back pointers for the full $l \times [K]$ matrix. By following the paradigm of Hirschberg 1975 for linear-space pairwise alignment, we provide an algorithm which requires only linear space.

The reduced space complexity is traded for time complexity, which increases by an $O(\log l)$ factor.

For each position $j = l, l-1, \ldots, k, k-1$, we can decompose the score of the entire sequence. We present the total score as a sum of two expressions: the contribution of its $(j - k + 1)$-prefix, which equals the score of this prefix computed by S^u, plus the contribution of the corresponding suffix. Formally, for each vertex $\vec{y} = \langle y_1 \cdots y_{k-1} \rangle \in [V]$ let $R^u[\vec{y}, j]$ be the maximum contribution to the score of a $(l-j+k-1)$-long sequence beginning with $\vec{y}$ aligned to $\langle h_{j-k+2} \cdots h_l \rangle$. Initialize, for each $\vec{y}$:

$$R^u[\vec{y}, l] = 0 \tag{3.20}$$

Loop over $j = l-1, l-2, \ldots, k-1$, and for each vertex $\vec{y} = \langle y_1 \cdots y_{k-1} \rangle$ recursively update:

$$\begin{aligned} R^u[\vec{y}, j] = \max_{e=(\vec{y},\vec{z})\in E} \{ & R^u[\vec{z}, j+1] \\ & + w(e) \\ & + \mathcal{L}^{(j+1)}[z_{k-1}, h_{j+1}] \} \end{aligned} \tag{3.21}$$

Observe that, for all $k - 1 \leq j \leq l$

$$MAXScore^u = \max_{\vec{y}\in V} \{ S^u[\vec{y}, j] + R^u[\vec{y}, j] \} \tag{3.22}$$

One can use Equation 3.22 to decompose the problem into two similar problems, of half its size. Recursively solving these sub-problems gives a a divide-and-conquer approach for finding the optimal sequence. The linear space algorithm is therefore as follows:

1. If l is smaller than some constant C:
   solve the problem directly, according to
   the dynamic program of Equation 3.13.
   Otherwise:
2. Set $m = \frac{l+k-1}{2}$.
3. For each $j = k-1, k, \ldots, m$:
   Compute $S^u[\vec{y}, j]$ for all $\vec{y}$, re-using space.
4. For each $j = l, l-1, \ldots, m$:
   Compute $R^u[\vec{y}, j]$ for all $\vec{y}$, re-using space.
5. Find $\vec{y}_m = arg\max_{\vec{y} \in V}\{S^u[\vec{y}, m] + R^u[\vec{y}, m]\}$,
   thereby computing: $MAXScore^u$, by (3.22).
6. Recursively compute:
   (a) The optimal sequence aligned to $\langle h_1 \cdots h_m \rangle$ ending with $\vec{y}_m$.
   (b) The optimal sequence aligned to $\langle h_m \cdots h_l \rangle$ beginning with $\vec{y}_m$.

Observe, that for each $\vec{y}, j$, the values of $S^u[\vec{y}, j]$ and $R^u[\vec{y}, j]$ are computed a total of $\log l$ times. Thus the algorithm takes $O([K] l \log l)$ time and $O([K])$ space using the effective spectrum.

3.3.4 Handling Gaps

Deletions

In this section we assume that the unknown target sequence $\mathcal{T} = \langle t_1 \cdots t_{l'} \rangle$ is obtained from its reference $\mathcal{H} = \langle h_1 \cdots h_l \rangle$, by substitutions and deletions only, and base insertions do not occur. Insertions in the target are, of course, equivalent to deletions in the reference and vice versa, but since the reference is known we consider all sequence editing operations (mutations) to have occurred in the target sequence.

Although a model of homology without insertions is unrealistic, we include

discussion of this case due to the simplicity and efficiency in which deletions fit into our model. The general case, allowing insertions, will be described in the next subsection.

We begin with a few notations. Denote the probability of initiating a gap right before h_j (aligning h_j to *space*) is 2^{α_j}. Similarly, β_j is the logarithm of the probability for gap extension at h_j. Also define $\widehat{\beta_j} = \log(1-2^{\beta_j})$, $\widehat{\alpha_j} = \log(1-2^{\alpha_j})$. To overcome boundary problems at the ends of the sequence, we extend the alphabet by including left and right space characters: $\overline{\Sigma} = \Sigma \cup \{\triangleright, \triangleleft\}$. We augment the reference sequence by the string $\triangleright^k$ on its left and $\triangleleft^k$ on the right. We extend the substitution matrix by using probabilities that force alignment of each of $\triangleright$, $\triangleleft$ to itself. Formally, we define:

$$\begin{aligned} \overline{\Sigma^{k-1}} = \Sigma^{k-1} \quad &\cup \quad \{\vec{x}\vec{z} | \vec{x} = \triangleright^j, \vec{z} \in \Sigma^{k-1-j}\} \\ &\cup \quad \{\vec{z}\vec{x} | \vec{z} \in \Sigma^j, \vec{x} = \triangleleft^{k-1-j}\} \end{aligned} \tag{3.23}$$

We arbitrarily set $w(\vec{y})$ to 0 for each $\vec{y} \in \overline{\Sigma^{k-1}} \setminus \Sigma^{k-1}$. Thus, the weighted de-Bruijn graph is naturally extended over $\overline{\Sigma^{k-1}}$, and so is $[G] = ([V], [E])$, its effective subgraph. Hereafter, we use the notation $[G]$ for the extended graph.

For each $\vec{y} = \langle y_1 \cdots y_{k-1} \rangle \in [V]$, $j = k-1, k, k+1, \ldots, l$, let $S^d[\vec{x}, j]$ be the maximum score of aligning a sequence ending with $\vec{y}$ to $\langle h_1 \cdots h_j \rangle$ where h_j is aligned to a gap (and y_{k-1} is aligned to some $h_i, i < j$). Further let $T^d[\vec{x}, j]$ be the maximum score of aligning a sequence ending with $\langle y_1 \cdots y_{k-1} \rangle$, to $\langle h_1 \cdots h_j \rangle$ where h_j is aligned to y_{k-1}. Initialize, for each $\vec{y}$:

$$S^d[\vec{y}, k-1] = -\infty; \tag{3.24}$$

$$T^d[\vec{y}, k-1] = \begin{cases} 0 & \vec{y} = \triangleright^{k-1} \\ -\infty & \text{otherwise} \end{cases} \tag{3.25}$$

Loop over $j = k, \ldots, l$, and for each $\vec{y} = \langle y_1 \cdots y_{k-1} \rangle \in [V]$, recursively update:

$$S^d[\vec{y}, j] = \max\{T^d[\vec{y}, j-1] + \alpha_j, S^d[\vec{y}, j-1] + \beta_j\} \tag{3.26}$$

$$\begin{aligned} T^d[\vec{y}, j] = &\mathcal{L}^{(j)}[y_{k-1}, h_j] \\ &+ \max_{e=(\vec{z},\vec{y}) \in E} \left\{ w(e) + \max \begin{Bmatrix} T^d[\vec{z}, j-1] + \widehat{\alpha_j} \\ S^d[\vec{z}, j-1] + \widehat{\beta_j} \end{Bmatrix} \right\} \end{aligned} \tag{3.27}$$

Finally, return:

$$MAXScore^d = T^d[\triangleleft^{k-1}, l] \tag{3.28}$$

The complexity of this algorithm is still $O(l[K])$ and a linear space variant can be obtained, similarly to the one presented previously.

Insertions and Deletions

In this subsection, and in the following one, we present two distinct algorithms for sequence reconstruction assuming both insertions and deletions with respect to the reference sequence. The algorithm we present in this section is a relatively simple extension of the dynamic programs we have presented thus far. The only change is that a different weighted graph is used, forcing higher complexity. In the next section we will also present a faster algorithm.

The algorithm presented in the previous section computes a maximum-likelihood target sequence, when the log-likelihood is a sum of edge weights along the weighted de-Bruijn graph (G, w), and log-probabilities derived from homology. In this section, we compute a different weighted graph, (G', w'). Substituting (G, w) for (G', w'), the algorithm described in the previous section solves the problem variant with both deletions and insertions.

We introduce some more notation. Denote by $\mathcal{T}_j$ the target prefix whose last nucleotide is aligned to h_j in the reference sequence. Further denote by a_j (respectively, b_j) the log-probability of initiating (extending) an insertion in the target after $\mathcal{T}_j$, and define $\widehat{a_j} = 1 - a_j$,$\widehat{b_j} = 1 - b_j$.

Consider the weighted graph (G, w). Define the $K \times K$ matrix W as follows:

$$W[\vec{x}, \vec{y}] = \begin{cases} 2^{w(\vec{y})} & \text{The } (k-1)\text{-suffix of } \vec{x} \\ & \text{is the } (k-1)\text{-prefix of } \vec{y}. \\ 0 & \text{Otherwise.} \end{cases} \tag{3.29}$$

$W^i[\vec{x}, \vec{y}]$ is thus the probability of moving from $\vec{x}$ to $\vec{y}$ along i edges. The probability of an insertion of length i after $\mathcal{T}_j$ is $a_j b_j^i \widehat{b_j}$. Suppose that the prefix T_j

ends with $\vec{x}$. Then $a_j b_j^{i-1} \widehat{b_j} W^i[\vec{x}, \vec{y}]$ is the probability of $\mathcal{T}_{j+1}$ ending with $\vec{y}$ and being i nucleotides longer than $\mathcal{T}_j$. We are now ready to compute the matrix W', that governs the stochastic progression from $\mathcal{T}_j$ to $\mathcal{T}_{j+1}$:

$$\begin{aligned} W' &= \widehat{a_j} W + a_j b_j W^2 \widehat{b_j} + a_j b_j^2 W^3 \widehat{b_j} \cdots && (3.30) \\ &= \widehat{a_j} W + a_j b_j \widehat{b_j} W^2 \sum_{i \geq 2} b_j^{i-2} W^{i-2} && (3.31) \\ &= \widehat{a_j} W + a_j b_j \widehat{b_j} W^2 (I - b_j W)^{-1} && (3.32) \end{aligned}$$

We define a new weighted graph (G', w'). The vertex set of G is also the vertex set of G'. The edge set E' of G' is the set of all pairs $(\vec{x}, \vec{y})$ with $W'[\vec{x}, \vec{y}] > 0$. Each such edge $e = (\vec{x}, \vec{y})$ is associated with a weight $w'(e) = \log W'[\vec{x}, \vec{y}]$. One can apply the algorithm for deletions only, but use (G', w') instead of (G, w). This solves the problem for insertions and deletions.

Note that in contrast to G, degrees in G' are not bounded by 4. Therefore, computing each dynamic program cell has complexity $O(K)$ in the worst case, with the total complexity of the algorithm being $O(l|E'|)$. Again, considering only the effective size of the graph allows more efficient computation, taking $O(l|[E']|)$. Unfortunately, this is may be $\Omega(l[K]^2)$ in the worst case.

A Faster Algorithm

A general probabilistic model of homology can facilitate a more efficient algorithm that allows both insertions and deletions. Hidden Markov Models (HMMs) were proved useful for profiling protein families [Krogh et al., 1994]. We use a similar formulation to describe homology between nucleotide sequences. The reference, along with the statistical assumptions, actually creates a profile.

Below, we briefly sketch the model. The reader is referred to Durbin et al. [1998b, chapter 5] for details. The model assumes a set Q of Markov chain states with a predefined set of allowed transitions between them. For each level (position along the profile) $j = 1, \ldots, l_Q$, Q includes three states: M_j (match), I_j (insert), and D_j (delete). M_j and D_j can be reached from the three $(j-1)$-th level states. I_j can be reached from the three j-th level states (including a self-loop). Transition

probabilities are as described in previous sections, e.g., $a_j = Prob(M_j \mapsto I_j)$. Additionally, each insert or match state, q, induces a vector of emission probabilities $\mathcal{M}^q$, where $\mathcal{M}^q[i]$ is the probability that the target nucleotide is i. We denote $\mathcal{L}^q[i] \equiv 0$ for $q = D_j$, $\mathcal{L}^q[i] \equiv \log \mathcal{M}^q[i]$ otherwise. We write $lpb(\mathcal{X}) \equiv \log Prob(\mathcal{X})$ for short.

The dynamic programming scheme we use for ungapped homology cannot be directly modified to handle the HMM because of the insertion loops. To wit, we generalize this scheme by an additional dimension, which denotes the position along the target sequence.

Define a three dimensional array S, where for each $q \in Q$, $\vec{y} = \langle y_1 \cdots y_{k-1} \rangle \in [V]$, $r = k, \ldots, L$ let $S[q, \vec{y}, r]$ be the maximum score of an r-long sequence ending with $\langle y_1 \cdots y_{k-1} \rangle$, whose alignment to the profile ends in q. Initialize:

$$S[q_{start}, \triangleright^{k-1}, k-1] = 0 \tag{3.33}$$

$$S[q, \vec{y}, k-1] = -\infty \quad \text{for other values of } \vec{y}, q \tag{3.34}$$

Loop over $r = k, \ldots l$, and for each $\vec{y} = \langle y_1 \cdots y_{k-1} \rangle \in [V]$, $r \leq l_Q$, recursively update:

$$\begin{aligned} S[q, \vec{y}, r] = \mathcal{L}^q[y_{k-1}] \\ + \max_{\substack{e=(\vec{z},\vec{y}) \in E \\ q' | q' \mapsto q}} \{ & S[q', \vec{z}, r-1] \\ & + lpb(q' \mapsto q) \\ & + w(e) \} \end{aligned} \tag{3.35}$$

Finally, return:

$$MAXScore = \max_l \{ S[q_{end}, \triangleleft^{k-1}, l] \} \tag{3.36}$$

Let L be some known bound on the size of the target sequence. Naive implementation of this algorithm requires $O(l_Q \cdot [K] \cdot L)$ time and space. By the means presented earlier, the complexity of this algorithm can be reduced to $O(l_Q \cdot [K] \cdot L \log L)$ time and $O(l_Q \cdot [K])$ memory. Furthermore, one can consider the dynamic program as filling a $l_Q \times L$ matrix, with a $[K]$-long vector in each matrix

cell. Since all values far from the main diagonal of this matrix should be negligible, we can settle for computing only values with distance smaller than R to the main diagonal, reducing the complexity to $O(R(l_Q + L) \cdot [K] \cdot \log L)$ time and $O(R(l_Q + L) \cdot [K])$ space.

3.3.5 Computational Results

The algorithm was implemented and tested on simulated data. Each simulation scenario we benchmarked specified the total sequence length, mutation probabilities, hybridization error (false positive/negetive rate), and probe length, k. As a reference, we used prefixes of real coding sequences: For each simulation scenario we collected statistics from 100 such sequences, arbitrarily taken from GenBank's collection of human transcripts. Sequences with long repeats were discarded. For testing the reconstruction of long targets we pooled (concatenated) several transcripts.

Each simulation run was performed as follows:

1. Introduce mutations in the reference sequence R and obtain the target sequence T.
2. Form the probabilistic spectrum of T.
3. Reconstruct the target given the reference R and the spectrum.
4. Compare the reconstructed sequence to T.

For simplicity, substitutions by different nucleotides were equiprobable, and mutation rates remained fixed along the sequence. In some of the simulation scenarios we further restricted mutations to nucleotide substitutions only.

We used a single parameter p for the hybridization error, setting $P_i(\vec{x}) = 1 - p$ if $\mathcal{T}(\vec{x}) = i$. All probabilistic parameters were constant, i.e., position/k-mer independent.

We quantified the performance by the following figures of merit:

1. *Perfect reconstruction rate* - The fraction of runs for which the target sequence was perfectly reconstructed.

2. *Average sequencing error* - The fraction of base-calling errors made by our algorithm.

We focused our simulations around a *basic scenario*. This scenario assumed hybridizaion to an all-8-mer array, with $p = 0.05$ false positive and false negative rate. The differences between the target and reference sequences were assumed to be substitutions only, with SNP rate being 1:200bp. To examine effects of different parameters on performance, we performed several series of simulations. In each such series, we changed one or more parameter values while keeping the rest at their values as in the basic scenario.

Our simulations check perfect data with 6-mer probes (Figure 3.7), 8-mer data with varying hybridization error (Figure 3.8) varying mutation rate (Figure 3.9) varying probe length (Figures 3.10 and 3.12), varying target length (Figure 3.11) and indel data with varying hybridization error (Figure 3.13). A discussion of the results will be given in Section 3.3.6.

The algorithm was implemented in $C++$ and executed on Linux and SGI machines. Running times, on a Pentium 3, 600MHz machine, were roughly $0.12l \log l$ seconds for an l-long reference sequence on a full 8-mer array (ranging from roughly 7 minutes for a 500bp-long sequence to 2.5 hours for 6Kb). Only the main memory was used, with the application consuming at most 40Mb. As a first implementation, we did not reduce the graph to its effective size. This would of course reduce both space and time dramatically, at the expense of possibly missing the truely maximal scoring sequence.

3.3.6 Discussion

We have developed a new method that combines spectrum data and homology information in order to algorithmically reconstruct a target sequence. The method is general enough to allow for insertions and deletions, hybridization errors, and a profile or a HMM instead of a single reference sequence. As the spectrum data

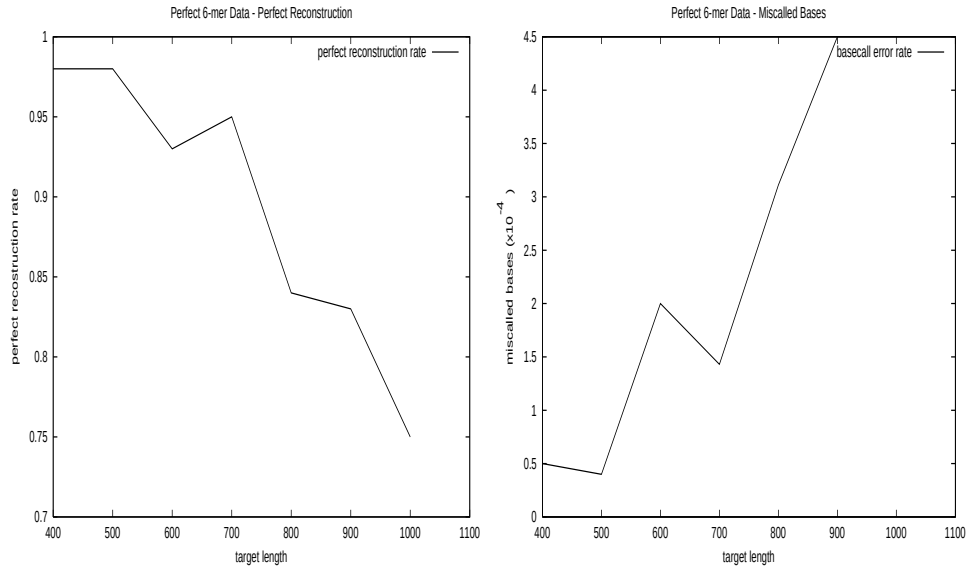


Figure 3.7: Impact of target length on perfect reconstruction rate (left) and average sequencing error (right) on errorless 6-mer data. Other simulation parameters are as in the basic scenario.

needed originates from standard chips that can easily be mass produced, the cost of generating the hybridization data can potentially be reduced to a very small fraction in comparison to current special-purpose chips.

The algorithm was implemented and tested, and simulation results are reported here. With perfect hybridization data, even 6-mer arrays suffice to achieve read length of 700bp, which is competitive to gel-based sequencing (see Figure 3.7). Such arrays are small enough for cost-effective manufacturing, and the low spot density helps achieving high quality hybridization.

With 8-mer arrays, assuming practical levels of hybridization error, once can resequence 2.5-3kb with success rate of about 90% and basecall error rate less than 1:10,000 (see Figure 3.8). These results are quite robust to changes in hybridization error rate. Observe, that high false negative rate is more problematic to handle than high false positive rate. This is counter-intuitive: The number of probes that do not occur along the target is larger by an order of magnitude than the number of probes that do. Hence, each increase in false positive rate implies many more false

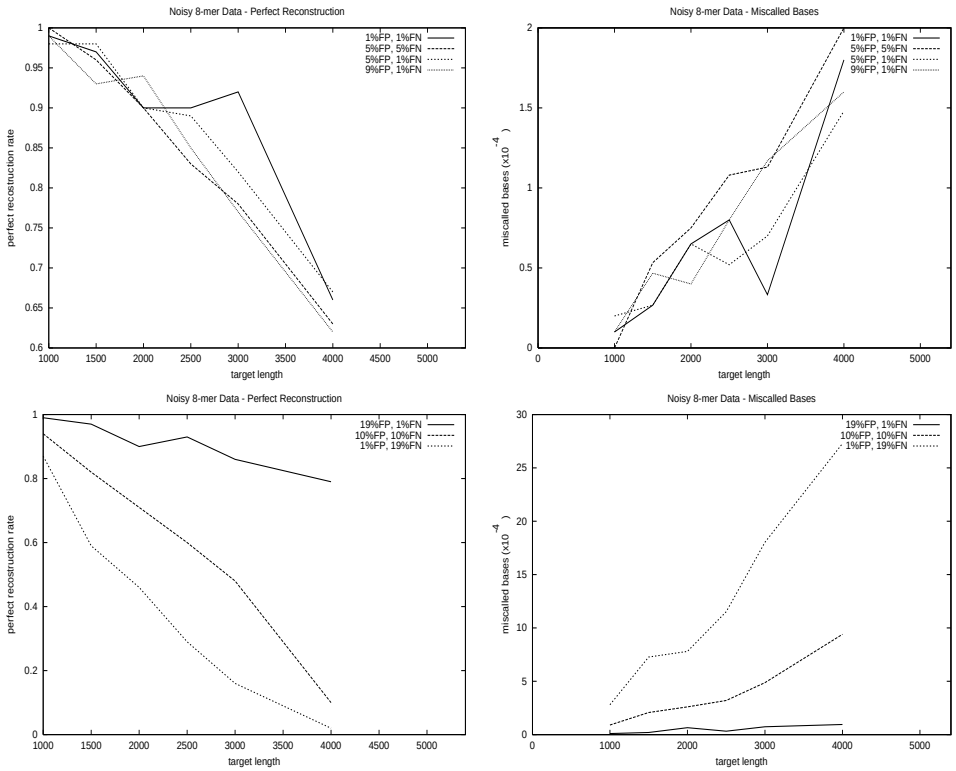


Figure 3.8: Impact of different false positive (FP) and false negative (FN) rates, on perfect reconstruction rate (left charts) and average sequencing error (right charts), for targets of different lengths.

signals than the same increase in false negative rate. On a closer look, we suggest that most of the false positive signals concern edges of the de-Bruin graph that are far from hi-scoring vertices, thus they do not damage performance as much as false negatives do. This preference for the type of error is of practical value for the implementation of this method, as signal digitization methods usually offer false positive/negative tradeoffs.

Higher mutation rates still comfortably allow sequencing 1kb, although performance for 2.5kb severely deteriorates (see Figure 3.9). This enables, e.g., sequencing a chimpanzee gene using our method, with the available human homologous gene as a reference sequence. Observe, that the plots show a non-

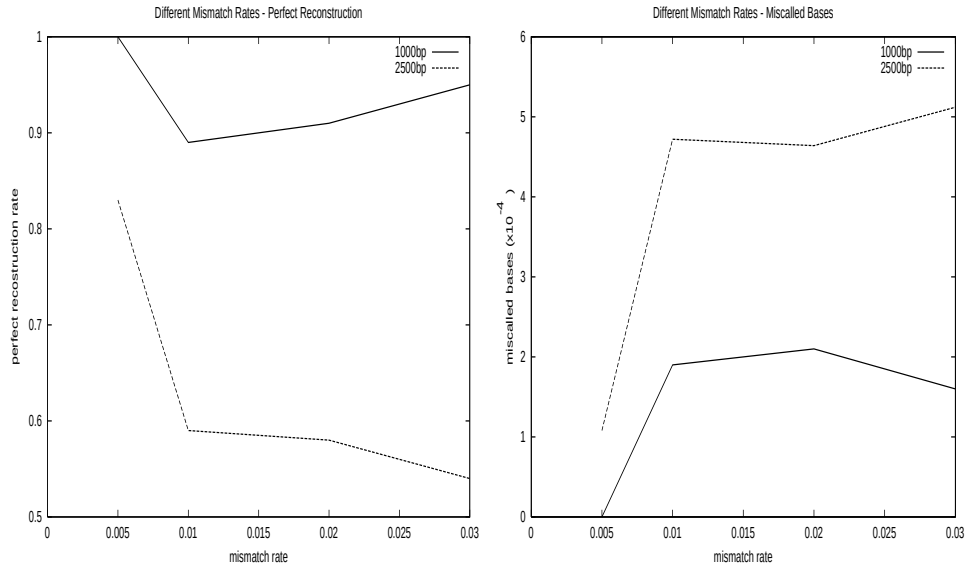


Figure 3.9: Impact of mutation rate, for different target lengths, on perfect reconstruction rate (left) and average sequencing error (right). Other simulation parameters are as in the basic scenario.

monotonous trend. This is not a stochastic error, but rather an artifact of our simulation and probabilistic model, as we briefly explain. Consider the score evaluation of two sequences, one of which is identical to the reference sequence, while the other has a single mismatch. To compare these scores one needs to examine the weight loss due to the indroduction of that mismatch, versus a possible weight gain due to probes that overlap the mismatch position and support the mismatched sequence over the wild-type. However, our naive model of hybridization error assumes constant error probabilities, and thus constant weight contribution to every probe. Therefore, that comparison boils down to whether the discrete number of mismatch-supporting probes is greater than a certain integer threshold. This threshold is a monotone, but piecewise constant function of the mismatch probability. Thus, while the probability of mutation increases gradually, the algorithm will compensate for that increase by changing the threshold only at discrete values. The non-monotonous plots we see are due to the resulting non-continuous behavior. (In applications to real data, with mismatch probabilities that are not constant but vary

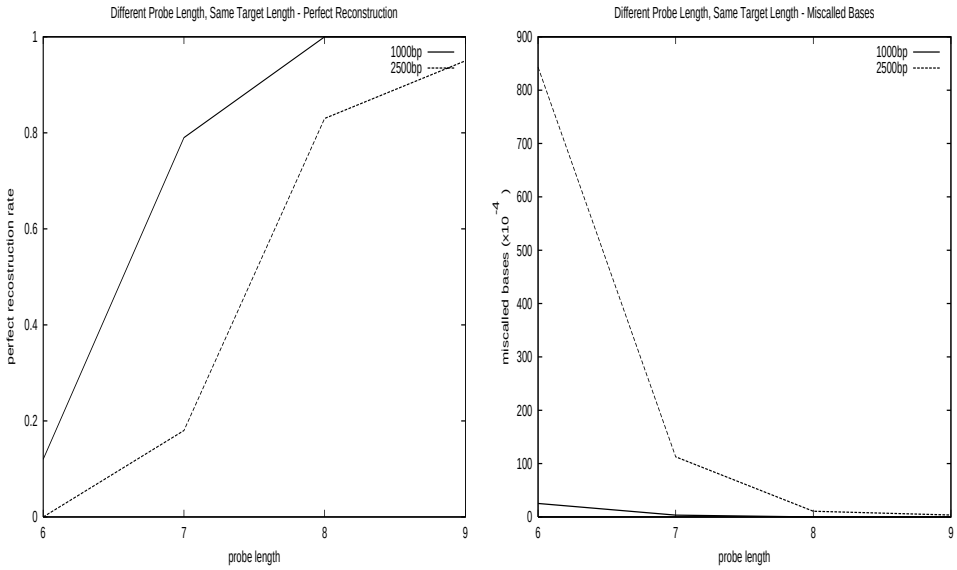


Figure 3.10: Impact of probe length for different target lengths on perfect reconstruction rate (left) and average sequencing error (right). Other simulation parameters are as in the basic scenario.

depending on probe contents and real-valued hybridization signal, we expect this artifact to be negligible.

When comparing arrays with probes of increasing lengths (see Figures 3.10, 3.11, and 3.12), the expected increase in performance is evident. Observe, that while incrementing the probe length by one quadruples the microarray size, in practice it only doubles the length of reconstructible target. This factor of two corresponds to theoretical bounds on feasible target length in classical SBH [Pevzner and Lipshutz, 1994].

Finally, Figure 3.13 demonstrates performance in the presence of insertions and deletions. Even in this case we are able to achieve good performance at 2kb, which is four times the read length in a standard gel-based sequencing machine.

This study constitutes a proof of concept, and there is much room for further work. Modifications and improvements are possible both in the theoretical analysis, and in the implementation details, especially in view of some of the practical

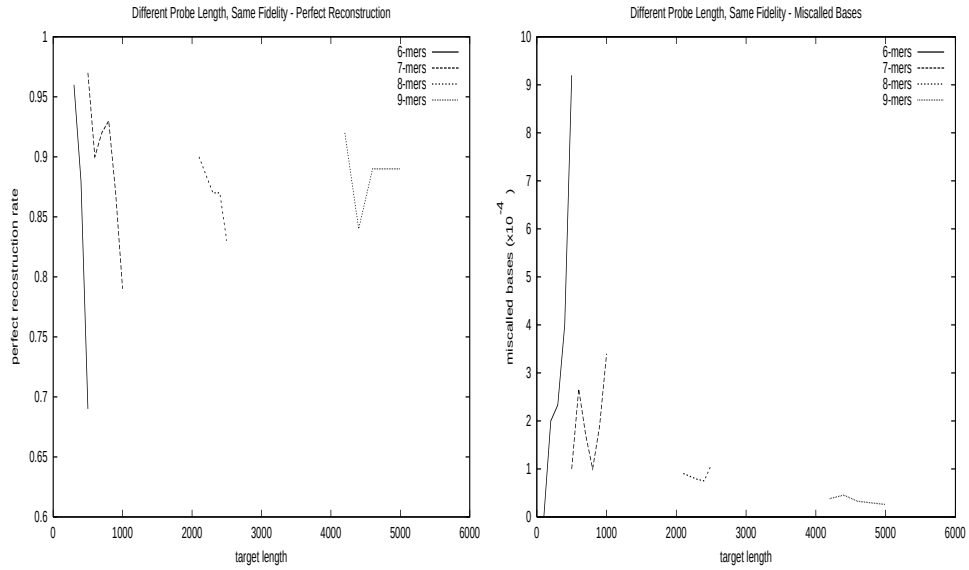


Figure 3.11: Impact of target length for different probe lengths on perfect reconstruction rate (left) and average sequencing error (right). Other simulation parameters are as in the basic scenario.

considerations. On the algorithmic side, there are several possible promising refinements and future developments:

- It is clear that one can do better by using the exact likelihood score, instead of the approximate score that we give. Refined examination of the errors made by our application suggests that post-processing the output sequence, locally modifying it in search for the optimum point of the exact score, may correct many of these errors.

- For simplicity, we imposed on the problem the assumption of independence of hybridization signals from different oligos, leading to Equation 3.7. This obviously oversimplifies the problem. For instance, oligonucleotides corresponding to k-substrings which overlap along the target sequence would have correlated hybridization results. Thus, replacing the independence assumption by a more realistic one will render the results more practical.

- Another promising direction is sequence profiles, which have been exten-

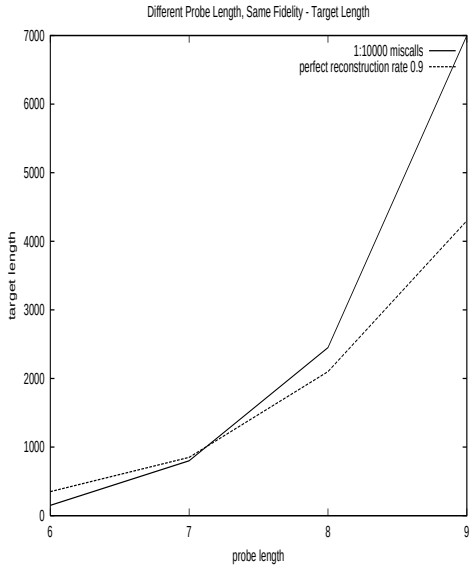


Figure 3.12: Impact of probe length on performance. The performance is measured by the reconstructible target length given specific fidelity requirements. Solid line: at most 1:10000 miscalled bases. Dotted line: perfect reconstruction rate of at least 0.9. Other simulation parameters are as in the basic scenario.

sively used for protein sequences [Krogh et al., 1994]. Our method can be applied, as is, also for sequencing a target whose reference is not a single related sequence, but rather an HMM profile (for nucleotides, instead of amino acids).

Practical aspects suggest several extensions to the basic procedure, in order to render it applicable to real-life data:

- Though the algorithm is polynomial for fixed k, its dependence on k is exponential (4^k), which makes it rather slow in practice. Using the effective size of the graph instead, as suggested, would economize on time and space

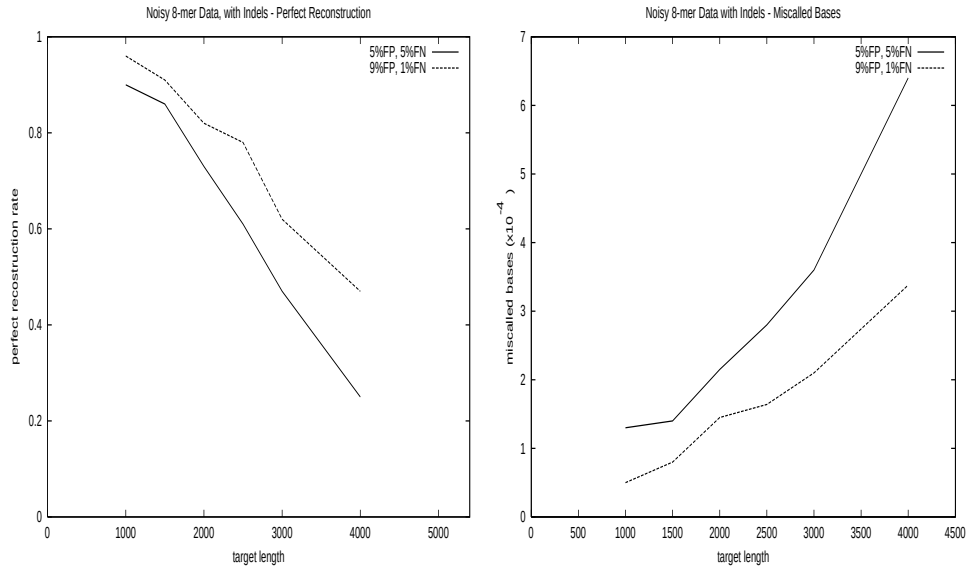


Figure 3.13: Impact of indels: The plots show the effect of target length, for different false positive and false negative rates, on perfect reconstruction rate (left) and average sequencing error (right), when indels are introduced. Deletion and insertion probabilities were set to $\frac{1}{3}$ and $\frac{1}{10}$, respectively, of the substitution probability. Other simulation parameters are as in the basic scenario.

considerably. This may be necessary for incrementing k, as microarray technology progresses.

- Since the target is usually a PCR product, its primer sequences are known. This information can and should be used when initializing and terminating our dynamic program.

- The target is usually diploid DNA, and might be a heterozygote. The algorithm should thus be extended to handle a hybridization signal from two sequences.

- When searching for mutations in many candidate genes, or exons, one could use pooling of hybridization targets. The algorithm can be easily modified to re-sequence several short targets in one experiment, instead of a single long one.

- Despite their mathematical elegance, all-k-mer chips have practical problems. Oligonucleotides with different GC-content impose conflicting constraints on the experiment temperature, and some are unusable due to self loops. Our algorithm can be modified to work with a collection of equi-temperature oligos, instead of equal-length ones, and to compensate for defunct k-mers.

Chapter 4

Reconstructing a Physical Map from Optical Data

This chapter concerns *Optical Mapping*: a revolutionary method for obtaining maps of enzymatic cleavage sites along large DNA molecules. The method is based on immobilization of many copies of the target molecule in elongated form. Enzymes that perform site-specific cleavage (*restriction*) are then applied, and each cleaved, fluorescent molecule is then photographed, to detect cleavage sites. Unfortunately, the orientation of each photographed molecule is unknown. Furthermore, the reported list of cleavage sites suffers from inaccurate locations, false positives and false negatives. The computational problem is to reconstruct the true map of cleavage sites from this noisy data. We define a maximum-likelihood formalization of this problem, and devise an appropriate algorithm. We implement our methods and demonstrate performance on a blind test using real life data. This study has been published in [Karp et al., 1999, 2000].

Optical mapping is a novel technique for generating the restriction map of a DNA molecule by observing many single, partially digested copies of it, using fluorescence microscopy. The real-life problem is complicated by numerous factors: false positive and false negative cut observations, inaccurate location measurements, unknown orientations and faulty molecules. We present an algorithm for solving the real-life problem. The algorithm combines continuous optimization

and combinatorial algorithms, applied to a non-uniform discretization of the data. We present encouraging results on real experimental data, and on simulated data.

4.1 Introduction

Even in the era of whole genome methods, the mapping of restriction sites still plays an important role in genomic analysis and motivates further development of mapping procedures [Cai et al., 1998, Lin et al., 1999]. Optical mapping is a novel technique for obtaining restriction maps [Samad et al., 1995a, Cai, 1996, Schwartz et al., 1993, Meng et al., 1995, Samad et al., 1995b, Jing et al., 1998, Cai et al., 1998, Lin et al., 1999]. In an optical mapping experiment, many copies of the target DNA molecule are elongated and attached to a glass surface. Restriction endonuclease enzymes are applied to the molecules, partially digesting them. At the cleaved cut sites the two cleaved ends of the molecule coil away from each other due to the elasticity of DNA. The molecules are stained and photographed. Molecules appear in the image as lines separated by gaps at cut locations. For each molecule the locations of these cuts along the DNA are recorded. The goal is to deduce from this data the correct locations of all restriction sites.

Optical mapping has several advantages over traditional mapping methods: It can provide high resolution maps which determine physical landmarks in the DNA; it is insensitive to repetitions in the sequence, and can be automated to a considerable extent (see also Schwartz and Samad [1997]). Moreover, since the data preserves the linear order of the sites, it is more informative than the data from traditional gel-based restriction mapping methods. Currently, BAC sized clones are routinely mapped, and it is possible to map molecules of up to several hundred Kb. A variant of this technique in which cut sites in sub-fragments of the target are recorded was successfully used recently to obtain a 3.2Mb map of Deinococcus radiodurans [Lin et al., 1999].

In this chapter we address the basic optical mapping technique, where cuts are recorded with respect to the complete target molecule. Even in that case, the deduction of the true restriction sites is not straightforward, due to the following factors:

- An observed cut may not correspond to a restriction site. We call such a cut *false*.
- No cut may be observed at a true restriction site, due to partial digestion.
- The orientation of each molecule is unknown, *i.e.*, observed cut locations are known only up to complete reversal of the molecule.
- There is a sizing error when measuring observed cut locations.
- There are molecules on which the observed cuts do not correspond to real sites in any orientation, and instead produce "random" results. We call these molecules *faulty*, and the other molecules *proper*.

Optical mapping uses a number of copies of the target molecule ranging between several dozen and several hundred. The digestion rate, false cut rate, and the sizing error vary considerably, even when comparing restriction sites in the same experiment [Cai, 1996].

Various combinatorial formulations of the optical mapping problem have been proven NP-complete [Anantharaman et al., 1997, Parida and Mishra, 1998, Muthukrishnan and Parida, 1997] and even hard to approximate [Parida, 1999]. Bayesian approaches using global optimization have been applied in Anantharaman et al. [1997] and Lee et al. [1998], where a probabilistic model of the experiment is set up, and global optimization techniques are used to find the most probable parameters of this model, which comprise the desired solution. The implementation reported in Anantharaman et al. [1997] is successfully used in practice for analysis of real laboratory data. Other studies [Karp and Shamir, 2000, Daněk et al., 1997, Muthukrishnan and Parida, 1997] discretize the input, and devise combinatorial solution methods. The algorithm of Karp and Shamir [2000] has a proven performance guarantee under a simple probabilistic model of the data.

In this chapter we describe a new strategy combining the combinatorial approach with the global optimization approach. We first determine the orientations of the molecules and then determine the restriction sites by applying continuous optimization in the space of certain model parameters, as done in Anantharaman et al. [1997] and Lee et al. [1998]. In order to orient the molecules we adapt a

combinatorial algorithm from Karp and Shamir [2000]. However, instead of using a fixed, uniform discretization of the data as in Karp and Shamir [2000], we use a nonuniform discretization based on identifying certain informative intervals derived from the data itself. The chief innovation in this chapter is the use of continuous and discrete algorithmic methods to enhance each other.

Our method was applied in a blind test to eight examples provided by the lab of David Schwartz at New York University. The results show that we determine the orientation of the molecules correctly, enabling us to identify almost all the restriction sites. Occasional misestimation of the number of restriction sites can probably be alleviated by finer tuning of the algorithm. We also present results on simulated data, for several different choices of simulation parameters.

The chapter is organized as follows: Section 4.2 outlines the general strategy of our algorithm. In Section 4.3 we present our probabilistic model. Sections 4.4, 4.5, and 4.6 describe the three major stages in our algorithm. Section 4.7 gives results on experimental and simulated data.

4.2 General Strategy

4.2.1 Disadvantages of the Discrete Approach

We sketch the *signature method* that was suggested in Karp and Shamir [2000] as a method of determining the locations of restriction sites in a discretized version of the problem. We assume that the data is scaled so that the target molecule and each of its copies extends over the interval $[0, 1]$. This interval is partitioned into n equal sections, each of which may contain a single restriction site (typically, $n = 200$). Sections $k \leq \frac{n}{2}$ and $n + 1 - k$ are called *conjugate*. Thus each conjugate pair contains a section from the left half of the target molecule and a symmetrically placed section from the right half of the molecule. For each conjugate pair, the numbers of molecules with no observed cuts, one observed cut and two observed cuts are determined. On the basis of this information, the conjugate pairs are divided into three types: those likely to contain no restriction sites, those likely to contain two restriction sites, and those likely to contain one restriction site. The conjugate pairs

of the first two types are set aside, and the remaining conjugate pairs are then divided into two classes such that two pairs are in the same class if their restriction sites appear to lie in the same half of the target molecule, and in the opposite classes otherwise. The details will be omitted here, but the idea is that two conjugate pairs should be placed in the same class if, for those copies of the molecule in which each of the two conjugate pairs contains one observed cut, the observed cuts tend to occur in the same half of the molecule. We refer to this process as *resolving* the conjugate pairs. In Karp and Shamir [2000] the signature method was directly used to determine the restriction sites. In this chapter a refinement of the signature method is used to orient the molecules, in preparation for a later stage in which the restriction sites are determined.

The signature method (and other algorithms of Karp and Shamir [2000]) were applied to uniformly discretized real data and failed. The main reason is that the sizing errors (and hence, the errors in measured cut locations) are too large to conform with the uniform discretization. Probabilistic analysis of the error under uniform discretization [Anantharaman and Mishra, 1998] illuminates the limitations of such discretization from a theoretical perspective, and motivates finding a better way to apply combinatorial methods. The fact that the algorithms in Karp and Shamir [2000] disregard faulty molecules adds to the problem by assuming errors occur independently in the data.

4.2.2 Disadvantages of the Global Optimization Approach

In the continuous approach, one formulates a probabilistic model with many parameters (*e.g.*, molecule orientations, site locations, cut intensities, noise, etc.) and attempts to find, by global optimization methods, the most likely parameter values given the data. This approach has been demonstrated to perform well on real data [Anantharaman et al., 1997, Lee et al., 1998]. However, there is room for improvement in some important respects, as detailed below.

The score (likelihood) function takes into account both orientations of each molecule. This is done by averaging two probability functions, one for each orientation, so one of them just adds noise to the computation. Clearly, if all other

factors are unchanged, a score based on correctly oriented data would give better optimization results.

Furthermore, among the parameters to be optimized, and probably the most important ones, are the locations of the restriction sites. These locations are determined by global optimization of a joint likelihood function which associates, with each finite set S of points along the molecule, the likelihood that S is the set of locations of the restriction sites. Usually there are many short intervals in which a restriction site is likely to have occurred, separated by intervals in which the occurrence of a restriction site is unlikely. Thus the joint likelihood function tends to have many local optima, corresponding to selections of points from different combinations of these short intervals. This multi-modality makes the task of finding the global optimum quite difficult for standard techniques. Our methods alleviate this difficulty in two ways. First, since the molecules have been oriented before global optimization is undertaken, the number of short intervals likely to contain a restriction site is reduced, leading to a great reduction in the number of local maxima of the joint likelihood function. Secondly, we initialize the search with a reasonably good estimation of the locations of the restriction sites. If the initialization is good enough, then the global optimum point will lie within a small neighborhood of the initialization point, and within this neighborhood the likelihood function will be unimodal, and therefore easy to maximize.

4.2.3 Our Approach

Our approach uses elements of both the continuous and the discrete approaches, attempting to remedy the shortcomings of each: We reduce the main error source in the discrete procedure for orienting the molecules (*i.e.* the uniform discretization) by a more subtle consideration of the continuous data. The results of the refined orientation procedure eliminate the need to optimize over orientation parameters for individual molecules, and thus help the continuous global search heuristic avoid local optima. We now intuitively sketch our algorithm, deferring formal description to later sections.

We assume a continuous probabilistic model of the data, similar to the model

used in Anantharaman et al. [1997] and Lee et al. [1998]. To orient the molecules we use a variant of the signature algorithm that is less sensitive to sizing errors. This variant depends on the concept of an *informative interval*, which we now define. The *folding operation* maps each point $0 \leq x \leq 1$ to $\hat{x} = \min\{x, 1-x\}$. Two intervals along the target molecule are called *conjugate* if their folded images coincide. This generalizes the definition of conjugate pairs from Karp and Shamir [2000], Muthukrishnan and Parida [1997], since it does not specify the sizes of these intervals or restrict their possible endpoints to a predefined discrete set. This generalization enhances the performance of the signature algorithm, allowing it to overcome sizing errors. Under this folding operation each conjugate pair of intervals maps to a single *folded interval*. A folded interval is called an *informative interval* if it has a substantial density of observed cuts and passes a statistical test indicating that all the restriction sites within it come from one half of the target molecule; *i.e.*, from one member of the corresponding conjugate pair.

Using dynamic programming we identify a set S of disjoint informative intervals. Using the signature algorithm we can resolve S into two classes, such that two folded intervals in S are in the same class if their restriction sites appear to lie in the same half of the target molecule, and in opposite classes otherwise. We then choose a standard orientation of the target sequence in which the restriction sites occurring in intervals from the first class are placed in the left half, and those occurring in intervals from the second class are placed in the right half. In this way the restriction sites from each folded interval I in S are assigned to one of the two conjugate intervals that map onto I. This process is called *resolving the informative intervals*. Finally, we orient each molecule so as to maximize the number of observed cuts in it that lie within folded intervals from S and occur in the "correct" members of the corresponding conjugate pairs.

Once the molecules have been oriented we apply likelihood optimization to determine the restriction sites. To improve the search for restriction sites, we initialize it with a good approximation of the site locations, which is obtained by identification of *good intervals*. Informally, an interval is good if its density of observed cuts is high, and most of the observed cuts within it can be attributed to restriction sites within the interval itself. The process of identifying good inter-

1. Screen out faulty molecules from the unoriented data.
2. Identify informative intervals.
3. Apply the signature algorithm to resolve the informative intervals and orient the molecules.
4. Screen out more faulty molecules from the oriented data.
5. Identify good intervals.
6. Determine the restriction site locations.

Figure 4.1: The general scheme of our algorithm. Stages 1 and 4 are described in Section 4.4. Stages 2 and 5 are described in Section 4.5. Stage 6 is described in Section 4.6.

vals makes use of the fact that the molecules have been oriented. We also screen out molecules suspected to be faulty, first in a pre-processing step, and again after orienting the molecules. The general scheme of our algorithm is presented in Figure 4.1. Subsequent sections describe the different stages of the algorithm in detail.

4.3 Model and Terminology

We now define our probabilistic model of the problem. Similar models were used in Anantharaman et al. [1997], Lee et al. [1998]. Each of the N_{mol} molecules is faulty with (independent) probability p_{faulty}, in which case, it displays (false) cuts with Poisson rate λ_{faulty}. In each proper molecule false cuts are Poisson distributed with rate λ. We denote $\lambda_{av} = p_{faulty}\lambda_{faulty} + (1 - p_{faulty})\lambda$.

We assume there is some unknown number t of (true) restriction sites, with the i-th site R_i located at position μ_i along the molecule. The input data D is a set of N_{mol} lists, $D_1, \ldots, D_{N_{mol}}$. The list D_m of the m-th molecule contains $N_{cut}(m)$ entries (observed cuts), at positions $c_{m,1}, \ldots, c_{m,N_{cut}(m)}$. Further define N_{cuts} to be $\sum_m N_{cut}(m)$. In each proper molecule, the cut R_i is actually *ob-*

served (registers as a cut) with (independent) probability p_i. Its actual observed position is normally (and independently) distributed around μ_i with variance σ_i^2. Additionally, each molecule is independently oriented as straight or reverse with equal probability. Our problem is to determine the restriction sites $\{\mu_i\}$ from the data D. In the course of doing this we shall also determine the orientations of the molecules and the other parameters of the probabilistic model.

4.4 Screening Out Faulty Molecules

4.4.1 Screening Oriented Data

We begin by describing how to screen out faulty molecules from oriented data, and then, in Section 4.4.2, present the slightly more complicated analysis, for unoriented data. This order of description is the reverse of the chronological order of these stages in our algorithm.

Denote by $\vec{D} = \{\vec{c}_{m,j}\}$ the data after the molecules have been oriented. We define f, the probability density of observing a cut at x, to be the probability of observing a cut in a short interval centered at x, per molecule, per unit length of the interval. Formally, $f(x) : [0,1] \mapsto \mathcal{R}$ is:

$$f(x) \equiv \lim_{\epsilon \to 0} \frac{\left|\left\{(m,j) : |\vec{c}_{m,j} - x| < \frac{\epsilon}{2}\right\}\right|}{\epsilon N_{mol}} \tag{4.1}$$

Cuts in proper molecules tend to be observed near restriction sites, while cuts in faulty molecules occur at random locations. Thus, the observed cuts in proper molecules should tend to occur at points of higher probability density than the observed cuts in faulty molecules. This is the basis for our screening procedure, which we now detail more formally.

According to our assumptions,

$$f(x) = \lambda_{av} + (1 - p_{faulty}) \sum_i p_i Normal(x, \mu_i, \sigma_i^2)$$

where $Normal(x, \mu, \sigma^2)$ is the probability density, at point x, of a normal random

variable with mean μ and variance σ^2, *i.e.*,

$$Normal(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The overall average value of f, denoted f_{av}, is

$$f_{av} = \lambda_{av} + (1 - p_{faulty}) \sum_i p_i$$

However, depending upon whether x is near restriction sites, $f(x)$ behaves differently. If x is not near any μ_i, then the probability of an observation in this interval is:

$$f(x) \simeq \lambda_{av}$$

In contrast, if x is close to some μ_i, and the μ_i-s are sufficiently separated, then

$$f(x) \simeq \lambda_{av} + (1 - p_{faulty}) p_i Normal(x, \mu_i, \sigma_i^2)$$

We now describe how we can estimate $f(x)$ from the data, and how we can use this estimation. To this end, define $X_I(D)$, for an interval I, as the number of cuts observed in D, inside I. Note that $X_I(D)$ is a random variable. Let $I(x, \epsilon)$ denote the interval of length ϵ centered at x. Then for a small ϵ, we may estimate $f(x)$ by:

$$\phi_{\epsilon,\vec{D}}(x) = \frac{X_{I(x,\epsilon)}(\vec{D})}{\epsilon N_{mol}}$$

We compute $\phi_{\epsilon,\vec{D}}(x)$ for every cut position $x = \vec{c}_{m,j}$. In practice, we choose $\epsilon = \epsilon_x$ so that $X_{I(x,\epsilon)}(\vec{D})$ will be some predetermined constant. This way we have a large enough sample size when $\phi_{\epsilon,\vec{D}}(x)$ is small, while concentrating on a small interval around x for better accuracy, when $\phi_{\epsilon,\vec{D}}(x)$ is large.

For each molecule m, we compute ϕ_m, the average of the estimated density in all the molecule's observed cut positions:

$$\phi_m = \frac{\sum_{j=1}^{N_{cut}(m)} \phi_{\epsilon,\vec{D}}(\vec{c}_{m,j})}{N_{cut}(m)}$$

The expected value of ϕ_m for a faulty molecule m is f_{av}, which in turn, can be estimated as

$$f_{av} \simeq \frac{\sum_m N_{cut}(m)}{N_{mol}}$$

In contrast, for a proper molecule m the expected value of $\phi_{\epsilon,\vec{D}}(\vec{c}_{m,j})$ is larger, and thus, the expected value of ϕ_m is larger.

The molecules with the smallest ϕ_m are the ones most likely to be faulty, and should therefore be discarded. We repeatedly find the molecule m with minimal ϕ_m (the molecule most likely to be faulty), designate it as faulty, filter it out of our data set and recompute the quantities ϕ_m on the basis of the remaining molecules. We repeat this process N_{faulty} times, where N_{faulty} is an input number. We note that a more subtle analysis might be able to estimate N_{faulty} from the observed distribution of ϕ_m.

4.4.2 Screening Unoriented Data

We now refer to the original unoriented data set D. We fold the molecule in half to create a *folded* data set $\widehat{D} = \{\widehat{c_{m,1}}, \ldots, \widehat{c_{m,N_{cut}(m)}}\}_{m=1}^{N_{mol}}$, using the folding operation $\widehat{x} = \min\{x, 1-x\}$ introduced in Section 4.2.3. Note, that if the orientation of a molecule D_j is unknown, then the folded molecule $\widehat{D_j}$ data conveys all the information obtainable from D_j on each cut separately. We can now screen out faulty molecules from the oriented data using the same idea as in Section 4.4.1. We present the formulae for completeness:

We define $\hat{f}$, the probability density, of observing a (folded) cut at x, to be the probability of an observed cut being in a short interval centered at x, relative to the length of this interval. Formally, $\hat{f}(x) : [0, \frac{1}{2}] \mapsto \mathcal{R}$, is defined:

$$\hat{f}(x) \equiv \lim_{\epsilon \to 0} \frac{\left|\left\{(m,j) : |\widehat{c_{m,j}} - x| < \frac{\epsilon}{2}\right\}\right|}{\epsilon N_{mol}} \tag{4.2}$$

where our assumptions imply:

$$\hat{f}(x) = 2\lambda_{av} + (1 - p_{faulty}) \sum_i p_i (Normal(x, \mu_i, \sigma_i^2) + Normal(x, 1-\mu_i, \sigma_i^2))$$

The expected value of $\hat{f}(x)$ is $\hat{f_{av}} = 2f_{av}$, which is estimated by $2\frac{\sum_m N_{cut}(m)}{N_{mol}}$. Again,

$$\hat{f}(x) \simeq \begin{cases} 2\lambda_{av} + (1 - p_{faulty}) p_i Normal(x, \widehat{\mu_i}, \sigma_i^2) & \text{if } x \text{ is near } \widehat{\mu_i} \\ 2\lambda_{av} & \text{otherwise} \end{cases}$$

An unbiased estimator of $\hat{f}(x)$ is

$$\phi_{\epsilon,\hat{D}}(x) = \frac{X_{I(x,\epsilon)}(\hat{D})}{\epsilon}$$

and we can compute the average of the estimated folded density in all the molecule's observed cut positions:

$$\hat{\phi}_m = \frac{\sum_{j=1}^{N_{cut}(m)} \phi_{\epsilon,\hat{D}}(\widehat{c_{m,j}})}{N_{cut}(m)}$$

We observe that the expected value of this average estimated density is, for a faulty m, $\hat{\phi}_m = \hat{f}_{av}$; for a proper m, $\hat{\phi}_m$ is larger. We can therefore perform a screening procedure similar to the one described in Section 4.4.2, this time using $\hat{\phi}_m$, instead of ϕ_m.

From this point on, we shall assume there are no more faulty molecules, and N_{mol} will denote the number of proper molecules.

4.5 Good Intervals and Informative Intervals

4.5.1 Good Intervals

Once the molecules have been oriented we seek to determine the restriction sites by an iterative maximum likelihood computation. Since the method we use (the EM algorithm) is only guaranteed to converge to a local maximum, it is important to start the iteration with a good estimate of the restriction site locations. For this purpose we attempt to localize the restriction site positions to a set of disjoint *good intervals*. Informally, an interval is good if its density of observed cuts is high and most of the observed cuts within it can be attributed to restriction sites which indeed reside in this interval. In this subsection we describe the process of finding the good intervals.

Goal and Compromise

We would like to partition the observed cuts according to their true restriction sites, and to separately handle all the observed cuts originating from a single restriction site. More formally, we want to find intervals $I_1, \ldots, I_t$ contained in $(0,1)$, so that I_i contains only observations originating from R_i, and all of the observations originating from R_i are contained in I_i. However, this is not always possible, because the sizing error may be large relative to the distance between restriction sites. In such cases, there are true restriction sites R_i and R_{i+1} such that the observed cut sites originating from R_i are interleaved with those originating from R_{i+1}.

We therefore settle for finding disjoint intervals $I_1, \ldots, I_k$, and a partition of the sites $R_1, \ldots, R_t$ into k disjoint subsets, with I_i containing mostly true observed cuts originating from the restriction sites of the i-th subset. We call such intervals *good intervals*.

Initial Filtering

We restrict attention to the $O(N_{cut}^2)$ intervals having observed cuts as their end points. We also reject unreasonably long intervals. More precisely, we discard intervals in which at least one of our molecules exhibits more than k cuts (we use $k = 2$). For each such interval I, we estimate the average value of the density $f(x)$ within I by $\frac{X_I(\vec{D})}{|I|N_{mol}}$. We further eliminate those intervals for which this value is smaller than a chosen threshold, as they are unlikely to be good. On the remaining intervals we perform a more subtle analysis, as explained in the next section.

Finding Good Intervals

We consider the oriented data set $\vec{D}$. For each interval $I = (a,b) \subset (0,1)$ we wish to estimate the number of restriction sites within I and the probability that I contains at least one restriction site. We base this estimation on the random variables $X_I(\vec{m})$, where $\vec{m}$ ranges over the molecules in the probability space $\vec{D}$. Assuming there are j true restriction sites in I, and that a negligible number of

the observed cuts within I arise from restriction sites outside I, the distribution of $X_I(\vec{m})$ is a function of the following parameters:

1. The false cut rate λ
2. For each restriction site R_i within I, the probability p'_i of a true cut originating from R_i to be observed in I. For simplicity, we further approximate all the p'_i values by a single p'_I.

We define:

$$\begin{aligned} Poisson(h,\alpha) &= \frac{e^{-\alpha}\alpha^h}{h!} \\ Bin(i,j,p) &= \binom{j}{i} p^i (1-p)^{j-i} \\ P(k,j,\lambda,p'_I) &= Prob\{X_I(\vec{m}) = k | j \text{ restriction sites in } I, \lambda, p'_I\} \\ &= \sum_{h+i=k, i\le j} Poisson(h,\lambda|I|) Bin(i,j,p'_I) \end{aligned}$$

Each molecule in $\vec{D}$ gives rise to an independent sample $X_I(\vec{m})$ from this distribution. From these samples we can obtain the empirical frequency count:

$$\mathcal{X}_k(\vec{D}) = |\{\vec{m} \in \vec{D} | X_I(\vec{m}) = k\}|$$

Hence:

$$\begin{aligned} Likelihood(j,\lambda,p'_I) &= Prob(\vec{D}|j,\lambda,p'_I) \\ &= \prod_{\vec{m}\in\vec{D}} P(X_I(\vec{m}),j,\lambda,p'_I) \\ &= \prod_k P(k,j,\lambda,p'_I)^{\mathcal{X}_k(\vec{D})} \end{aligned}$$

Using standard numerical maximization techniques, for each such j, we can get the most likely parameters, λ and p'_I, given the observed values of $\mathcal{X}_k(\vec{D})$. We optimize these parameters, and denote the log of the maximum likelihood by $L^j(I)$. Let $L(I) = \max_{j>0} L^j(I) - L^0(I)$ be the log-likelihood of the most likely

such assumption, compared to the null hypothesis of no restriction sites at all. In practice, it is enough to consider only small values of j, *i.e.*, $j \leq 2$. The higher $L(I)$ is, the more we consider I a good interval.

We extend this measure to any set S of non-overlapping intervals:

$$L(S) = \sum_{I \in S} L(I)$$

It is possible to find the set S maximizing $L(S)$ by dynamic programming, as follows. We denote the ordered set of observed cut locations by $x_1, \ldots, x_{N_{cuts}}$, and set $x_0 = 0$. Let $OPT(k)$ be an optimal set of non-overlapping good intervals in $[0, x_k]$ and let $F(k)$ be the corresponding optimal value. Then $F(0) \equiv 0$, and we compute for $k = 1, 2, \ldots, N_{cuts}$:

$$F(k) = \max\{F(k-1), \max_{j<k-1}\{F(j) + L([x_{j+1}, x_k])\}\} \tag{4.3}$$

and save $OPT(k)$, a set of intervals attaining that optimum. $OPT(N_{cuts})$ is the desired solution.

4.5.2 Informative Intervals

Naive Discretization of the Data

We now discuss the original, unoriented data set D. We naturally "fold the molecule in half" to create a *folded* data set $\widehat{D} = \{\widehat{c_{m,1}}, \ldots, \widehat{c_{m,N_{cut}(m)}}\}_{m=1}^{N_{mol}}$. The straightforward way to discretize the input data is to uniformly partition the (folded) molecule m into k segments of equal length, replacing it with a binary vector $v = (v_1, \ldots, v_k)$, such that v_j is an indicator variable, indicating whether a cut was observed in the j-th segment. However, real life sizing error is rather large: In many cases, it is of the same order of magnitude as the distance between nearby restriction sites (1-3 percent of the molecule length). This causes naive discretization to fail.

Finding Informative Intervals

We now devise a discretization method that non-uniformly slices the interval $[0,\frac{1}{2}]$ corresponding to the folded molecule in a manner more favorable for our purposes. This method aims at finding those intervals whose associated conjugate pairs supply much information on the orientation of the data. In order for such a pair, $(I, \bar{I})$, to be informative, there must be strong evidence that either I contains at least one restriction site and I' does not, or vice versa. We measure the informativeness of a folded interval by a likelihood calculation analogous to the one given in Section 4.5.1. The added complications are that we now consider the original, unfolded data set, D, and examine the two dimensional random variable $Y_I(m) = (X_I(m), X_{\bar{I}}(m))$, for a single molecule $m \in D$. Clearly, its distribution is symmetric, *i.e.*, $Prob(Y_I(m) = (k,l)) = Prob(Y_I(m) = (l,k))$. We would use the observations we have on this variable to learn its distribution, and estimate the probability of having true sites at I.

Assuming there are i true restriction sites in I, and j restriction sites in $\bar{I}$, this distribution can be approximated as a function of the following instance parameters:

1. The noise (false cuts) rate λ.
2. For each restriction site R_x of the $i+j$ sites, the probability $\widehat{p'_x}$ of a true cut originating from R_x to appear in I or in $\bar{I}$. As in Section 4.5.1, we approximate all the $\widehat{p'_x}$ values by a single $\widehat{p'_I}$.

We can define:

$$\begin{aligned}\widehat{P}((k,l),i,j,\lambda,\widehat{p'_I}) &= Prob\{Y_I(m) = (k,l) | i,j \text{ restriction sites}, \lambda, \widehat{p'_I}\} \\ &= \sum_{\substack{g+g'=k \\ h+h'=l}} Poisson(g+h, 2\lambda|I|) Bin(g', i, \widehat{p'_I}) Bin(h', j, \widehat{p'_I})\end{aligned}$$

The molecules in D are N_{mol} independent random observations of this random variable, from which we obtain the empirical frequency count:

$$\mathcal{Y}_{k,l}(D) = |\{m \in D | Y_I(m) = (k,l) \text{ or } Y_I(m) = (l,k)\}|$$

Hence:

$$\begin{aligned} Likelihood(i,j,\lambda,\widehat{p'_I}) &= Prob(D|i,j,\lambda,\widehat{p'_I}) \\ &= \prod_{m\in D} \widehat{P}(Y_I(m),i,j,\lambda,\widehat{p'_I}) \\ &= \prod_{k,l} \widehat{P}((k,l),i,j,\lambda,\widehat{p'_I})^{\mathcal{Y}_{k,l}(D)} \end{aligned}$$

Using standard numerical maximization techniques, we find, for each such i, j, the most likely parameters, given the observed values of $\mathcal{Y}_{k,l}(D)$. We optimize these parameters, and denote the log of this likelihood, by $\hat{L}^{i,j}(I)$. We define:

$$\hat{L}(I) = \max_{i>0} \hat{L}^{i,0}(I) - \max_{i,j\neq 0 \text{ or } i=j=0} \hat{L}^{i,j}(I)$$

In other words, $\hat{L}(I)$ is the likelihood of the most likely such informative assumption, compared to the null-hypothesis of the most likely non-informative assumption. The higher $\hat{L}(I)$ is, the more informative interval I is. In practice, it is enough to consider only small values of i, j, *i.e.*, $i + j \leq 2$.

After computing $\hat{L}(I)$ for each candidate interval, we employ a dynamic program, analogous to the one presented in Equation 4.3, in order to find a set of non overlapping intervals $I_1, \ldots, I_k$, with maximal $\sum_i \hat{L}(I_i)$. This is a set of informative intervals, and we use it to discretize the data to a $k \times N_{mol}$ binary matrix, on which the signature methods of Karp and Shamir [2000] are applicable.

4.6 Determining Restriction Site Locations

4.6.1 Probabilistic Model

After stage 5 of the algorithm, we have a set of good intervals, $I_1, \ldots, I_k$ in $(0, 1)$. For each I_i, and for each value of j, we know $L(I_i, j)$, the likelihood of the data within I_i (for the best values of λ and p'_I) assuming there are j restriction sites in I_i and that no observed cuts within I_i arise from restriction sites outside I_i. In practice, j is no more than 2 (for larger values, this likelihood is negligible). For any vector $J = (j_1, \ldots, j_k)$, we can estimate the likelihood of the event that, for

each i, I_i contains exactly j_i restriction sites, by $\prod_{i=1}^{k} L(I_i, j_i)$. This formula is a good approximation to the actual likelihood, provided that for each good interval I_i, restriction sites outside I_i do not give rise to a significant number of observed cuts within I_i. For each vector J with significantly high likelihood, we generate initial values for λ and the set of triplets (p_r, μ_r, σ_r), where r ranges from 1 to $\sum_{i=1}^{k} j_i$, as follows: for an interval I_i containing j_i restriction sites, a different site is placed at the center of each disjoint sub-interval of size $\frac{|I_i|}{j_i}$, and for each such site, p_r is set to p'_{I_i}. λ is set to $\frac{N_{cuts}}{N_{mol}} - \sum p_r$. It turns out that there are not too many likely values for the vector J, and we perform a heuristic likelihood maximization from the starting solution associated with each such vector. Although our starting solutions have all their restriction sites within good intervals, this property is not required to hold at later iterations.

4.6.2 Approximate Likelihood Score

We now describe the score which we optimize. Let $\psi = (\lambda, \{(p_r, \mu_r, \sigma_r)\}_{r=1}^{k})$ be a set of assumed parameters. We need to compute the likelihood score $s(\psi)$, *i.e.*, the probability of the data given ψ:

$$s(\psi) = Prob(\vec{D}|\psi) = \prod_{m} Prob(\vec{D_m}|\psi) \tag{4.4}$$

For a molecule m with the observed cuts $\vec{c}_{m,1}, \ldots, \vec{c}_{m,N_{cut}(m)}$, we do not know which of these observed cuts originated from which of the true restriction sites in ψ, and which are due to background noise. A matching between the observed cuts $\vec{c}_{m,1}, \ldots, \vec{c}_{m,N_{cut}(m)}$ and the assumed restriction sites (or noise) is called an *alignment* between m and ψ. We can therefore write:

$$s(\psi) = \prod_{m} \sum_{a} Prob\left(\begin{array}{l} \vec{D_m}\psi, \text{ and the alignment } a \\ \text{between } m \text{ and } \psi \end{array} \middle| \, \psi \right) \tag{4.5}$$

The inner summation is done over all possible alignments between the restriction sites assumed by ψ, and the cuts observed in $\vec{D_m}$.

We call an alignment *order preserving* if for every two observed cuts c, c', which are matched to restriction sites r, r', respectively, $c < c'$ iff $\mu_r < \mu_{r'}$. Other

alignments are highly unlikely. In practice, we therefore perform the summation of Equation 4.5 only over the order preserving alignments. We calculate the logarithm of the required probability by dynamic programming.

Typically, the score of the optimal alignment a^* is considerably higher than the score of any other alignment. Therefore calculating the probability of a^* is a reasonable approximation to the true likelihood score. We denote this score by $s^*(\psi)$. $s^*(\psi)$ can be computed by a recurrence relation similar to the dynamic program used to compute $s(\psi)$, the only difference being that we compute maximum instead of summation, replacing Equation 4.5 with:

$$s^*(\psi) \cong \prod_m \max_a Prob\left(\begin{array}{l} \vec{D_m}\psi, \text{ and the alignment } a \\ \text{between } m \text{ and } \psi \end{array} \middle| \psi \right) \tag{4.6}$$

We have found the difference between these two scores to be small.

4.6.3 EM Optimization

We use a variant of the EM (Expectation Maximization) heuristic for this optimization, as detailed below. We remark that Dančík and Waterman [1997], Anantharaman et al. [1997] and Lee et al. [1998] use EM and other heuristics (gradient descent and Monte Carlo Markov Chain simulation) to optimize a related score. For completeness, we include a description of this stage in our algorithm, although it is not unique to this work. In contrast to prior art, we have the advantage of working with oriented data using a good starting solution.

At each iteration of the optimization procedure, we assume a parameter set $\tilde{\psi}$, and reassign observed cuts to the postulated restriction sites. Throughout this procedure, we maintain, for each restriction site r, the set $L(r)$ of locations of cuts, observed across all the molecules in the data set, that were assigned to r by the optimal alignments between each of the molecules and $\tilde{\psi}$; $L(noise)$ is the set of all observed cut locations that optimal alignments attribute to background noise. Define $N(r) = |L(r)|$, $N(noise) = |L(noise)|$.

In order to find the maximum of this score function, we iterate as follows:

1. **Expectation:** For each restriction site r, estimate its parameters given the set $L(r)$ of all observed cut locations $\{l_j\}_{j=1}^{N(r)}$ aligned with r. This statistical estimation is performed as follows:

   - $\tilde{p_r} = \frac{N(r)}{N_{mol}}$ is a maximum likelihood estimator for p_r.
   - Similarly, $\tilde{\lambda} = \frac{N(noise)}{N_{mol}}$ is a maximum likelihood estimator for λ.
   - Assuming that the locations in $L(r)$ are normally distributed with expectation μ_r, the term $\tilde{\mu_r} = \frac{\sum l_j}{N(r)}$ is a maximum likelihood estimator for μ_r.
   - Assuming that the locations in $L(r)$ are normally distributed with variance σ_r^2, the term $\tilde{\sigma_r^2} = \frac{\sum l_j^2}{N(r)} - \tilde{\mu_r}^2$ is a maximum likelihood estimator for σ_r^2.

2. **Maximization:** Given the current set $\tilde{\psi} = (\tilde{\lambda}, \{(\tilde{p_r}, \tilde{\mu_r}, \tilde{\sigma_r})\}_{r=1}^{k}$ of estimated parameters, for each molecule m, find the optimal alignment $a^*(m)$ between $\tilde{\psi}$ and m. Adjust the sets $L(r)$ for each r, to contain the observed cut locations that were matched to r by the new optimal alignments $\{a^*(m)\}_m$.

Note, however, that even if our probabilistic model is correct, and the locations of the observed cuts originating from the restriction site r indeed have the distribution $Normal(\mu_r, \sigma_r^2)$, the locations in $L(r)$ do not have the same distribution. Rather, the distribution of these locations is a doubly truncated Gaussian. Luckily, it seems from our experiments with real data that the difference between these distributions is negligible.

In practice, we have observed that the score s^* occasionally splits a site into two nearby sites with lower p_i-s. In order to make sure this only happens when the evidence for such a pair is solid, we do the following: For each pair of nearby sites $r, r+1$, we count the number $M(r, r+1)$ of molecules m, for which both sites are matched by $a^*(m)$. We multiply the score, s^*, by the probability of observing $M(r, r+1)$, given that there are two independent cuts $r, r+1$. This modification seems to solve the splitting problem in practice, provided there are no chains of such nearby sites.

4.7 Results

4.7.1 Simulations

The latter stage of our algorithm is actually a hard-decision EM algorithm, iteratively improving s^*. Our main contribution is being able to perform this search on the reduced space of oriented molecules. After orientation, any existing method for tackling this problem can, in principle, be employed, with the advantage of knowing the correct orientation. We therefore gathered statistics indicating our success in orienting the molecules, presenting the percentage of mis-oriented molecules as a figure of merit.

We have applied our algorithm to many data sets, generated randomly according to a range of statistical scenarios: we altered the number of restriction sites, digestion rate, magnitude of sizing error, and false cut rate. For each such set of parameters, data sets of different size (*i.e.*, number of molecules) were analyzed. The results are presented in Figure 4.7.1. They show that for standard values of the model parameters errors in orientation are quite rare, dropping further, of course, as more molecules are examined. Furthermore, these results are stable with respect to alterations of instance parameters.

4.7.2 Real Data

Our algorithm was implemented in a blind test on real biological data provided by D. Schwartz's laboratory. There were eight data sets, with cosmids (50Kb) and BAC (130-180Kb) molecules. We encountered at most a dozen restriction sites in each data set, with the digestion rate varying widely from ~ 0.1 to ~ 0.9 between data sets. There were differences of up to 0.3 in the digestion rate between sites in the same data set. The false cut rate varied too, sometimes exceeding one false cut per molecule, on the average. The average sizing error was on the order of magnitude of $\frac{1}{100}$ to $\frac{1}{20}$ of the molecule length. Examples of the results are given in Figures 4.3, 4.4 and 4.5. Figures 4.3 and 4.5 describe two experiments with disjoint data-sets for the same molecule, demonstrating application of our algorithm for different digestion rates.

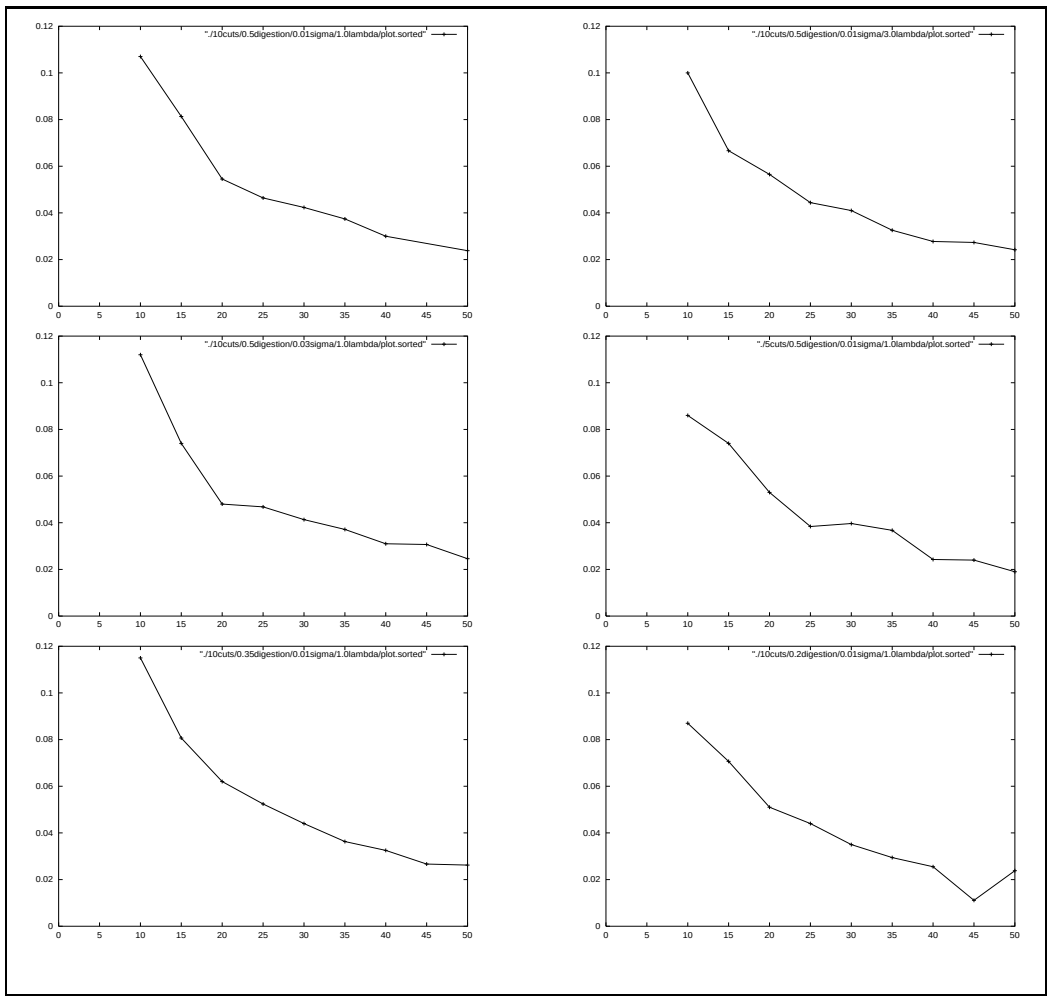
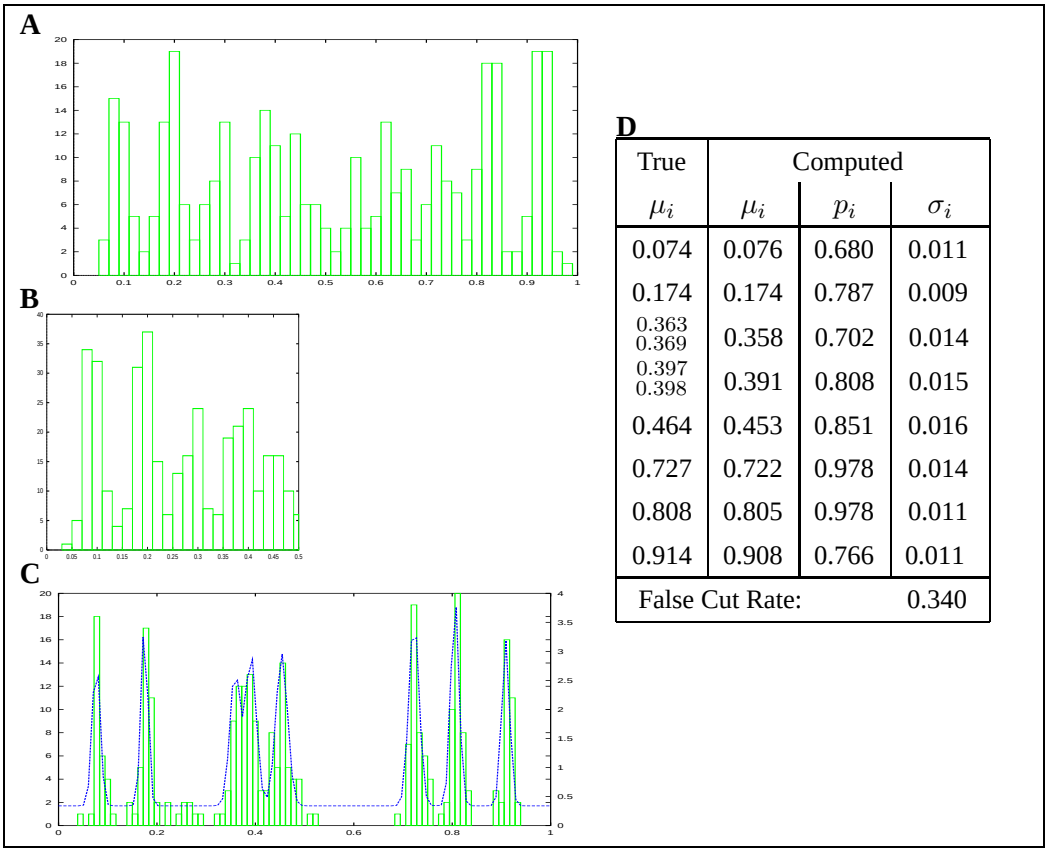


Figure 4.2: Results on simulated data. Each graph presents results with a specific set of simulation parameters (number of true cuts, digestion rate, standard sizing error, and false cut rate). The x-axis is the number of molecules, while the y-axis is the fraction of the molecules that are incorrectly oriented by the algorithm, averaged across 50 simulated data sets with the appropriate parameters and number of molecules. The basic scenario (top left) is for 10 restriction sites, digestion rate of 0.5, sizing error of standard deviation 0.01, and false cuts rate of 1.0. The other cases shown are for larger (3.0) false cut rate (top right), larger (0.03) sizing error (middle left), fewer (5) restriction sites (middle right), and lower (0.35 and 0.2) digestion rates (bottom left and right, respectively).

Bud Mishra and Thomas Anantharaman of N.Y.U. kindly examined our results and then provided us with the true restriction sites, as determined directly from sequence data or indirectly by inference from pulsed field gel electrophoresis data combined with the optical mapping data. In several cases where two restriction sites were separated by less than 1000 bases, our algorithm reported only one site. This difficulty was also encountered in Anantharaman et al. [1997] and may be an inherent problem due to the inability of the imaging system to detect small restriction fragments. Apart from this, our results were correct on four of the eight examples; these include the examples shown here. On two further examples the results were correct except that one restriction fragment with a low digestion rate was missed. More careful inspection showed that almost all (95%) of the molecules were correctly oriented for these two examples, with only the last stage of the algorithm, EM optimization, being inaccurate. On two additional examples where the data was of low quality (as measured by the method reported in Anantharaman et al. [1997]) the results were less good. In one of these examples two restriction sites were missed, despite the fact that they were evident by inspection from the histogram of observed cut locations in the oriented molecules. In the other example our program failed, missing two true restriction sites and introducing three false ones. On the positive side, we provided the correct solution for the data-set in Figure 4.5, which was classified by Anantharaman et al. [1997] as the hardest among the eight data-sets we have considered.

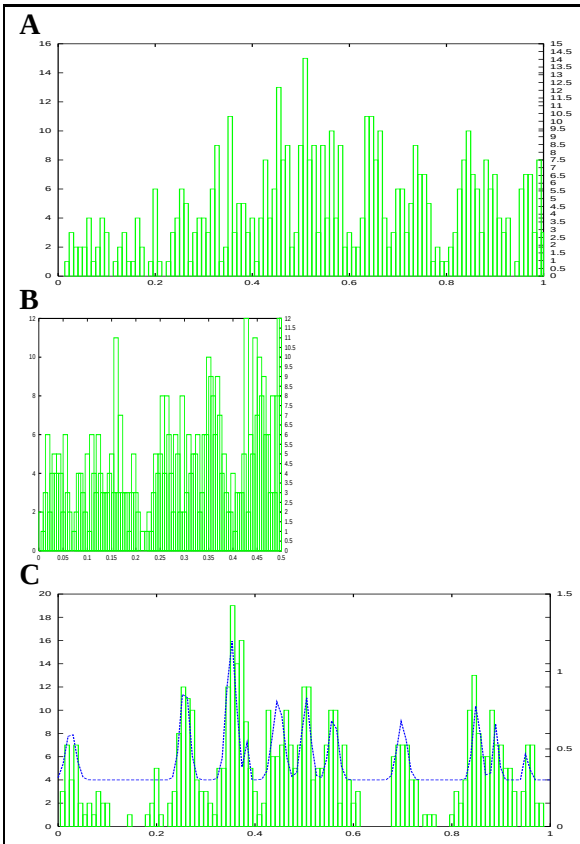
It appears that our program tends to determine the orientations of the molecules correctly, thereby substantially reducing the parameter space for any subsequent maximum likelihood method. The occasional misestimation of the number of restriction sites can be alleviated by further tuning of the algorithm for determining the restriction site data from the oriented molecules.

The results presented here are the first blind test of a novel algorithm. The performance of our algorithm does not match up, at this point, to that of Anantharaman et al. [1997], which has been employed for three years now with considerable success. The results illustrate that a refined discretization procedure, combined with the signature method, greatly help the global optimization in cases where the likelihood landscape is unfavorable for optimizing over unoriented data. Since the



True	Computed		
μ_i	μ_i	p_i	σ_i
0.074	0.076	0.680	0.011
0.174	0.174	0.787	0.009
$\begin{matrix}0.363\\0.369\end{matrix}$	0.358	0.702	0.014
$\begin{matrix}0.397\\0.398\end{matrix}$	0.391	0.808	0.015
0.464	0.453	0.851	0.016
0.727	0.722	0.978	0.014
0.808	0.805	0.978	0.011
0.914	0.908	0.766	0.011
False Cut Rate:			0.340

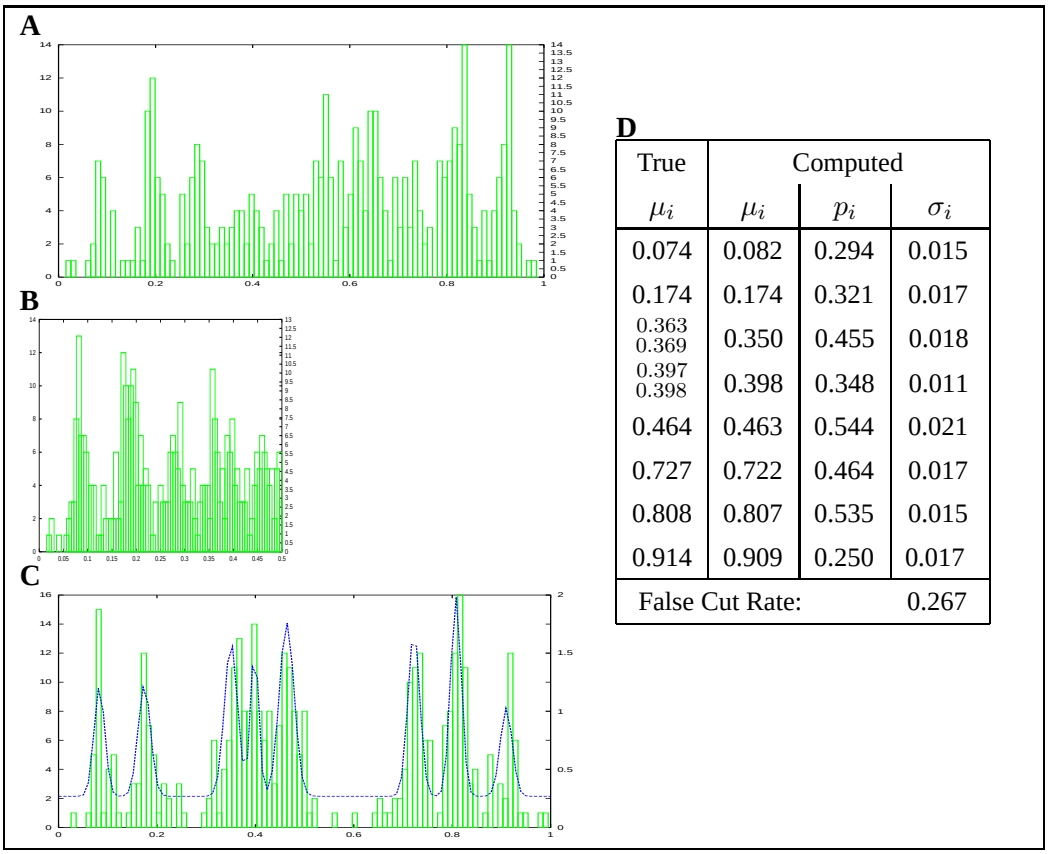
Figure 4.3: Test example 6262-1: Instance parameters: 54 molecules, 370 observed cuts in total. 7 molecules screened out. A: Histogram of observed cuts in equalized sub-intervals. B: Histogram of the 'folded' data. C: Histogram of the oriented, cleaned data superimposed with the optimal density function. D: The suggested solution. Pairs of sites in the same row of the "True" column indicate close-by sites that cannot be distinguished.



D

True	Computed		
μ_i	μ_i	p_i	σ_i
0.008	short: reported as 0		
0.024 0.031	0.025	0.097	0.014
0.254	0.257	0.180	0.013
0.346	0.352	0.266	0.013
0.361	0.381	0.050	0.004
0.448	0.447	0.162	0.015
0.515	0.504	0.147	0.012
0.562	0.559	0.119	0.013
0.697	0.698	0.111	0.013
0.837 0.854	0.850	0.129	0.011
0.889	0.889	0.079	0.007
0.951 0.953	0.953	0.050	0.005
False Cut Rate:			0.300

Figure 4.4: Test example 6401: Instance parameters: 280 molecules, 494 observed cuts in total. 7 molecules screened out. See Figure 4.3 for legend.



True	Computed		
μ_i	μ_i	p_i	σ_i
0.074	0.082	0.294	0.015
0.174	0.174	0.321	0.017
0.363 0.369	0.350	0.455	0.018
0.397 0.398	0.398	0.348	0.011
0.464	0.463	0.544	0.021
0.727	0.722	0.464	0.017
0.808	0.807	0.535	0.015
0.914	0.909	0.250	0.017
False Cut Rate:			0.267

Figure 4.5: Test example 6262-0: Instance parameters: 114 molecules, 396 observed cuts in total. 7 molecules screened out. See Figure 4.3 for legend.

orientation phase of our algorithm seems to be very successful, one natural suggestion for future work is to combine that phase with a refined global optimization procedure for finding the cut sites on the more favorable landscape of oriented data.

Chapter 5

Reconstructing Gene Order of the Median Genome

In this chapter we discuss changes in gene order between genomes during evolution. These changes can be measured by the *breakpoint distance* between gene orders: the number of gene pairs adjacent in one but not the other. For three contemporary gene orders, the ancestral gene order in their evolutionary divergence point is called their *median*. A fundamental task is to find this median given the contemporary gene orders, so that its sum of (breakpoint) distances to the leaves is minimum. We prove this problem to be NP-hard. This result had been published in [Pe'er and Shamir, 1998]. We further provide polynomial time approximation algorithms that guarantee a $\frac{7}{6}$ approximation. This study has been published in [Pe'er and Shamir, 2000a].

5.1 Introduction

The breakpoint distance between two n-permutations is the number of pairs that appear consecutively in one but not in the other. In the median problem for breakpoints one is given a set of permutations and has to construct a permutation that minimizes the sum of breakpoint distances to all the original ones. Recently, the

problem was suggested as a model for determining the evolutionary history of several species based on their gene orders. We show that the problem is already NP-hard for three permutations, and that this result holds both for signed and for unsigned permutations.

The study of genome rearrangements has drawn a lot of attention in recent years. Large amounts of genomic data on various organisms become available rapidly, and they make possible for the first time a large scale study of evolutionary relations among species by comparing the order of appearance of common genes in their chromosomes. Changes in gene order are much less frequent than point mutations, and therefore, in principle, one can elucidate the evolutionary history of speciation more precisely and further backwards in time using gene orders. This is done by comparing gene orders in the studied species and reconstructing the sequences of gene rearrangement events that led from the ancestral genome species to the current species.

When restricting the discussion to only one chromosome, and assuming genes occur only once, genomes are modeled as permutations. When taking into account the orientation of the genes, the objects discussed are *signed* permutations. The case where all rearrangement events are reversals (inversion of a chromosome segment) has been studied intensively in recent years. For two species, this problem, of finding the *reversal distance* between two permutations, is already NP-hard in case the gene orders are given as unsigned permutations [Caprara, 1997b], but it is polynomial if the permutations are signed [Hannenhalli and Pevzner, 1999, Berman and Hannenhalli, 1996, Kaplan et al., 1999]. Finding a tree minimizing the total number of reversals for three signed permutations is already NP-hard [Caprara, 1997a]. Other models of the genome [Hannenhalli, 1996, Kececioglu and Ravi, 1995] or of the assumed events were also studied [Hannenhalli and Pevzner, 1995, Hannenhalli, 1996, Sankoff and Nadeau, 1996, Bafna and Pevzner, 1995, DasGupta et al., 1996]. When studying more than two species the key problem arising is to reconstruct the phylogenetic tree achieving minimal total distance along edges (most parsimonious), given only permutations (gene orders) at the leaves (contemporary species). When the topology of the tree is restricted to a star, this problem is known as *the median problem* [Sankoff et al., 1996a, DasGupta et al., 1996].

Recently, Sankoff and Blanchette [1997] introduced a new model for studying the reconstruction of evolutionary tree of more than two species. They argued forcefully that the reversal distance and similar distance measures have certain weaknesses that render them inappropriate for studying complex trees, and suggested a simpler criterion of *breakpoint distance*, i.e., merely counting the number of breakpoints between every two permutations connected by an edge of the tree. This number can be easily computed for a single edge, for signed and unsigned permutations. Sankoff and Blanchette studied *the median problem for breakpoints*, and developed several efficient heuristics for the problem. However, the computational complexity of this problem was not determined. Here we settle that question by showing that the problem is NP-hard already for three permutations. The hardness result applies both to the signed and to the unsigned model. The same result was obtained independently by Bryant [1998]. Then we give a constant factor polynomial approximation algorithm for the signed model.

The chapter is organized as follows: Section 5.2 proves hardness of the median problems. Section 5.2.1 recalls the standard notation in the field, which we follow. Section 5.2.2 defines further terminology used throughout the chapter. Section 5.2.3 defines a problem which is equivalent to the median problem for breakpoints, for 3 signed permutations, and proves these problems are NP-complete. Section 5.2.5 extends the hardness result also to unsigned permutations.

Section 5.3 studies approximation algorithms to the median problems. Section 5.3.1 provides necessary notation and definitions. Section 5.3.2 presents a very straightforward $\frac{6}{5}$-approximation algorithm, while section 5.3.3 presents a more subtle algorithm and tighter analysis, achieving an approximation ratio of $\frac{7}{6}$. In section 5.3.4 we derive an approximation algorithm for the median problem on four taxa.

5.2 Hardness of the problem

5.2.1 Preliminaries

Let d be a measure of genomic distance, namely an integer distance function on the space of (signed) permutations. The (signed) median problem is defined as follows:

Instance: Three (signed) permutations π_0, π_1, π_2, and an integer k
Question: Does there exist a (signed) permutation $\widehat{\pi}$ s.t. $\sum_i d(\pi_i, \widehat{\pi}) \leq k$?

We denote a signed permutation π on n elements by $(\pi(1), \pi(2), \ldots, \pi(n))$, with $\pi(i) \in \{\pm 1, \ldots, \pm n\}$, and $|\pi(i)| \neq |\pi(j)|$ for $i \neq j$. We rescale a signed permutation on n elements to an unsigned permutation on $2n$ elements, with each positive element $\pi(j) = i$ mapped to the pair $\pi(2j-1) = 2i-1, \pi(2j) = 2i$ and each negative element $\pi(j) = -i$ mapped to the pair $\pi(2j-1) = 2i, \pi(2j) = 2i-1$. We further expand an unsigned permutation by defining $\pi(0) = 0$, and $\pi(n+1) = n+1$. We call this transformation of a signed n element permutation into an unsigned $2n+2$ elements the *standard augmentation* (see also Hannenhalli and Pevzner [1999]). Note that the range of the standard augmentation is a set of permutations which is closed under composition.

For an unsigned permutation π, $\pi(i)$ and $\pi(i+1)$ are said to be *successive* in π. For a signed permutation π', standardly augmented to an unsigned permutation π, $\pi(2i)$ and $\pi(2i+1)$ are said to be *successive* in π'.

A pair of successive elements in π which are also successive in σ is called an *adjacency*. A pair of successive elements in π which are not successive in σ is called a *breakpoint* in π with respect to σ. The number of breakpoints of π with respect to σ is denoted by $bp(\pi, \sigma)$. Trivially, $bp(\pi, \sigma) = bp(\sigma, \pi)$.

For example, let $\pi' = (1, -3, -2)$ be a signed permutation on three elements. It is rescaled to the 6-permutation $(1, 2, 6, 5, 4, 3)$, and it's standard augmentation is the 8-permutation $\pi = (0, 1, 2, 6, 5, 4, 3, 7)$. The only breakpoints in π with respect to the identity permutation σ are $(2, 6)$ and $(3, 7)$, thus $bp(\pi, \sigma) = 2$.

All graphs in this chapter are finite, and unless specifically indicated otherwise, undirected. They do not contain self loops, but may contain parallel edges: single, double or triple edges. For a set of $2n$ vertices, a *perfect matching* is a set of n edges, incident on every vertex. A *Hamiltonian cycle* is a set of $2n$ edges, forming one cycle passing through all the vertices.

5.2.2 Definitions

Let $V = \{v_i\}_{i=0}^{2n-1}$ be a set of $2n$ vertices. We call the perfect matching $M_b = \{(v_{2i}v_{2i-1})\}_{i=0}^{n-1}$ the *base matching* on V (subscripts for V are calculated modulo $2n$).

Let M_b be the base matching on the set V of $2n$ vertices. A perfect matching M on V is called a *Hamiltonian matching* with respect to M_b (or simply a Hamiltonian Matching) if $M_b \cup M$ forms a Hamiltonian cycle. For a Hamiltonian matching M, let $unsigned(M)$ be the permutation on $\{0, 1, \ldots, 2n-1\}$, denoting the order of appearance of the vertices in V along the cycle $M_b \cup M$, starting from v_0, and ending on v_{2n-1}. Clearly, $2i-1$ and $2i$ are consecutive in $unsigned(M)$, for each $0 < i \leq n$, hence, there exists a unique signed permutation denoted $signed(M)$ on $n-1$ elements, whose standard augmentation into an unsigned permutation on $2n$ elements is $unsigned(M)$. Furthermore, for every signed permutation π on $n-1$ elements, its standard augmentation into an unsigned permutation on $2n$ elements is π', we can define a Hamiltonian matching w $hmatch(\pi) = \{v_0v_{\pi'(1)}, v_{\pi'(2)}v_{\pi'(3)}, \ldots, v_{\pi'(2n-4)}v_{\pi'(2n-3)}, v_{\pi'(2n-2)}v_{2n-1}\}$ with respect to the base matching on $2n$ vertices.

Let π, σ be two signed permutations on $n-1$ elements. Each adjacency between them corresponds to an edge common to both $hmatch(\pi)$ and $hmatch(\sigma)$ and, on the other hand, each such common edge corresponds to such an adjacency.

Corollary 5.2.1 *Let π and σ be signed permutations on $n-1$ elements. Then $bp(\pi, \sigma) = n - |hmatch(\pi) \cap hmatch(\sigma)|$*

5.2.3 The Consensus of 3 Hamiltonian Matchings problem

Problem Definition

We define the Consensus of 3 Hamiltonian Matchings(*C3HM* for short) problem as follows:

Instance: Three Hamiltonian matchings M_0, M_1, M_2 with respect to the base matching M_b on a set V of $2n$ vertices, and an integer w_{target}. M_b, M_0, M_1, M_2 define a weight function w for any Hamiltonian matching M, setting $w(M) \equiv \sum_i |M \cap M_i|$.
Question: Does there exist a Hamiltonian matching $\widehat{M}$ with respect to M_b, s.t.

$$w(\widehat{M}) \geq w_{target}$$

We extend the weight function to edges, by defining $w(e) = |\{i|e \in M_i\}|$. Then for a Hamiltonian matching M, $w(M) = \sum_{e \in M} w(e)$.

Equivalence to the Median Problem

Proposition 5.2.2 *The instance $(V, M_b, M_0, M_1, M_2, w_{target})$ of the C3HM problem and the instance $(signed(M_0), signed(M_1), signed(M_2), 3|M_b| - w_{target})$ of the signed permutations median problem are equivalent.*

Proof: Follows directly from corollary 5.2.1. ■

Note that the transformation in the other direction, i.e., from signed permutations median to C3HM, is also immediate.

5.2.4 NP-completeness

Theorem 5.2.3 *C3HM is NP-complete.*

Proof: Membership in NP is trivial.

We prove NP-hardness by reduction from the Hamiltonian cycle problem, restricted to 3-regular graphs. This problem is well known to be NP-complete [Garey et al., 1976]. Let $G(N, A)$ be a simple 3-regular input graph, with $n = |N|$. We construct, in polynomial time, an instance $I = (V, M_b, M_0, M_1, M_2, w_{target})$ of C3HM, of linear size in n, s.t. I is a "yes" instance iff G admits a Hamiltonian path. To avoid confusion, we refer to G as a graph on the set N of *nodes* with the set A of *arcs*, reserving the terms *vertex* and *edge* for the graph constructed as a C3HM instance.

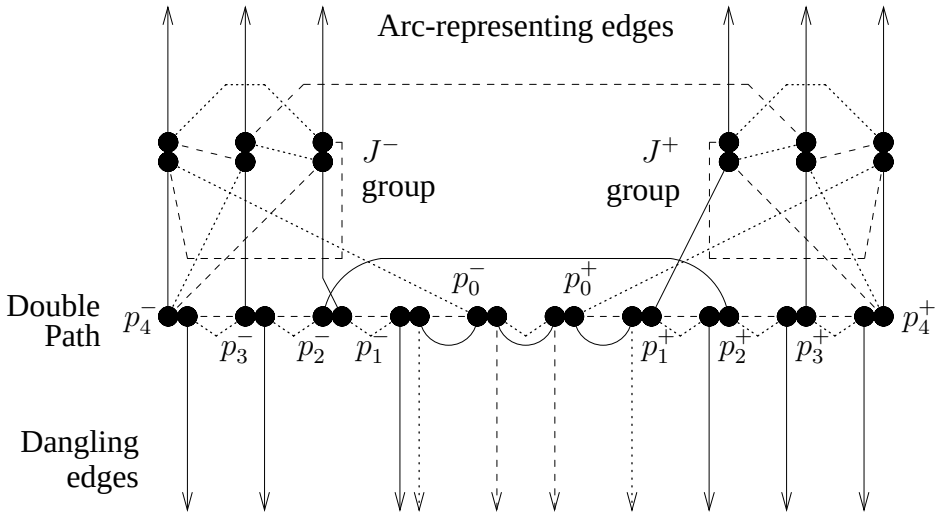


Figure 5.1: The whole node component. Each pair of tangent discs denotes a pair of vertices $u\widetilde{u}$ (matched by M_b). In such a pair $\widetilde{u}$ is the bottom disc (in the J groups) or the disc closer to the center (along the double path) Solid, dotted and dashed lines denote edges of M_0, M_1, and M_2 respectively. Arrows downwards denote dangling edges, while arrows upwards denote arc-representing edges.

We first give an overview of the reduction. We build a node component for each node of G. There are edges between these components, some of them correspond to the arcs of G. Our construction maps Hamiltonian cycles in G to Hamiltonian cycles $\widehat{M} \cup M_b$ in the constructed graph, with $\widehat{M}$ being a large-weight Hamiltonian matching. We can force specific properties of such Hamiltonian cycles in our construction, and urge them to pass through certain edges, by using double edges, with which $\widehat{M}$ must concur, in order to maximize its weight. In this way, we construct,

for each node, a component all of whose vertices must be visited *consecutively* by $\widehat{M} \cup M_b$, and the order in which the different node components are visited by $\widehat{M} \cup M_b$ corresponds to the order of the nodes along a Hamiltonian cycle in G. We make sure that a Hamiltonian cycle $\widehat{M} \cup M_b$ in our construction can visit two node components consecutively if and only if the two corresponding nodes in G are adjacent.

We now start describing our construction in detail. For each node $\nu \in N$, we build a node-component $C(\nu)$ of 32 vertices. See figure 5.1 for the outline of a full component. We will now describe the construction of the node-component. We explicitly specify the vertices of such a component, and some of the edges. The other edges in the constructed graph are either:

- *Arc-representing*, and will be explicitly specified in a later stage.
- *Dangling* edges, which will be constructed implicitly. When describing the node component, we only specify:
  - On which vertices are the dangling edges incident.
  - To which of the three matchings these edges belong.

The component $C(\nu)$ is composed of two (not fully symmetrical) halves, whose vertices and vertex-groups will be superscripted with $+$ and $-$, respectively.

The 32 vertices in $C(\nu)$ are divided into four groups:

- A half-path $P^+(\nu)$ of 10 vertices $\{p_i^+(\nu)\}_{i=0}^4$ and $\{\widetilde{p_i^+}(\nu)\}_{i=0}^4$.
- A half-path $P^-(\nu)$ of 10 vertices $\{p_i^-(\nu)\}_{i=0}^4$ and $\{\widetilde{p_i^-}(\nu)\}_{i=0}^4$.
- A junction $J^+(\nu)$ of 6 vertices $\{j_i^+(\nu)\}_{i=0}^2$ and $\{\widetilde{j_i^+}(\nu)\}_{i=0}^2$.
- A junction $J^-(\nu)$ of 6 vertices $\{j_i^-(\nu)\}_{i=0}^2$ and $\{\widetilde{j_i^-}(\nu)\}_{i=0}^2$.

Note that vertices in our construction come in pairs $(u(\nu), \widetilde{u}(\nu))$, where $u(\nu)$ and $\widetilde{u}(\nu)$ are always in the same component half and in the same group. Our base

matching M_b is simply the set of all edges $u(\nu)\widetilde{u}(\nu)$. Whenever the node ν or the $+/-$ superscript are clear from context, we omit them.

The edges in a component C include:

- Double (*path*) edges:
  - Between the component half paths : $\widetilde{p_0^+}\widetilde{p_0^-} \in M_0 \cap M_1$.
  - In each half path: $p_0\widetilde{p_1} \in M_0 \cap M_2$, and for each $i = 1, 2, 3$: $p_i\widetilde{p_{i+1}} \in M_1 \cap M_2$.

  We call the path in $M_b \cup \cup_i M_i$ between $p_4^+(\nu)$ and $p_4^-(\nu)$, using the path edges, the *double path* of ν.

- *Bypassing edges*: The edges $\widetilde{j_1}p_3 \in M_0$ and $\widetilde{j_0}p_0 \in M_1$ for each half, and the edges $\widetilde{j_2^+}p_1^+, \widetilde{j_2^-}\widetilde{p_2^-}, p_2^+p_2^- \in M_0$.

- *Junction* edges (see Figure 5.2):
  - The edges $p_4\widetilde{j_i} \in M_i$ for each $0 \le i \le 2$ and for each half.
  - The edges $j_2\widetilde{j_0}, j_0\widetilde{j_1} \in M_2$ and $j_1\widetilde{j_2} \in M_1$ for each half.
  - The edges $j_0j_2 \in M_1$ for each half, and the edge $j_1^+j_1^- \in M_2$.

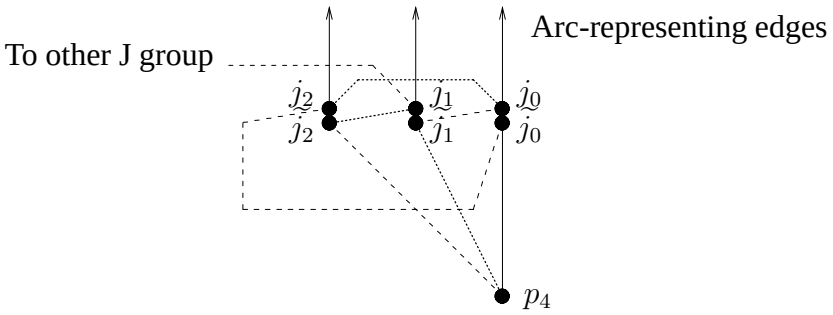


Figure 5.2: Zoom in on one J group. Only junction edges are drawn.

- *Dangling edges*: As explained, at this stage we only specify the incidence of a dangling edge, without explicitly specifying the edges.

- Dangling edges of M_0 are incident on $\widetilde{p_4}$, $\widetilde{p_3}$ for each half, and to $\widetilde{p_2^+}$ and $\widetilde{p_1^-}$.
- Dangling edges of M_1 are incident on $\widetilde{p_1}$ for each half.
- Dangling edges of M_2 are incident on $\widetilde{p_0}$ for each half.

We now describe the arc-representing edges. For each node $\nu \in N$, arbitrarily number its three arcs $a_0(\nu), a_1(\nu), a_2(\nu)$. For an arc $\nu\nu' = a_i(\nu) = a_{i'}(\nu') \in A$ add the edge $j_i^+(\nu)j_{i'}^-(\nu')$ to M_0. Note that the edge $j_{i'}^+(\nu')j_i^-(\nu)$ will also be added to M_0.

We will now describe some simple properties of our construction so far:

Property 5.2.4 *Each bypassing or dangling edge is incident on at least one vertex along the double path, which is not an endpoint of this path.*

Property 5.2.5 *For each matching M_i, let M_i' be the subset of M_i excluding all dangling edges Observe that each of $M_b \cup M_1'$ and $M_b \cup M_2'$ is a collection of $|N|$ (disjoint) paths, and each such path passes through all the vertices of some node component $C(\nu)$ (see Figure 5.3 and Figure 5.4).*

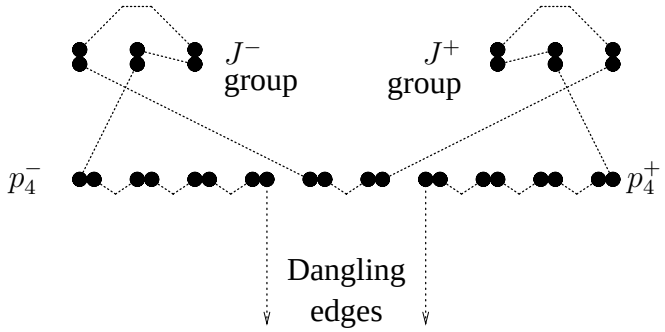


Figure 5.3: $M_b \cup M_1$ forms a path in each component.

Denote the paths described in property 5.2.5 $PATH_1(\nu)$ and $PATH_2(\nu)$, respectively. Both endpoints of such a path are vertices where dangling edges are incident on and both are along the double path of ν, one in $P^+(\nu)$ (called *head*) and the other in $P^-(\nu)$ (called *tail*)

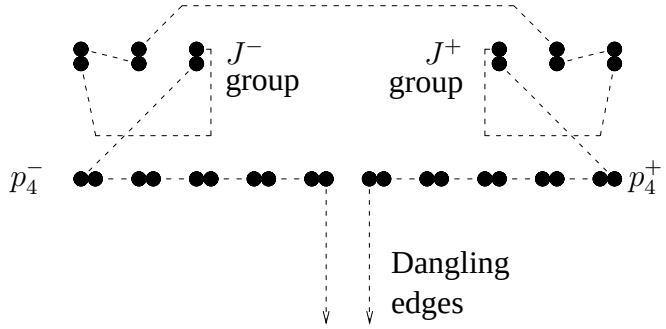


Figure 5.4: $M_b \cup M_2$ forms a path in each component.

Property 5.2.6 *$M_b \cup M_0'$ is a collection of $2|A|$ (disjoint) paths, each passing through one arc-representing edge $j_i^-(\nu)j_{i'}^+(\nu')$ (see Figure 5.5).*

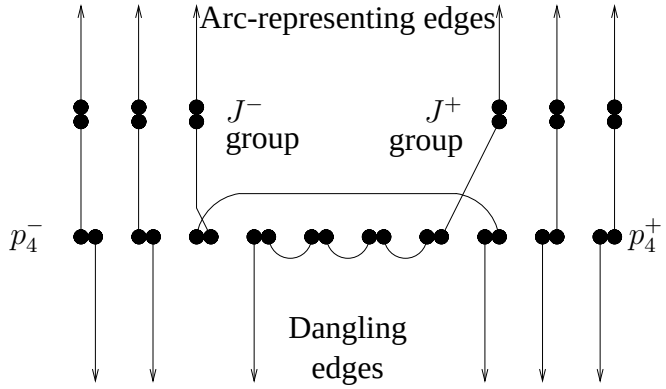


Figure 5.5: $M_b \cup M_0$ forms a path in through each arc-representing edge.

Denote the paths described in property 5.2.6 $PATH_0(\nu, \nu')$. This path has its endpoints on two vertices along the double paths of ν and ν', respectively, with a dangling edge incident on each of them. The lengths of such paths may vary between 4 and 8 vertex pairs.

We now describe how to connect the vertices on which dangling edges are incident, thus constructing the dangling edges themselves. Number the nodes $\nu_0, \nu_1, \ldots, \nu_{n-1}, \nu_n = \nu_0$ in an arbitrary order. For each $i = 1, 2$ and $k = 0, \ldots, n-1$, add to M_i a dangling edge connecting the tail of each $PATH_i(\nu_k)$

to the head of $PATH_i(\nu_{k+1})$.

Let $D(N, \vec{A})$ be a directed graph formed by replacing every arc $\nu\nu'$ of A by a pair of anti-parallel directed arcs $\nu\nu'$ and $\nu'\nu$. Clearly, D is Eulerian. Let T be an arbitrary directed Euler tour of D. For every two consecutive directed arcs $\nu\nu'$ and $\nu'\nu''$ in T, the paths $PATH_0(\nu\nu')$ and $PATH_0(\nu'\nu'')$ each have one endpoint in $C(\nu')$. Connect these two endpoints by a M_0 edge. All the dangling edges are constructed by this process, since every node ν' is visited exactly 3 times by T, with each of the six dangling edges endpoints in $C(\nu')$ visited once.

Note that no dangling edge thus constructed is parallel to any other edge in any M_i. Therefore, each dangling edge is single, and no double or triple edges are generated when determining the dangling edges.

We set w_{target} to be $25n$.

Clearly, the reduction is polynomial. We prove now the validity of our construction.

Claim 5.2.7 *I is an instance of C3HM.*

Proof: M_b is a perfect matching. For each $i = 0, 1, 2$, every vertex in V is incident to exactly one edge of each M_i, so also M_i is a perfect matching. The construction of the dangling edges connects all the paths $\{PATH_1(\nu)|\nu \in N\}$ into one Hamiltonian cycle, whose edges are exactly $M_b \cup M_1$. Similarly, $M_b \cup M_2$ is a Hamiltonian cycle. Also, the construction of the dangling edges connects all the paths $\{PATH_0(\nu\nu')|\nu\nu' \in A\}$ into one Hamiltonian cycle, whose edges are exactly $M_b \cup M_0$. Hence, M_i is a Hamiltonian matching for $i = 0, 1, 2$. ■

Claim 5.2.8 *If there is a Hamiltonian cycle in G, then I is a "yes" instance.*

Proof: Assume there exists a Hamiltonian cycle h in G. We construct a matching $\widehat{M}$ as follows:

- Include all 9 double path edges of each node component (4 of each half, and one connecting the two halves).

- Choose an arbitrary orientation for h. Let $\nu\nu' = a_i(\nu) = a_{i'}(\nu')$ be an arc of h in this orientation. For each such arc add to $\widehat{M}$ the following seven edges:
  - Edges in J^+ and P^+: $p_4^+(\nu)\widetilde{j_{i+1}^+}(\nu)$, $j_{i+1}^+(\nu)\widetilde{j_{i+2}^+}(\nu)$, $j_{i+2}^+(\nu)\widetilde{j_i^+}(\nu)$.
  - Edges in J^- and P^-: $p_4^-(\nu')\widetilde{j_{i'+1}^-}(\nu')$, $j_{i'+1}^-(\nu')\widetilde{j_{i'+2}^-}(\nu')$, $j_{i'+2}^-(\nu')\widetilde{j_{i'}^-}(\nu')$.
  - The edge $j_i^+(\nu)j_{i'}^-(\nu')$.

  Subscripts for junction vertices are calculated modulo 3. Compare Figure 5.6 and Figure 5.2.

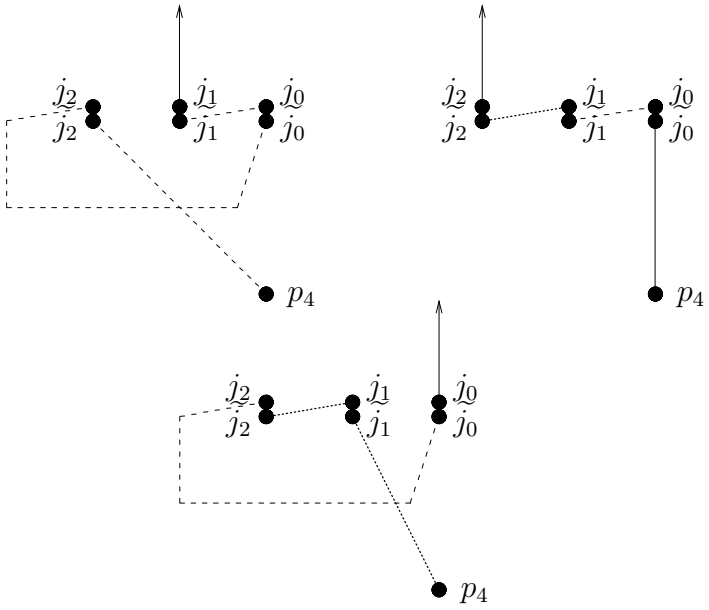


Figure 5.6: The three possible ways we use to connect the vertices of a junction group by a Hamiltonian matching.

Clearly, $\widehat{M}$ is a matching. Since h is Hamiltonian, $\widehat{M} \cup M_b$ is a Hamiltonian cycle. For the double edges in $\widehat{M}$, $w(e) = 2$. For the other edges in $\widehat{M}$, $w(e) = 1$. It follows that $w(\widehat{M}) = 2 \cdot 9n + 7n = 25n$ ■

Claim 5.2.9 *If I is a "yes" instance, then there is a Hamiltonian cycle in G.*

Proof: Let $\widehat{M}$ be a Hamiltonian matching in I, with $w(\widehat{M}) \geq 25n$. We call a Hamiltonian matching M in I *proper* if all the vertices of each node component appear consecutively along the cycle $M \cup M_b$. We now prove that $\widehat{M}$ is proper.

Since there are exactly $16n$ edges in $\widehat{M}$, and there are only $9n$ double edges in I, $\widehat{M}$ must contain all the double edges, and furthermore, for every other edge $e \in \widehat{M}$, $w(e) = 1$, i.e., $e \in M_i$ for some i. It follows that $\widehat{M} \cup M_b$ contains the double path of each ν. Furthermore, according to property 5.2.4, every bypassing or dangling edge is incident on an internal vertex of such a path, $\widehat{M}$ contains no bypassing edges, nor dangling edges (See Figure 5.7).

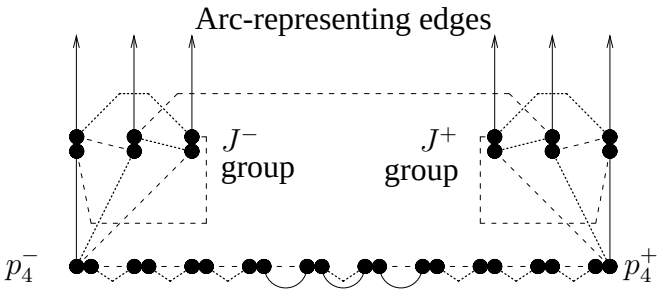


Figure 5.7: The component without the bypassing edges and the dangling edges. A proper Hamiltonian matching must pass through the six J^+ vertices consecutively, and immediately "after" (direction is arbitrary) the P vertices.

Fix some ν. $\widehat{M}$ must contain one edge incident on $p_4^+(\nu)$. Since this edge is a member of some Hamiltonian matching M_i, it is $p_4^+(\nu)\widetilde{j_i^+}(\nu)$, for some $0 \leq i \leq 2$ (see Figure 5.6). Since $\widehat{M}$ must contain no other edge incident on $p_4^+(\nu)$, the edges of the two other Hamiltonian matchings incident on $p_4^+(\nu)$ are not in $\widehat{M}$. Formally, for $i' \neq i$, $p_4^+(\nu)\widetilde{j_{i'}^+}(\nu) \notin \widehat{M}$. Now we can examine the vertex $\widetilde{j_{i'}^+}(\nu)$. There must be some non-bypassing edge in $\widehat{M}$ incident on it, and the only possible candidate is $j_{i'-1}^+(\nu)\widetilde{j_{i'}^+}(\nu)$ (see Figure 5.2), therefore $j_{i'-1}^+(\nu)\widetilde{j_{i'}^+}(\nu) \in \widehat{M}$.

Hence, the path through $J^+(\nu)$ $j_{i+2}, \widetilde{j_{i+2}}, j_{i+1}, \widetilde{j_{i+1}}, j_i^+, \widetilde{j_i}p_4$ is contained in $\widehat{M} \cup M_b$. The argument for a path passing through $J^-(\nu)$ is similar, proving that $\widehat{M}$ is proper.

Let $C(\nu), C(\nu')$ be two components visited consecutively by $\widehat{M} \cup M_b$. There must be an edge e connecting the two components, and since $\widehat{M} \cup M_b$ contains no

dangling edges, e must be an arc-representing edge (all other edges are incident on two vertices in the same component). According to our construction, there must be an arc in G between ν and ν'.

Let $h = \nu_0, \nu_1, \ldots, \nu_{n-1}, \nu_0$ be the order in which components are visited by $\widehat{M} \cup M_b$. It follows that h is a Hamiltonian cycle in G.

■

Claims 5.2.7, 5.2.8, and 5.2.9 complete the proof of Theorem 5.2.3. ■

Corollary 5.2.10 *The signed permutations median problem is NP-complete.*

Proof: Immediate from Theorem 5.2.3 and Proposition 5.2.2. ■

Corollary 5.2.11 *Finding a tree and corresponding signed permutations in internal nodes so as to minimize overall breakpoint distance along tree edges is NP-hard, both if the tree topology is known and if it is unknown.*

5.2.5 Unsigned Permutations

We map each unsigned permutation π on the elements $1, 2, \ldots, n-2$ to the Hamiltonian cycle on n vertices $hcycle(\pi) = v_0, v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n-2)}, v_{n-1}, v_0$. This is a 1-1 mapping, whose range is all the Hamiltonian cycles on n vertices passing through the *compulsory* edge $v_{n-1}v_0$. We call such cycles *image* cycles. For two image cycles, $hcycle(\pi)$ and $hcycle(\sigma)$, a non-compulsory edge in $hcycle(\pi) \cap hcycle(\sigma)$, maps to an adjacency between π and σ, and vice versa.

The unsigned permutations median problem formulates as the Consensus of 3 Hamiltonian Cycles (C3HC for short) problem:
Instance: Three image cycles C_0, C_1, C_2 on n vertices, and an integer w_{target}.
Question: Does there exist an image cycle $\widehat{C}$, s.t. $w(\widehat{C}) = \sum_i |\widehat{C} \cap C_i| \geq w_{target}$.

Theorem 5.2.12 *C3HC (and therefore also the unsigned permutations median problem) is NP-complete.*

Proof: (sketch): The proof is very similar to the proof of Theorem 5.2.3. We use a similar construction to reduce the problem of Hamiltonian cycle on 3-regular graphs passing through a specific edge (actually, the Hamiltonian path problem), to C3HC. The only differences are:

- We set $C_i = M_i \cup M_b$, thus replacing every edge $e \in M_b$ with a triple edge $e \in C_0 \cap C_1 \cap C_2$.
- We number the vertices choosing a triple edge to be the compulsory edge.
- We increase w_{target} by the total weight of the triple edges and set it to $25n + 3 \cdot 16n = 73n$.

Cardinality considerations of the weight of a consensus image cycle make sure that this cycle passes through all triple edges, and the rest of the proof is similar. ■

Proof:(Other option): We reduce signed permutations median to unsigned permutations median. Let $I' = (\pi'_0, \pi'_1, \pi'_2, d_{target})$ be a signed permutations median problem instance with each π'_i being a signed permutation on n elements. Let π_i be the $2n + 2$ element signed permutation obtained by the standard augmentation of π'_i. We claim that the unsigned permutations median problem instance $I = (\pi_0, \pi_1, \pi_2, d_{target})$ is a "yes" instance iff I' is.

For every two signed permutations σ'_1, σ'_2, whose standard augmentations are, respectively, σ_1, σ_2, $bp(\sigma_1, \sigma_2) = bp(\sigma'_1, \sigma'_2)$. Therefore it remains only to be shown that:

Claim 5.2.13 *If τ is an unsigned permutation satisfying $\sum_i bp(\tau, \pi_i) \leq d_{target}$, then there is also some unsigned permutation σ satisfying $\sum_i bp(\sigma, \pi_i) \leq d_{target}$, with a signed permutation σ' whose standard augmentation is σ.*

Proof: We prove this by induction on the number $p(\tau)$ of pairs $2i - 1, 2i$ of non-successive elements in τ.

Clearly, for $p(\tau) = 0$, $\sigma = \tau$ is a standard augmentation of some unsigned σ'.

For positive k, assume correctness for all permutations ρ with $p(\rho) < k$, and suppose $p(\tau) = k$. It suffices to find a permutation ρ satisfying $\sum_i bp(\rho, \pi_i) \leq$

$\sum_i bp(\tau, \pi_i)$, with $p(\rho) < k$. Let $C_0, C_1, C_2, w_{target}$ be the C3HC instance corresponding to I'. Define $w(e) = |\{i | e \in C_i\}|$. Note that for each vertex v:

$$\sum_u w(uv) = 6 \tag{5.1}$$

We call every edge $e = v_{2i-1}v_{2i}$ a *parity* edge. Note that every parity edge e has $w(e) = 3$.

Then there is a Hamiltonian cycle $C_\tau = v_0, v_{\tau_1}, \ldots, v_{\tau_{2n}}, v_{2n+1}$ with weight $w(C_\tau) \geq w_{target}$, and with $n + 1 - k$ parity edges. Fix some i for which the parity edge $v_{2i-1}v_{2i}$ is not contained in C_τ. $2i - 1$ and $2i$ are not successive in τ, i.e. $\tau_j = 2i - 1, \tau_{j'} = 2i$, and $|j - j'| > 1$. Let $x = v_{\tau_{j-1}}$, $y = v_{2i-1}$, $z = v_{\tau_{j+1}}$ and $x' = v_{\tau_{j'-1}}$, $y' = v_{2i}$, $z' = v_{\tau_{j'+1}}$. $w(yy') = 3$, so according to equation 5.1 $w(xy)+w(yz)+w(x'y')+w(y'z') \leq 6$, either $w(xy)+w(x'y') \leq 3$ or $w(yz)+w(y'z') \leq 3$. In the first case, define $C_\rho = (C_\tau \backslash \{xy, x'y'\}) \cup \{xx', yy'\}$ and in the latter define $C_\rho = (C_\tau \setminus \{yz, y'z'\}) \cup \{zz', yy'\}$. In either case, C_ρ is an image cycle, and $w(C_\rho) \geq w(C_\tau)$. The corresponding permutation ρ is as required. ■

Now it is clear that I is a "yes" instance iff I' is, completing the proof of theorem 5.2.12 ■

5.3 Approximations to the Median Problem

In this section we consider the median problem for signed permutations, counting breakpoints to measure genomic distance. Sankoff and Blanchette studied this problem and developed several efficient heuristics. This problem was shown to be NP-complete in the previous section and independently by Bryant [1998]. We provide polynomial approximation algorithms for the signed model, which guarantee an approximation ratio of $\frac{7}{6}$.

5.3.1 Preliminaries

All graphs in this chapter are finite, and unless specifically indicated otherwise, undirected. Self loops are not allowed, but parallel edges are: edges may be single, double or triple. For a set of $2n$ vertices, a *perfect matching* is a set of n edges, incident on every vertex. A *Hamiltonian cycle* is a set of $2n$ edges, forming one cycle passing through all the vertices.

Let d be a measure of genomic distance, namely an integer distance function on the space of (signed) permutations. The (signed) median problem is defined as follows:
Instance: Three (signed) permutations π_0, π_1, π_2, and an integer k
Question: Does there exist a (signed) permutation $\widehat{\pi}$ s.t. $\sum_i d(\pi_i, \widehat{\pi}) \leq k$?
Standard terminology of genome rearrangements can be found in Pe'er and Shamir [1998]. We omit these details, since this chapter deals only with an equivalent representation of this problem from a graph theoretic approach, which we now explain.

Let $V = \{v_i\}_{i=0}^{2n-1}$ be a set of $2n$ vertices. We call the perfect matching $M_b = \{(v_{2i}v_{2i-1})\}_{i=0}^{n-1}$ the *base matching* on V (subscripts for V are calculated modulo $2n$).

Let M_b be the base matching on the set V of $2n$ vertices. A perfect matching M on V is called a *Hamiltonian matching* (with respect to M_b), if $M_b \cup M$ forms a Hamiltonian cycle. For a Hamiltonian matching M, let $unsigned(M)$ be the permutation on $\{0, 1, \ldots, 2n-1\}$, denoting the order of appearance of the vertices in V along the cycle $M_b \cup M$, starting from v_0, and ending on v_{2n-1}. Clearly, $2i-1$ and $2i$ are consecutive in $unsigned(M)$, for each $0 < i \leq n$, hence, there exists a unique signed permutation denoted $signed(M)$ on $n-1$ elements, whose standard augmentation into an unsigned permutation on $2n$ elements is $unsigned(M)$. Furthermore, for every signed permutation π on $n-1$ elements, its standard augmentation into an unsigned permutation on $2n$ elements is π', and the edge set $hmatch(\pi) = \{v_0 v_{\pi'(1)}, v_{\pi'(2)}v_{\pi'(3)}, \ldots, v_{\pi'(2n-4)}v_{\pi'(2n-3)}, v_{\pi'(2n-2)}v_{2n-1}\}$ is a Hamiltonian matching with respect to the base matching on $2n$ vertices.

Let π, σ be two signed permutations on $n-1$ elements. Each adjacency be-

tween them corresponds to an edge common to both $hmatch(\pi)$ and $hmatch(\sigma)$ and, on the other hand, each such common edge corresponds to such an adjacency. Recall, that by 5.2.1, for signed permutations π and σ on $n-1$ elements, $bp(\pi,\sigma) = n - |hmatch(\pi) \cap hmatch(\sigma)|$

The problem of Consensus of 3 Hamiltonian Matchings(*C3HM* for short) is defined as follows:
Instance: Three Hamiltonian matchings M_0, M_1, M_2 with respect to the base matching M_b on a set V of $2n$ vertices, and an integer w_{target}. M_b, M_0, M_1, M_2 define a weight function w for any Hamiltonian matching M, setting $w(M) \equiv \sum_i |M \cap M_i|$.
Question: Does there exist a Hamiltonian matching $\widehat{M}$ with respect to M_b, s.t.

$$w(\widehat{M}) \geq w_{target}$$

We extend the weight function to edges, by defining $w(e) = |\{i|e \in M_i\}|$. Then for a Hamiltonian matching M:

$$w(M) = \sum_{e \in M} w(e) \tag{5.2}$$

Recall, that according to 5.2.2, The instance $(V, M_b, M_0, M_1, M_2, w_{target})$ of the C3HM problem and the instance $(signed(M_0), signed(M_1), signed(M_2), 3|M_b| - w_{target})$ of the signed permutations median problem are equivalent. Moreover, the optimization versions of these two problems are equivalent, and the mapping between their solutions guarantees that approximating one of the problems with a ratio R implies an R-approximation to the other as well.

For a C3HM instance, we call $G(V, M_b \cup M_0 \cup M_1 \cup M_2)$ the consensus graph, with $|V| = 2n$. W.l.o.g., we assume there are no triple edges in G: Otherwise, such an edge trivially belongs to an optimal consensus Hamiltonian matching, and can be dropped, contracting its incident vertex pairs to one pair. In such a case, we can solve the contracted instance, and the approximation ratio of our solution would only improve upon re-insertion of the triple edge.

Let $E^1(G)$ (respectively, $E^2(G)$) be the set of single (double) edges of G. Let $E^0(G)$ be the set of all complete graph edges not in G. Let $\bar{G}$ be the complete

graph on V, with edge weights $w : E(\bar{G}) \mapsto \{-\infty, 1, 2, 3\}$, defined as follows:

$$w(e) = \begin{cases} 3 & \text{if } e \in E^0 \\ 2 & \text{if } e \in E^1 \\ 1 & \text{if } e \in E^2 \\ -\infty & \text{if } e \in M_b \end{cases}$$

$(\bar{G}, w)$ can be considered an instance of the Traveling Salesperson Problem (TSP). Solving the original C3HM instance is equivalent to solving TSP on $(\bar{G}, w)$ [Sankoff and Blanchette, 1997, 1998], with the weight $-\infty$ replaced by a finite negative constant with large absolute value.

Let G_2 be the subgraph of G whose edge set is $E^2 \cup M_b$. Since each vertex has exactly one M_b edge, and at most one E^2 edge incident on it, G_2 is a collection of paths and cycles with alternating E_2-M_b edges. We partition E^2 edges to a set C of *cycle edges* and a set P of *path edges*, according to their role in G_2. The case of G_2 being one $2n$-long cycle is trivial to solve (E^2 is then an optimal solution) and we ignore it in the rest of the chapter. Assume G_2 is a collection of n_c cycles and n_p paths. The *length* of a cycle or path is the number of E^2 edges in it. Let l_c (l_p) denote the total length of all the cycles (paths, respectively). A pair of vertices connected by an edge of M_b is considered a path of length 0. All the cycles contain a total of l_c edges of M_b, while all the paths contain a total of $l_p + n_p$ edges of M_b. Observe that a Hamiltonian matching cannot include all the E^2 edges of a cycle, therefore

$$r \equiv |E^2| - n_c = l_p + l_c - n_c \tag{5.3}$$

is an upper bound on the number of E^2 edges used by a solution. Note that

$$n - r = n_p + n_c \tag{5.4}$$

5.3.2 A Simple Approximation for C3HM

Outline

We prove that the optimal median solution value w_{opt} satisfies $w_{opt} \geq 2n - r$. We also provide two simple approximation algorithms, indexed $\mathcal{A}$ and $\mathcal{B}$, with

performance guarantees of $2n - \frac{2r}{3}$ and $3n - 2r$, respectively. The approximation ratio of the algorithm $\mathcal{A}$ is bounded by $\frac{2n-\frac{2r}{3}}{2n-r}$, which is an increasing function of r, while the approximation ratio of the algorithm $\mathcal{B}$, is bounded by $\frac{3n-2r}{2n-r}$, which decreases whenever r increases. The threshold value of r, in which these two bound meet, is $r_t = \frac{3n}{4}$. Applying the both algorithms $\mathcal{A}$ and $\mathcal{B}$, choosing the better solution, guarantees an approximation ratio with guarantee $\min(2n - \frac{2r}{3}, 3n - 2r)$, applying algorithm $\mathcal{A}$ iff $r \leq r_t$. The worst approximation ratio guarantee we thus get is for $r = r_t$, and then

$$\frac{w_{approx}}{w_{opt}} \leq \frac{\min(2n - \frac{2r}{3}, 3n - 2r)}{2n - r} \leq \frac{3n - 2 \cdot \frac{3n}{4}}{2n - \frac{3n}{4}} = \frac{6}{5} \tag{5.5}$$

Bound on w_{opt}

Let M be an optimal solution. Since $M \cup M_b$ is a single cycle, for each cycle C_i in G_2, there is at least one E^2 edge not in M. Hence, $|M \cap (\cup_i C_i)| \leq l_c - n_c$, and $|M \cap E^2| \leq l_p + l_c - n_c = r$. It follows that:

$$w_{opt} \geq 1 \cdot r + 2 \cdot (n - r) = 2n - r \tag{5.6}$$

Approximation Algorithm $\mathcal{A}$

Let $d_i = |M_i \cap E^2|$. Then $d_0 + d_1 + d_2 = 2|E^2| = 2(r + n_c)$. Therefore there is some i for which $d_i \geq \frac{2(r+n_c)}{3}$. Our approximate solution is simply M_i. The value of that solution satisfies:

$$w^1_{approx} = \sum_{e \in M_i} w(e) = 1 \cdot d_i + 2 \cdot (n - d_i) =\leq 2n - \frac{2(r + n_c)}{3} \leq 2n - \frac{2r}{3} \tag{5.7}$$

Note that M_i is the input Hamiltonian matching, whose sum of disagreement distances from the two other input Hamiltonian matchings is minimal. Such an approximation for the median problem is known [Sankoff et al., 1996b] for any kind of metric, attaining an approximation ratio $\frac{3}{2}$, based on the triangle inequality. Eq [5.7] provides a better bound on the performance of this approximation, when the metric at hand is the number of breakpoints.

Approximation Algorithm $\mathcal{B}$

For each cycle C_i in G_2, arbitrarily choose some edge e_i, and denote $E' = E^2 \setminus \{e_i\}_{0 \leq i < n_c}$. The graph $H(V, E' \cup M_b)$ is a collection of disjoint paths. Concatenate these paths to a Hamiltonian cycle (which passes through all the edges of M_b), by connecting path endpoints arbitrarily. Let E'' be the set of those arbitrarily added edges. $E' \cup E''$ is our approximated solution. As $|E'| = r$ and $|E''| = n - r$, we conclude

$$w^2_{approx} = \sum_{e \in E' \cup E''} w(e) \leq 1 \cdot |E'| + 3 \cdot |E''| \leq 3n - 2r \tag{5.8}$$

5.3.3 Better Approximation

Outline

The bound in Eq [5.6] can be attained by an optimal solution if and only if such a solution employs the maximum number, r, of E^2 edges, and all the rest of the edges are E^1 edges. In this ideal scenario:

- All path edges are used.
- All cycle edges are used, except for one edge per cycle, breaking it into a path.
- The paths and broken cycles are chained by E^1 edges.

In such a case none of the paths is broken, and every cycle is broken only once. Whenever the ideal bound in Eq [5.6] is not attained by a solution, the discrepancy between the solution and the ideal bound can be attributed to deviations from the ideal scenario, namely, to edges not of the above specified types, called *non-ideal* edges. We call such a discrepancy *waste*, and charge non-ideal edges for this waste.

In this section we formalize these notions, distinguishing between several kinds of waste, according to the non-ideal edges charged for it. We devise a better approximation algorithm, which tries to conform with the ideal scenario as much

as possible, attempting to minimize the usage of non-ideal edges, thus reducing the waste. In particular, we try to chain paths and (broken) cycles by E^1 edges whenever possible. More specifically:

- We use E^1 edges which comprise a matching on a set of cycles and connected components of paths These edges chain broken cycles and paths, without any implied waste.
- We greedily choose E^1 edges which are not charged for more waste.

Furthermore, we observe that sometimes waste is inevitable. We characterize and quantify some of the inevitable waste, thus obtaining a better bound for the optimal solution. The new algorithm and the new bound guarantee a $\frac{7}{6}$ approximation ratio.

Some More Definitions

Each vertex v in $V(G) = V(G_2)$ is either a cycle vertex in G_2, or an internal path vertex, or a path endpoint. In these cases, we say that v is a *c-vertex*, an *i-vertex*, or a *p-vertex*, respectively. Let I be the set of all i-vertices and p-vertices, and let $M_b[I]$ be the set of M_b along paths of G_2. We distinguish between six categories of edges in $E^0 \cup E^1 \cup E^2$ according to the vertices they are incident on: E_{cc}, E_{ii}, E_{pp}, E_{ci}, E_{cp} and E_{ip}. For $i = 0, 1, 2$, define $E^i_{cc} = E^i \cap E_{cc}$, and similarly for the rest of the edge categories. We define the subgraph $G_{pp}(I, E^2_{ii} \cup E^1_{pp} \cup M_b[I])$ of G, including all the paths of G_2, and all the single edges of G connecting their endpoints (see figure 5.8, top-middle).

Construct a bipartite graph $H_{cp}(A \cup B, F)$, with $A = \{a_v | \text{p-vertex } v\}$ and $B = \{b_i | \text{cycle } C_i\}$, and for each $uv \in E^1_{cp}$, with $u \in C_i$, include in F its *representative* $a_v b_i$ (see figure 5.8, bottom-right). Define also the graph $H_{cc}(A \cup B, F \cup F_{cc})$, with F_{cc} being the set of all representatives $b_i b_j$ of edges $uv \in E^1_{cc}$, with $u \in C_i$ and $v \in C_j$. Define $H'_{cp}(A' \cup B, F'_{cp})$, with a vertex a'_S for each connected component S in G_{pp}. Set F'_{cp} to include all representatives $a'_S b_i$ of edges $uv \in E^1_{cp}$, for which $u \in S$, $v \in C_i$. Let M_{cc}, M_{cp},M'_{cp} be maximum matchings of H_{cc}, H_{cp}, and H'_{cp}

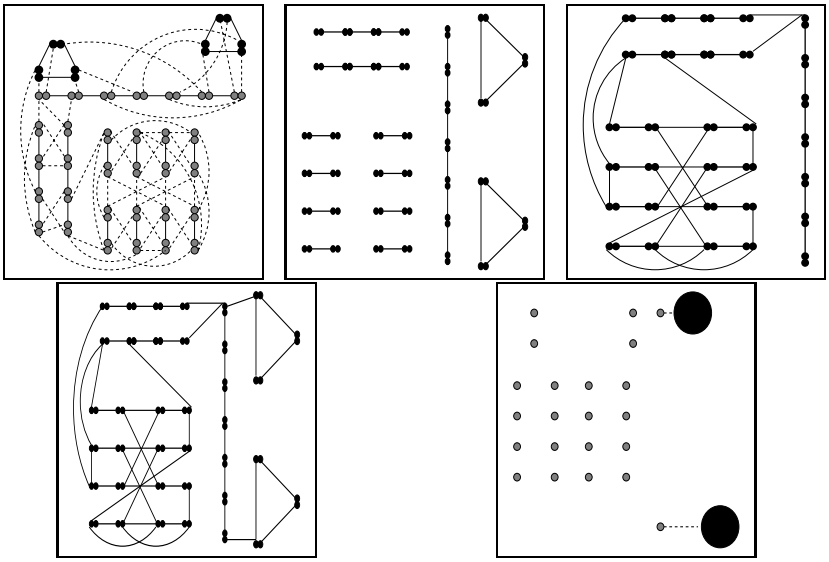


Figure 5.8: This is a 3HMM instance graph which we shall use for our examples (top left). Solid lines denote double (E^2) edgexs. Dashed lines denote single (E^1) edges. E^0 edges are omitted. Instead of drawing M_b edges, each pair of adjacent circles denotes a pair of M_b-matched vertices. The c-vertices appear in bold. Also shown in the top row are G_2 (middle) and G_{pp} (right) of this example. On the bottom left G_{pp} is depicted, along with cycle edges and E^1_{cp} edges (in our example, $E^1_{cc} = \emptyset$). H_{cp} of our example appears on the bottom right. The large nodes denote the cycles C_i. Since in our example $E^1_{cc} = \emptyset$, this is also H_{cc}. Furthermore, since this graph contains only two edges with no common endpoint, $M_{cc} = M_{cp} = H_{cp}$.

respectively (see figure 5.8, bottom-right). Surely, $|M_{cc}| \geq |M_{cp}| \geq |M'_{cp}|$. We use M_{cp} and M'_{cp} only for analyzing performance, while M_{cc} is actually computed by our algorithm.

For a Hamiltonian matching M, define the *waste* of M as

$$waste(M) = w(M) - (2n - r)$$

Substituting Eq [5.2] and Eq [5.3], we obtain:

$$waste(M) = |E^2 \setminus M| - n_c + |E^0 \cap M| \tag{5.9}$$

Let the *i-waste* of a Hamiltonian matching M be

$$iwaste(M) = 2|M \cap (E^0_{ii} \cup E^1_{ii})| + |M \cap (E_{ci} \cup E_{ip})|$$

Claim 5.3.1 *A Hamiltonian matching M with $i\text{-}waste(M) = 2x$, uses no more than $l_p - x$ path edges.*

Proof: Define $W_I = \{\text{i-vertex } v | \exists e(v) = uv \in (E^0 \cup E^1) \cap M\}$, and observe that $iwaste(M) = |W_I|$. Each $v \in W_I$ has a path edge $e_2(v) \in E^2_{ii}$ incident on it. Since $e_2(v) \neq e(v)$, and they both share a vertex, surely $e_2(v) \notin M$. Therefore:

$$|M \cap E^2_{ii}| = |E^2_{ii}| - |E^2_{ii} \setminus M| \leq l_p - \{e_2(v) | v \in W_I\} \leq l_p - \frac{|W_I|}{2} = l_P - x$$

■

Let the *c-waste* of a Hamiltonian matching M be

$$cwaste(M) = 2|M \cap (E^1_{cc} \cup E^0_{cc})| + |M \cap (E_{ci} \cup E_{cp})| - 2n_c$$

Claim 5.3.2 *A Hamiltonian matching M with $c\text{-}waste(M) = 2y$, uses no more than $l_c - y - n_c$ cycle edges.*

Proof: Define $W_C = \{\text{c-vertex } v | \exists e(v) = uv \in (E^0 \cup E^1) \cap M\}$, and observe that $cwaste(M) = |W_C| - 2n_c$. Each $v \in W_C$ has a cycle edge $e_2(v) \in E^2_{cc}$

incident on it. Since $e_2(v) \neq e(v)$, and they both share a vertex, surely $e_2(v) \notin M$. Therefore:

$$|M \cap E^2_{cc}| = |E^2_{cc}| - |E^2_{cc} \setminus M| \leq l_c - \{e_2(v) | v \in W_C\} \leq l_c - \frac{|W_C| - 2n_c}{2} = l_c - y - n_c$$

■

Let the *o-waste* of a Hamiltonian matching M be $owaste(M) = 2|M \cap E^0|$.

Eq [5.9], claim 5.3.1, and claim 5.3.2 imply:

Corollary 5.3.3 $2waste(M) = cwaste(M) + owaste(M) + iwaste(M)$

Algorithm $\mathcal{C}$

For a set X of edges, and an edge e, we define the *neighborhood* of e in X, $N_x(e)$, to be the set of all edges in X sharing a vertex with e. Algorithm $\mathcal{C}$ is as follows:

1. Initialize $M, \tilde{M} \leftarrow \emptyset$.
2. Construct H_{cc} and find a maximum matching M_{cc}, in it. For each edge xy in M_{cc}, if both x,y represent cycles, add to $\tilde{M}$ an edge incident on vertices of these cycles, respectively. Otherwise, x represents a cycle C_x, while y represents a vertex v_y in G. In this case, add to $\tilde{M}$ an edge incident on v_y and on a vertex of C_x (see figure 5.9, top-left).
3. Find a forest $F \subseteq E^2_{ii} \cup E^1_{pp}$, s.t. $F \cup M_b$ is a maximum spanning forest of G_{pp}. (see figure 5.9, top-middle).
4. Set $F \leftarrow F \setminus \cup_{e \in \tilde{M}} N_F(e)$ (see figure 5.9, top-right).
5. While $F \neq \emptyset$ do:
   (a) Choose a leaf u of F, with the edge $uv \in F$.
   (b) Add uv to M.
   (c) Set $F \leftarrow F \setminus N_F(uv)$ (see figure 5.9, middle row).
6. Add $\tilde{M}$ to M (see figure 5.9, bottom-left).

7. For each cycle C_i, add to M all the cycle edges of C_i, except one edge. If C_i is matched by M_{cc}, then choose the omitted edge to be incident on the vertex in $M \cap C_i$ (see figure 5.9, bottom-middle).

8. Add arbitrary edges to M to complete a Hamiltonian matching (see figure 5.9, bottom-right).

Correctness

Claim 5.3.4 *Algorithm $\mathcal{C}$ produces a Hamiltonian matching*

Proof: It suffices to show that until step 8, M satisfies:

1. M is a matching.

2. $M \cup M_b$ includes no cycles.

- **Edges added in step 5b**:
  Denote the set of all of the edges added to M in step 5b by M_{5b}. Since $M_{5b} \subseteq F$ and $F \cup M_b$ is a forest in G_{pp}, $M_{5b} \cup M_b$ is guaranteed not to form a cycle with path edges, nor with cycle edges (which are never connected to G_{pp} vertices). Furthermore, step 5c makes sure no added edge is incident on a vertex of a previously chosen edge, thus M_{5b} is a matching.

- **Edges added in step 6**:
  Edges added in step 6 connect p-vertices to c-vertices, and until step 7 no cycle edges have been considered. Hence, also adding $\tilde{M} \subseteq E^1_{cc} \cup E^1_{cp}$ to M (step 6) does not create cycles in $M \cup M_b$. Since no c-vertices were considered till step 6, and $\tilde{M}$ is a matching, no degree 2 c-vertices are created by this step. Also step 4 makes sure that no G_{pp} vertices receive degree 2 by adding $\tilde{M}$ to M.

- **Edges added in step 7**:
  Each cycle C_i is connected to the rest of $M \cup M_b$ by at most one edge, added

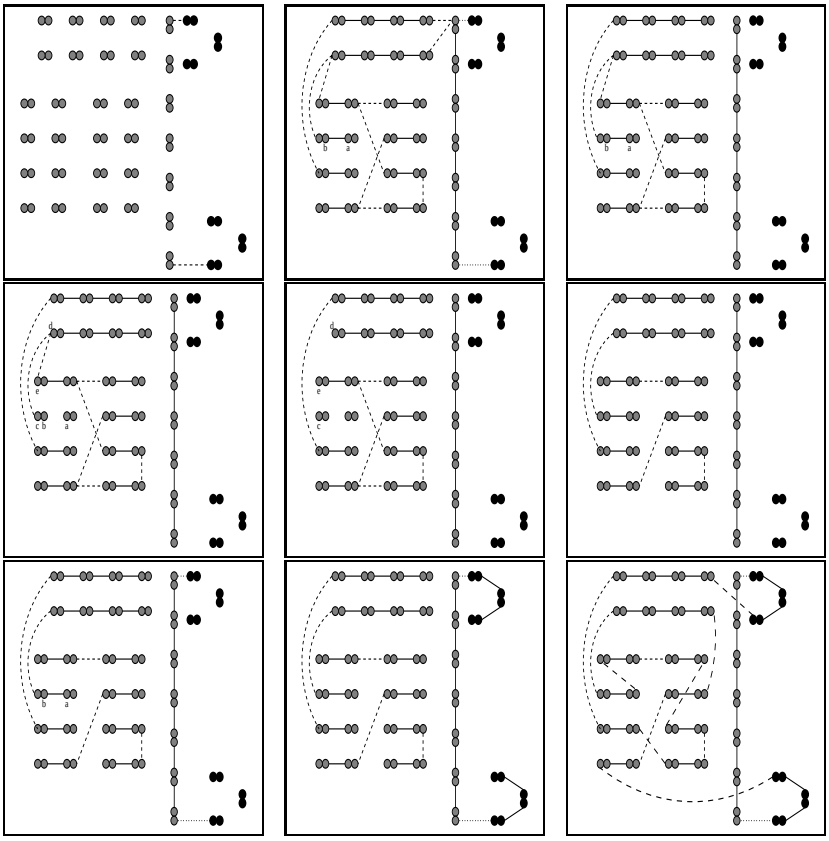


Figure 5.9: Top-left: $\tilde{M}$ in our example. Top-middle: F, when first constructed. $\tilde{M}$ edges are dotted. ab is an arbitrary edge, indexed for future reference. Top-right: F, after discarding the edges and neighboring edges of $\tilde{M}$ (after step 4). Middle-left: F, after the first loop iteration, choosing the edge ab to M. The vertices c, d, and e are indexed for future reference. Middle: F, after the second loop iteration, choosing the edge cd to M, discarding the edge de. Middle-right: M, after all the loop iterations (end of step 5). All path edges are in M. Bottom-left: M, after adding $\tilde{M}$ (end of step 6). Bottom-middle: M, after adding cycle edges (end of step 7). Bottom-right: The final M. The edges added in Step 8 appear in longer dashes line.

to $\tilde{M}$ in step 2, and merged into M in step 6. Therefore, the path of C_i edges added in step 7 creates no cycles in $M \cup M_b$. C_i has at most one edge e sharing a vertex with an edge in $M \setminus C_i$, and e is never added to M, thus step 7 does not create any vertices of degree 2 in M,

■

A Better Bound on the Optimal Solution

Denote by F_0 the forest F when it is first constructed. Let k be the number of connected components in $F_0 \cup M_b$. Let $m = |M_{cc}|$, $m' = |M'_{cp}|$.

Claim 5.3.5 *If $k > 2m'$, then $waste(M) \geq \frac{k}{2} - \frac{3m'}{4}$ for any Hamiltonian matching M:*

Proof: Let M be a Hamiltonian matching, and let the matching M' be a maximum cardinality subset of $E^1_{cp} \cap M$, such that no two edges of M' are incident on vertices of the same connected component of $F_0 \cup M_b$, nor of the same cycle in G_2. M' is a matching, and each of its edges has a unique representative in H'_{cp}. These representatives compose a matching of equal size in H'_{cp}, therefore $|M'| \leq m'$. Let $\mathcal{P}^-$ be the set of all connected components in $F_0 \cup M_b$, unmatched by M'. Obviously, $|\mathcal{P}^-| = k - |M'|$. Let U be the set of all vertices of connected components in $\mathcal{P}^-$, with an edge of M incident on them. The Hamiltonian cycle $M \cup M_b$ must enter and leave each component in $\mathcal{P}^-$ through a U vertex, thus:

$$|U| \geq 2|\mathcal{P}^-| \geq 2k - 2m'$$

Denote the edge of M incident on u by uv_u. Let $\mathcal{C}^+$ be the set of all G_2 cycles matched by M', and let $W \subseteq U$ be a maximal set of vertices x, with each v_x being in a distinct cycle in $\mathcal{C}^+$. Note that none of the vertices in U may be connected by edges of M to cycles outside $\mathcal{C}^+$.

$$|W| \leq |\mathcal{C}^+| = |M'| \leq m'$$

therefore $|U \setminus W| \geq 2k - 3m'$. We charge every $u \in U \setminus W$ for some increase in $waste(M)$, as follows:

- If $uv_u \in E^0$, then $owaste(M)$ increases by 1 on account of u. We charge u for this increase.

- If either u or v_u is an i-vertex, then $iwaste(M)$ increases by at least 1 on the account of uv_u, charging u for half of this increase.

- Otherwise, $uv_u \in E^1_{cp}$. v_u must be on a G_2 cycle C_i in $\mathcal{C}^+$, or else adding uv_u to M' contradicts its maximality. Every such C_i has already one vertex with an (uncharged) M edge incident to it, the edge of M' connected to C_i. Furthermore, C_i has another vertex with an uncharged edge M edge incident to it, connecting it to a vertex of W, otherwise u can be added to W, contradicting its maximality. Hence, if d_i denotes the number of C_i vertices with M edges incident on them, then the sum $\sum_i (d_i - 2)$ increases by 1 on account of v. This sum equals $cwaste(M)$, and we charge u for this increase.

To conclude, u is charged for increasing $waste(M)$ by at least $\frac{1}{4}$, and no increase in $waste(M)$ is charged for more than once, hence:

$$waste(M) \geq \frac{|U \setminus W|}{4} \geq \frac{k}{2} - \frac{3m'}{4}$$

■

Performance Analysis of Algorithm $\mathcal{C}$

Since $F_0 \cup M_b$ is a maximum spanning forest, F_0 contains all path edges. For a path edge e, always $N_F(e) \subseteq \{e\}$. Therefore, no such edge is being discarded in step 5c or in step 4, prior to adding it to M.

Connecting a total of n_p paths, F_0 contains also exactly $n_p - k$ edges of E^1_{pp}. No more than $2m$ edges are discarded from F in step 4. Since at most two edges are discarded from F in step 5c, for every edge added to M in step 5b, the total number of E^1_{pp} edges in M is at least $m + \lceil \frac{n_p - k - 2m}{3} \rceil$. M further contains exactly

r edges of E^2, therefore:

$$\begin{aligned} w(M) &\leq 1 \cdot r + 2 \cdot (m + \lceil \frac{n_p - k - 2m}{3} \rceil) + 3 \cdot (n - r - (m + \lceil \frac{n_p - k - 2m}{3} \rceil)) \\ &\leq 3n - 2r - m - \frac{n_p - k - 2m}{3} \\ &= 3n - 2r - \frac{n - r - n_c - k + m}{3} \\ &= \frac{8n}{3} - \frac{5r}{3} + \frac{n_c + k - m}{3} \end{aligned}$$

This performance guarantee gives an approximation ratio which increases whenever r decreases, meeting the performance guarantee of Eq [5.7] at the threshold r_t value for r, for which

$$\begin{aligned} \frac{8n}{3} - \frac{5r_t}{3} + \frac{n_c + k - m}{3} &= 2n - \frac{2r_t}{3} - \frac{2n_c}{3} \\ \frac{2n}{3} + n_c + \frac{(k - m)}{3} &= r_t \end{aligned}$$

We apply both algorithms $\mathcal{A}$, and $\mathcal{C}$, choosing the best solution found. This guarantees an approximation ratio of $2n - \frac{2r_t}{3} - \frac{2n_c}{3}$. We distinguish two cases:

case 1: If $n_c + m - k \geq 0$, then:

$$\begin{aligned} \frac{w_{approx}}{w_{opt}} &\leq \frac{2n - \frac{2r_t}{3} - \frac{2n_c}{3}}{2n - r_t} \\ &= \frac{2n - \frac{4n}{9} - \frac{2n_c}{3} + \frac{2(m-k)}{9} - \frac{2n_c}{3}}{2n - \frac{2n}{3} - n_c + \frac{(m-k)}{3}} \\ &= \frac{28n - 24n_c + 4(m-k)}{24n - 18n_c + 6(m-k)} \\ &\leq \frac{28n - 24n_c + 4(m-k) + 3n_c + 3(m-k)}{24n - 18n_c + 6(m-k)} = \frac{7}{6} \end{aligned}$$

case 2: If $k > n_c + m \geq 2m$, then the bound of Claim 5.3.5 applies, in particular:

$$w_{opt} \geq 2n - r_t + \frac{k}{2} - \frac{3m'}{4} \geq 2n - r_t + \frac{k - m - n_c}{7}$$

$$\begin{aligned}\frac{w_{approx}}{w_{opt}} &\leq \frac{2n - \frac{2r_t}{3} - \frac{2n_c}{3}}{2n - r_t + \frac{k-m-n_c}{7}} \\ &= \frac{2n - \frac{4n}{9} - \frac{2n_c}{3} + \frac{2(m-k)}{9} - \frac{2n_c}{3}}{2n - \frac{2n}{3} - n_c + \frac{(m-k)}{3} - \frac{m-k}{7} - \frac{n_c}{7}} \\ &= \frac{98n - 84n_c + 14(m-k)}{84n - 72n_c + 12(m-k)} = \frac{7}{6}\end{aligned}$$

5.3.4 Extension to Four Taxa

We shall now extend this approximation to handle trees with four leaves. We shall use the algorithm of section 5.3.3 as a black box, using only the fact that it is a $\frac{7}{6}$ approximation for the 3-leaves minimal Steiner tree problem, assuming nothing on the norm space at hand.

More formally, consider a norm $\mathcal{D}$ over a space $\mathcal{S}$. For triplet of points $\tau = (a, b, c), a, b, c \in \mathcal{S}$, their distance to a point $x \in \mathcal{S}$ is $d(x, \tau) = \mathcal{D}(a, x) + \mathcal{D}(b, x) + \mathcal{D}(c, x)$. Define the *median* of a triplet $\tau \in \mathcal{S}^3$ to be $med(\tau) \equiv argmin_x d(x, \tau)$. The *Steiner tree* on a set $S \in \mathcal{S}$ of points calls for finding a tree $T = (V, E)$ whose leaf set is S, minimizing $\sum_{uv \in E} \mathcal{D}(u, v)$. Finding the median is a restriction of the Steiner tree problem to only three input points. In the sequel we write uv instead of $\mathcal{D}(u, v)$, for short.

Suppose there exists a polynomial algorithm which given $\tau \in \mathcal{S}^3$ finds a point $\widehat{med}(\tau)$ such that $d(\widehat{med}, \tau) \leq \frac{7}{6} d(med(\tau), \tau)$. We shall devise an algorithm, which given four points $Q = \{a, b, d'b'\}$ finds a Steiner tree of weight at most $\frac{11}{8}$ the optimal tree.

Define the triplet τ_x to be the triplet of all four elements a, b, d', b' except x. We apply the median approximation algorithm on some each τ_x, obtaining $\hat{m_x} = \widehat{med}(\tau_x)$. For each $x, y \in Q$, we now define the weight of the 4-taxa tree T_{xy}, formed by taking m_x as median of τ_x, and connecting x to the leaf $y \neq x$:

$$w(T_{xy}) = xy + d(\hat{m_x}, \tau_x) \tag{5.10}$$

Our approximated tree will simply be T_{xy} for which $w(T_{xy})$ is minimal. Let z and w be the elements of $Q \setminus \{x, y\}$, and let T^*_{xy} be the optimal tree with topology

separating x, y from z, w, and let m_{xy} and m_{zw} be its internal vertices connected to these pairs, respectively. We observe that:

$$\begin{aligned} w(T_{xy}) &= xy + d(\hat{m_x}, \tau_x) \\ &\leq xm_{xy} + ym_{xy} + \frac{7}{6}d(med(\tau_x), \tau_x) \\ &\leq xm_{xy} + ym_{xy} + \frac{7}{6}d(m_{zw}, \tau_x) \\ &\leq xm_{xy} + ym_{xy} + \frac{7}{6}zm_{zw} + \frac{7}{6}wm_{zw} + \frac{7}{6}ym_{zw} \\ &\leq xm_{xy} + ym_{xy} + \frac{7}{6}zm_{zw} + \frac{7}{6}wm_{zw} + \frac{7}{6}ym_{yz} + \frac{7}{6}m_{zw}m_{yz} \\ &\leq xm_{xy} + \frac{13}{6}ym_{xy} + \frac{7}{6}zm_{zw} + \frac{7}{6}wm_{zw} + \frac{7}{6}m_{zw}m_{xy} \end{aligned}$$

But:

$$\min\{w(T_{xy}), w(T_{yx})\} \leq \frac{19}{12}(ym_{xy} + xm_{xy}) + \frac{7}{6}(zm_{zw} + wm_{zw}) + \frac{7}{6}m_{zw}m_{xy} \tag{5.11}$$

obtaining, for $W_{xy} = \{w(T_{xy}), w(T_{yx}), w(T_{zw}), w(T_{wz})\}$, that:

$$\begin{aligned} \min W_{xy} &\leq \frac{33}{24}(ym_{xy} + xm_{xy} + zm_{zw} + wm_{zw}) + \frac{7}{6}m_{zw}m_{xy} \\ &\leq \frac{11}{8}(ym_{xy} + xm_{xy} + zm_{zw} + wm_{zw} + m_{zw}m_{xy}) \\ &= \frac{11}{8}w(T^*_{xy}) \end{aligned}$$

Finally,

$$w_{approx} = \min_{xy} T_{xy} \leq \frac{11}{8} = min_{xy} w(T^*_{xy}) = \frac{11}{8}w_{opt} \tag{5.12}$$

proving the approximation ratio.

Bibliography

J. Adachi. *Modeling of Molecular Evolution and Maximum Likelihood Inference of Molecular Phylogeny*. PhD thesis, The Graduate University for Advanced Studies, Hayama., 1995.

J. Adachi and M. Hasegawa. Model of amino acid substitution in proteins encoded by mitochondrial DNA. *J. Mol. Evol.*, 1996.

L. M. Adleman. Location sensitive sequecing of DNA. Technical report, University of Southern California, 1998.

T. Anantharaman and B. Mishra. Genomics via optical mapping (i): Probabilistic analysis of optical mapping models. Technical Report TR1998-770, Computer Science Department, New York University, 1998.

T. S. Anantharaman, B. Mishra, and D. C. Schwartz. Genomics via optical mapping ii: Ordered restriction maps. *Journal of Computational Biology*, 4(2):91–118, 1997.

B. Apsvall, M. F. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8 (3):121–123, 1979.

R. Arratia, D. Martin, G. Reinert, and M. S. Waterman. Poisson process approximation for sequence repeats, and sequencing by hybridization. *Journal of Computational Biology*, 3(3):425–463, 1997.

V. Bafna and P. Pevzner. Sorting permutations by transpositions. In *Proceedings of*

the 6th Annual Symposium on Discrete Algorithms, pages 614–623. ACM Press, January 1995.

W. Bains and G. C. Smith. A novel method for nucleic acid sequence determination. *J. Theor. Biology*, 135:303–307, 1988.

G. Bejerano and G. Yona. Modeling protein families using probabilistic suffix trees. In S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 15–24, Lyon, France, 1999. ACM Press.

A. Ben-Dor, I. Pe'er, R. Shamir, and R. Sharan. On the complexity of positional sequencing by hybridization. In *Proceedings of the Tenth International Conference on Combinatorial Pattern Matching (CPM'99)*, pages 88–100, New York, 1999. ACM Press.

A. Ben-Dor, I. Pe'er, R. Shamir, and R. Sharan. On the complexity of positional sequencing by hybridization. *Journal of Computational Biology*, 8(4):361–371, 2001a.

A. Ben-Dor, I. Pe'er, R. Shamir, and R. Sharan. On the complexity of positional sequencing by hybridization. Technical Report TR01-054, Electronic Colloquium on Computational Complexity (ECCC), 2001b.

P. Berman and S. Hannenhalli. Fast sorting by reversal. In Daniel S. Hirschberg and Eugene W. Myers, editors, *Combinatorial Pattern Matching, 7th Annual Symposium*, volume 1075 of *Lecture Notes in Computer Science*, pages 168–185, Laguna Beach, California, 10-12 June 1996. Springer. ISBN ISBN 3-540-61258-0.

S. D. Broude, T. Sano, C. S. Smith, and C. R. Cantor. Enhanced DNA sequencing by hybridization. *Proc. Nat. Acad. Sci. USA*, 91:3072–3076, 1994.

D. Bryant. The complexity of the breakpoint median problem. Technical Report CRM-2579, Centre de recherches mathématiques, Université de Montréal, 1998.

L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58:171–176, 1996.

W. Cai, J. Jing, B. Irvin, L. Ohler, E. Rose, H. Shizuya, U. J. Kim, M. Simon, T. Anantharaman T, B. Mishra, and D. C. Schwartz. High-resolution restriction maps of bacterial artificial chromosomes constructed by optical mapping. *Proceedings of the National Academy Science U.S.A*, 95(7):3390–3395, 1998.

A. Caprara. Formulations of complexity of multiple sorting by reversals. Technical Report OR-97-15, University of Bologna, Bologna, Italy, 1997a.

A. Caprara. Sorting by reversals is difficult. In *Proceedings of the First International Conference on Computational Molecular Biology*, pages 75–83, New York, January19–22 1997b. ACM Press. ISBN 0-89791-882-7.

M. Cargill, D. Altshuler, J. Ireland, P. Sklar, K. Ardlie, N. Patil, C. R. Lane, E. P. Lim, N. Kalyanaraman, J. Nemesh, L. Ziaugra, L. Friedland, A. Rolfe, J. Warrington, R. Lipshutz, G. Q. Daley, and E. S. Lander. Characterization of single-nucleotide polymorphisms in coding regions of human genes. *Nature Genetics*, 22:231–238, 1999.

V. Dančík and Michael S. Waterman. Simple maximum likelihood methods for the optical mapping problem. In *Proceedings of the Workshop on Genome Informatics (GIW '97)*, 1997.

V. Dančík, S. Hannenhalli, and S. Muthukrishnan. Hardness of flip-cut problems from optical mapping. *Journal of Computational Biology*, 4:119–125, 1997.

B. DasGupta, T. Jiang, S. Kannan, M. Li, and Z. Sweedyk. On the complexity and approximation of syntenic distance. unpublished, 1996.

M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A model of evolutionary change in proteins. In M.O. Dayhoff, editor, *Atlas of Protein Sequences and Structure*, volume 5, pages 345–352. National Biomedical Research Foundation, Washington, DC., 1978.

R. Drmanac and R. Crkvenjakov, 1987. Yugoslav Patent Application 570.

R. Durbin, S. Eddy, A. Krough, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, chapter 4. Cambridge University Press, 1998a.

R. Durbin, S. Eddy, A. Krough, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998b.

M. Dyer, A. Frieze, and S. Suen. The probability of unique solution of sequencing by hybridization. *Journal of Computational Biology*, 1:105–110, 1994.

S. R. Eddy. Hidden markov models. *Current Opinions in Structural Biology*, 6(3): 361–365, Jun 1996.

J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17:368–376, 1981.

W. M. Fitch and J. S. Farris. Evolutionary trees with minimum nucleotide replacements from amino acid sequences. *Journal of Molecular Evolution*, 3:263–278, 1974.

N. Friedman, M. Ninio, I. Pe'er, and T. Pupko. A structural EM algorithm for phylogenetic inference. In *Proceedings of the Fifth Annual International Conference on Computational Molecular Biology (RECOMB 2001)*, 2001.

M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Computing*, 5:704–714, 1976.

N. Goldman and Z. Yang. A codon-based model of nucleotide substitution for protein coding DNA sequences. *Molecular Biology and Evolution*, 11(5):725–736, 1994.

D. Gusfield, R. M. Karp, L. Wang, and P. Stelling. Graph traversals, genes and matroids: An efficient case of the travelling salesman problem. *Discrete Applied Mathematics*, 88:167–180, 1998.

J.G. Hacia. Resequencing and mutational analysis using oligonucleotide microarrays. *Nature Genetics*, 21(1):42–47, Jan 1999.

J.G. Hacia, J.B. Fan, O. Ryder, L. Jin, K. Edgemon, G. Ghandour, R.A. Mayer, B. Sun, L. Hsie, C.M. Robbins, L.C. Brody, D. Wang, E.S. Lander, R. Lipshutz,

S.P. Fodor, and F.S. Collins. Determination of ancestral alleles for human single-nucleotide polymorphisms using high-density oligonucleotide arrays. *Nature Genetics*, 22(2):164–167, Jun 1999.

S. Hannenhalli. Polynomial algorithm for computing translocation distance between genomes. *Discrete Applied Mathematics*, 71:137–151, 1996.

S. Hannenhalli, P. Pevzner, H. Lewis, and S. Skiena. Positional sequencing by hybridization. *Computer Applications in the Biosciences*, 12:19–24, 1996.

S. Hannenhalli and P. A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 581–592, Los Alamitos, October 1995. IEEE Computer Society Press. ISBN 0-8186-7183-1.

S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial alg orithm for sorting signed permutations by reversals. *Journal of the ACM*, 46 (1):1–27, January 1999.

D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18,6:341–343, 1975.

J. Jing, J. Reed, J. Huang, X. Hu, V. Clarke, J. Edington, D. Housman, T. S. Anantharaman, E. J. Huff, B. Mishra, B. Porter, A. Shenker, E. Wolfson, C. Hiort, R. Kantor, C. Aston, and D. C. Schwartz. Automated high resolution optical mapping using arrayed, fluid-fixed DNA molecules. *Proceedings of the National Academy Science U.S.A*, 95(14):8046–8051, 1998.

D. T. Jones, W. R. Taylor, and J. M. Thornton. The rapid generation of mutation data matrices from protein sequences. *Computer Applications in the Biosciences*, 8:275–282, 1992.

T. H. Jukes and C. R. Cantor. Evolution of protein molecules. In H.N. Munro, editor, *Mammalian protein metabolism*, pages 21–123. Academic Press, New York, 1969.

H. Kaplan, R. Shamir, and R. E. Tarjan. Faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal of Computing*, 29(3):880–892,

1999. (Preliminary version in Proceedings of the eighth annual ACM-SIAM Symposium on Discrete Algorithms 1997 (SODA 97), ACM Press, pages 344–351).

R. M. Karp, I. Pe'er, and R. Shamir. An algorithm combining discrete and continuous methods for optical mapping. In *Proceedings of the 7nd Annual International Conference on Intelligent Systems for Molecular Biology, (ISMB '99)*, pages 159–168. AAAI Press, 1999.

R. M. Karp and R. Shamir. Algorithms for optical mapping. *Journal of Computational Biology*, 7(1/2):303–316, 2000. (Preliminary version in *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology* (RECOMB), New York, ACM Press, p117–124, 1998).

R.M. Karp, I. Pe'er, and R. Shamir. An algorithm combining discrete and continuous methods for optical mapping. *Journal of Computational Biology*, 7(5): 745–760, 2000.

J. D. Kececioglu and R. Ravi. Of mice and men: Algorithms for evolutionary distances between genomes with translocation. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, pages 604–613, New York, NY, USA, January 1995. ACM Press. ISBN 0-89871-349-8.

K. R. Khrapko, Yu. P. Lysov, A. A. Khorlyn, V. V. Shick, V. L. Florentiev, and A. D. Mirzabekov. An oligonucleotide hybridization approach to DNA sequencing. *FEBS letters*, 256:118–122, 1989.

M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16:111–120, 1980a.

M. Kimura. A simple model for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16:111–120, 1980b.

J.M. Koshi and R.A. Goldstein. Probabilistic reconstruction of ancestral protein sequences. *Journal of Molecular Evolution*, 42:313–320, 1996.

A. Krogh, M. Brown, S. Mian, M. Sjölander, and D. Haussler. Hidden markov models in computational biology. applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1531, 4 February 1994.

E.S. Lander et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, Feb 2001.

J. K. Lee, V. Dančík, and M. S. Waterman. Estimation for restriction sites observed by optical mapping using reversible-jump markov chain monte carlo. In *Proc. RECOMB 98*, pages 147–152, 1998.

B. Lewin. *Genes VI*. Oxford University Press, New York, 1997.

J. Lin, R. Qi, C. Aston, J. Jing, T.S. Anantharaman, B. Mishra, O. White, M.J. Daly, K.W. Minton, J.C. Venter, and D.C. Schwartz. Whole-genome shotgun optical mapping of deinococcus radiodurans. *Science*, 285:1558–1562, 1999.

Y. Lysov, V. Floretiev, A. Khorlyn, K. Khrapko, V. Shick, and A. Mirzabekov. DNA sequencing by hybridization with oligonucleotides. *Dokl. Acad. Sci. USSR*, 303: 1508–1511, 1988.

S. C. Macevics, 1989. International Patent Application PS US89 04741.

X. Meng, K. Benson, K. Chada, E. J. Huff, and D. C. Schwartz. Optical mapping of lambda bacteriophage clones using restriction endonucleases. *Nature Genetics*, 9:432–438, 1995.

S. Muthukrishnan and L. Parida. Towards constructing physical maps by optical mapping: An effective, simple, combinatorial approach. In *Proc. RECOMB 1997*, pages 209–219. ACM Press, 1997.

National Center for Biotechnology Information. A database of single nucleotide polymorphisms, 19 January 2000. `http://www.ncbi.nlm.nih.gov/SNP/`.

L. Parida. On the approximability of physical map problems using single molecule methods. In *Procceedings of Discrete Mathematics and Theoretical Computer Science*, 1999.

L. Parida and B. Mishra. Partitioning k clones: Hardness results and practical algorithms for the k-populations problem. In *Proc. RECOMB 98*, pages 192–201, 1998. To appear in Discrete Applied Math.

I. Pe'er and R. Shamir. The median problem for breakpoints is NP-complete. Technical Report TR98-071, Electronic Colloqium of Computer Science (ECCC), 1998.

I. Pe'er and R. Shamir. Approximation algorithms for the permutations median problem in the breakpoint model,. In D. Sankoff and J. H. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and the Evolution of Gene Families (Proceedings of DCAF'00)*, pages 225–241, Dordrecht, 2000a. Kluwer.

I. Pe'er and R. Shamir. Spectrum alignment: Efficient resequencing by hybridization. In Russ Altman, L. Bailey, Timothy, Philip Bourne, Michael Gribskov, Thomas Lengauer, and Ilya N. Shindyalov, editors, *Proceedings of the 8th International Conference on Intelligent Systems for Molecular (ISMB-00)*, pages 260–268, Menlo Park, CA, August 16–23 2000b. AAAI Press.

P. A. Pevzner. l-tuple DNA sequencing: computer analysis. *J. Biomol. Struct. Dyn.*, 7:63–73, 1989.

P. A. Pevzner and R. J. Lipshutz. Towards DNA sequencing chips. In *Symposium on Mathematical Foundations of Computer Science*, pages 143–158. Springer, 1994. LNCS vol. 841.

P. A. Pevzner, Yu. P. Lysov, K. R. Khrapko, A. V. Belyavsky, V. L. Florentiev, and A. D. Mirzabekov. Improved chips for sequencing by hybridization. *J. Biomol. Struct. Dyn.*, 9:399–410, 1991.

F. Preparata, A. Frieze, and E. Upfal. Optimal reconstruction of a sequence from its probes. *Journal of Computational Biology*, 6(3-4):361–368, 1999.

T. Pupko. *Algorithmic improvements and biological applications of maximum likelihood methods of reconstruction of ancestral amino-acid sequences*. PhD thesis, Tel Aviv University, Tel Aviv, 2000.

T. Pupko, I. Pe'er, R. Shamir, and D. Graur. A fast algorithm for joint maximum likelihood reconstruction of ancestral amino acid sequences. *Molecular Biology and Evolution*, 17:890–896, 2000.

N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

A. Samad, E. J. Huff, and D. C. Schwartz. Optical mapping: A novel, single-molecule approach to genomic analysis. *Genome Rsearch*, 5(1), 1995a.

H. Samad, W. W. Cai, X. Hu, B. Irvin, J. Jing, J. Reed, X. Meng, J. Huang, E. Huff, B. Porter, et al. Mapping the genome one molecule at a time–optical mapping. *Nature*, 378:516–517, 1995b.

D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal of Applied Mathematics*, 28:35–42, 1975.

D. Sankoff and M. Blanchette. The median problem for breakpoints in comparative genomics. *Computing and Combinatorics, 3rd Ann. Int. Conf. COCOON 97*, 1276:251–263, 1997.

D. Sankoff and M. Blanchette. Multiple genome rearrangements and breakpoint phylogeny. *Journal of Computational Biology*, 5:555–570, 1998.

D. Sankoff and J. H. Nadeau. Conserved synteny as a measure of genomic distance. *Discrete Applied mathematics*, 71:247–257, 1996.

D. Sankoff, G. Sundaram, and J. Kececioglu. Steiner points in the space of genome rearrangements. *International journal on Foundations of Computer Science, World scientific*, 7(1):1–9, 1996a. preliminary version: *Genome Rearrangements Workshop*, University of Southern California, 1994.

S. Sankoff, G. Sundaram, and J. D. Kececioglu. Steiner points in the space of genome rearrangements. *IJFCS: International Journal of Foundations of Computer Science*, 7, 1996b.

D. C. Schwartz, X. Li, L. I. Hernandez, S. P. Ramnarain, E. J. Huff, and Y. K. Wang. Ordered restriction maps of saccharomyces cerevisiae chromosomes constructed by optical mapping. *Science*, 262:110–114, 1993.

D. C. Schwartz and A. Samad. Optical mapping approaches to molecular genomics. *Current Opinion. in Biotechnology*, 8(1):70–74, 1997.

S. S. Skiena and G. Sundaram. Reconstructing strings from substrings. *J. Comput. Biol.*, 2:333–353, 1995.

Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

E. Southern, 1988. UK Patent Application GB8810400.

E. M. Southern. DNA chips: analysing sequence by hybridization to oligonucleotides on a large scale. *Trends in Genetics*, 12:110–115, 1996.

E. M. Southern, U. Maskos, and J. K. Elder. Analyzing and comparing nucleic acid sequences by hybridization to arrays of oligonucleotides: evaluation using experimental models. *Genomics*, 13:1008–1017, 1992.

N. Takezaki and T. Gojobori. Correct and incorrect phylogenies obtained by the entire mitochondrial dna sequences. *Molecular Biology and Evolution*, 16:516–601, 1999.

J.C. Venter et al. The sequence of the human genome. *Science*, 291(5507):1304–51, Feb 2001.

D. G. Wang, J. Fan, C Siao, A.Berno, P. Young, R. Sapolsky, G. Ghandour, N. Perkins, E. Winchester, J. Spencer, L. Kruglyak, L. Stein, L. Hsie, T. Topaloglou, E. Hubbell, E. Robinson, M. Mittmann, M. S. Morris, N. Shen, D. Kilburn, J. Rioux, C. Nusbaum, R. Lipshutz, M. Chee, and E. S. Lander. Large scale identification, mapping, and genotyping of single-nucleotide polymorphisms in the human genome. *Science*, 280:1077–1082, 15 May 1998.

Y. Wen and D.M. Irwin. Mosaic evolution of ruminant stomach lysozyme genes. *Molecular Phylogenetics and Evolution*, 13:474–482, 1999.

Z. Yang. Maximum likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites. *Molecular Biology and Evolution*, 10:1396–1401, 1993a.

Z. Yang. Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites. *Molecular Biology and Evoluion*, 10: 1396–1401, 1993b.

Z. Yang. Maximum likelihood phylogenetic estimation from dna sequences with variable rates over sites: Approximate methods. *Journal of Molecular Evolution*, 39:306–314, 1994.

Z. Yang. *PAML: A Program Package for Phylogenetic Analysis by Maximum Likelihood*. University College London, 1999. Version 2.0.

Z. Yang, S. Kumar, and M. Nei. A new method of inference of ancestral nucleotide and amino acid sequences. *Genetics*, 141:1641–1650, 1995.

J. Zhang and S. Kumar. Detection of convergent and parallel evolution at the amino acid sequence level. *Mol. Biol. Evol.*, 14:527–536, 1997.

J. Zhang and M. Nei. Accuracies of ancestral amino acid sequences inferred by the parsimony, likelihood, and distance methods. *Journal of Molecular Evolution*, 44:s139–s146, 1997.

J. Zhang, H.F. Rosenberg, and M. Nei. Positive darwinian selection after gene duplication in primate ribonuclease genes. *Proceedings of the National Academy of Sciences, USA*, 95:3708–3713, 1998.