Dept. of Computer Science and Applied Mathematics

אלגוריתמים קומבינטוריים לניתוח
שינויי סדר גנומיים ומערכי ד.נ.א.

Combinatorial Algorithms for Genome Rearrangements and DNA Oligonucleotide Arrays

THESIS SUBMITTED FOR THE DEGREE OF
"DOCTOR OF PHILOSOPHY"

by

Tzvika Hartman

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

Submitted to the Feinberg Graduate School, Weizmann Institute of Science
September 2004

Abstract

This thesis concerns applications of combinatorial and algorithmic techniques in molecular biology. We focus on two main areas. In the first part we consider algorithms for tracing genome rearrangement events. Genome rearrangements are large scale DNA mutations, in which large segments are rearranged in order and orientation. Since these mutations are very rare, they are useful in evolutionary studies. In particular, we consider rearrangements in which a segment of DNA is cut out and inserted in a different location in the same chromosome. If the segment is inserted in the same orientation it is called transposition; otherwise it is a transreversal.

We first study the problem of sorting permutations by transpositions only. We develop a novel 1.5-approximation algorithm, that is considerably simpler than the extant ones. Moreover, the analysis of the algorithm is significantly less involved, and provides a good starting point for studying related open problems. Next, we consider the problem of sorting permutations by transpositions, transreversals and revrevs (an operation reversing each of two consecutive segments). Our main result is a quadratic 1.5-approximation algorithm for sorting by these operations, improving over the extant best known approximation ratio of 1.75. We present an implementation of both algorithms that runs in time $O(n^{3/2}\sqrt{\log n})$, improving over the quadratic running time of previous algorithms. The improvement is achieved by exploiting an efficient data structure.

The second part of the thesis is devoted to algorithmic issues that arise in the attempt to extract DNA sequence information from oligonucleotide arrays experiments. We consider two main drawbacks of the sequencing by hybridization (SBH) method: lack of methods to handle realistic levels of hybridization errors, and an inherent limitation on the length of uniquely reconstructible sequence by standard universal arrays. We introduce a simple polynomial reconstruction algorithm which can be applied to spectra from standard arrays and has provable performance in the presence of both false negative and false positive errors. We also propose a novel design of chips containing universal bases, and give a simple algorithm that uses spectra from such chips to reconstruct, with high probability, random sequences of length lower only by a squared log factor compared to the information theoretic bound. Our algorithm is very robust to errors, and has a provable performance even if there are both false negative and false positive errors. Simulations indicate that its sensitivity to errors is also very small in practice.

Finally, we study a design and optimization problem that occurs in universal DNA tag arrays. The problem of optimizing the universal array to avoid disruptive cross-hybridization between universal components of the system was addressed previously. Cross-hybridization can, however, also occur assay-specifically, due to unwanted complementarity involving the amplified probes. We examine the problem of identifying the most economic experimental configuration of the probes that avoids cross-hybridization. Our formulation translates this problem into that of covering the vertices of one side of a bipartite graph by a minimum number of balanced subgraphs of maximum degree 1. We show that the general problem is NP-complete. However, in a realistic biological setting the vertices that need to be covered have degrees bounded by d. Under this restriction we develop an $O(d)$-approximation algorithm for the problem. We also give an $O(d)$-approximation for a variant of the problem in which the covering subgraphs are required to be vertex-disjoint.

This thesis is based on the following papers:

1. T. Hartman and R. Shamir. A Simpler and Faster 1.5-Approximation Algorithm for Sorting by Transpositions. Submitted to the *SIAM journal on Computing*. Preliminary version in the *Proceedings of CPM'03*, pages 156-169, 2003.
2. T. Hartman and R. Sharan. A 1.5-Approximation Algorithm for Sorting by Transpositions and Transreversals. Submitted to the *Journal of Computer and System Sciences*. Preliminary version in the *Proceedings of WABI'04*, 2004.
3. E. Halperin, S. Halperin, T. Hartman and R. Shamir. Handling Long Targets and Errors in Sequencing by Hybridization. *Journal of Computational Biology* 10(3-4):483-497, 2003. Preliminary version in the *Proceedings of RECOMB'02*, pages 176-185, 2002.
4. A. Ben-Dor, T. Hartman, R. M. Karp, B. Schwikowski, R. Sharan, and Z. Yakhini. Towards Optimally Multiplexed Applications of Universal Arrays. *Journal of Computational Biology*, 11(2-3):477493, 2004. Preliminary version in the *Proceedings of RECOMB'03*, pages 48-56, 2003.

Declaration: The author, Tzvika Hartman, declares that this thesis summarizes his independent work under the supervision of Prof. Ron Shamir and Prof. Moni Naor, with the following exceptions: The results of Chapter 4 were obtained jointly with Roded Sharan. The results of Chapter 6 were obtained jointly with Eran and Shay Halperin. Chapter 7 is based on a part of reference 4 above. The results included in the chapter were obtained only jointly with Roded Sharan.

Acknowledgments

It is a great pleasure for me to thank the people who helped me during my studies and research.

Above all, I thank my advisors Ron Shamir and Moni Naor. It was Ron who introduced me to the exciting world of computational biology. I am grateful for his patience, guidance and for sharing with me so much of his knowledge. I appreciate very much his professional approach to science. I thank Moni for many fruitful discussions that gave me many useful insights.

I am grateful to all of my collaborators: Roded Sharan who has been both a friend and a close collaborator; Anne Bergeron who has a special and unique approach to mathematical problems; Eran Halperin; Cedric Chauve; Zohar Yakhini; Elad Verbin; Haim Kaplan; Isaac Elias; Eleazar Eskin; Sagi Snir; Richard Karp; Amir Ben-Dor and Benno Schwikowski.

The Weizmann Institute of Science has been a great place to study. I thank my friends that accompanied me all these years: Dani Harnik, Hillel Kugler, Alon Rosen, Ariel Elbaz, Udi Weider, Eran Ofek, Michael Langberg, Ariel Gabizon, Yuval Emek, Asaf Nussbaum and Iftach Haitner. I would like to thank my fellow students and friends from Ron Shamir's lab in Tel-Aviv University: Itsik Pe'er, Tamar Barzuza, Irit Gat-Viks, Adi Maron-Katz, Amos Tanay, Noga Amit, Gadi Kimmel and Chaim Linhart.

I deeply thank the Center for Complexity Science supported by the Yeshaya Horowitz Association for granting me a fellowship throughout my Ph.D. studies.

Finally, I would like to thank my family. My parents, Yehuda and Avigail, for their unconditional love and support, and for being there whenever I needed them. And my beloved and cute sons, Eyal and Ohad, who bring so much joy into my life.

Contents

Chapter 1

Introduction

In this thesis we study algorithmic problems that arise from research in molecular biology. More specifically, we consider issues that relate to DNA sequence information. *DNA* macro-molecules are located in the cells of every living organism, and contain its genetic material. They serve as a program for the cell to construct and operate the molecular machinery. Each DNA molecule is composed of small molecular building blocks, called *nucleotides*. There are four types of nucleotides, denoted by A, C, T and G. Thus, the DNA molecule is represented in computer science as a sequence over these four letters. The DNA sequence of a species is called a *genome*. Determination of the genome sequence (a process called *sequencing*) is a fundamental task towards revealing the secret of life. The sequencing of the human genome was completed recently by the *Human Genome Project* [Lander et al., 2001, Venter et al., 2001], and genomes of many other organisms have been or are currently being sequenced.

In its stable state the DNA is *double stranded*, i.e., it is composed of two DNA strands that are anti-parallel to each other. Each nucleotide in one strand is *hybridized* (i.e., chemically bound by hydrogen bonds) to a corresponding complementary nucleotide in the other strand, where A is the complement of T, and C is the complement of G. The phenomenon of hybridization is exploited in many experiments in molecular biology (see Part II).

Genes are regions of DNA that encode the instructions to the production of *proteins*, which are molecules that are in charge of virtually all organic activity in the cell. Proteins are polymers that are composed of building blocks called *amino acids*. The encoding is done by the *genetic code*, which is a simple mapping that assigns one of the 20 types of amino acids to each of the 64 possible triplets of nucleotides, called *codons*.

When cells divide the DNA is replicated. The replication, however, is not perfect. Errors in replication are called *mutations*. Most of the mutations are local point mutations, i.e., they involve change in the DNA sequence only in a specific location (such as deletion, addition or substitution of nucleotides). Large scale mutations, also known as *genome rearrangements*, are considerably more rare than point mutations. In this case, large segments of DNA are rearranged in both order and orientation. Since genome rearrangements are very rare they are used in evolution studies (see Part I).

For more background on molecular biology refer to [Alberts et al., 1994].

1.1 Overview of the Results

In this section we provide a brief summary of the thesis results. We defer definitions to the relevant chapters. In the first part of the thesis we study algorithms for tracing genome rearrangement events (for background refer to Chapter 2). In the second part we consider algorithmic issues that arise in the attempt to extract DNA sequence information from microarray experiments.

1.1.1 Part I: Algorithms for Genome Rearrangements

Chapter 3

In this chapter we study the problem of sorting permutations by transpositions. First, we prove that the problem of sorting circular permutations by transpositions is equivalent to the problem of sorting linear permutations by transpositions. Hence, all algorithms for sorting linear permutations by transpositions can be used to sort circular permutations. Then, we derive our main result: A novel 1.5-approximation algorithm, that is considerably simpler than the extant ones [Bafna and Pevzner, 1998, Christie, 1999]. Moreover, the analysis of the algorithm is significantly less involved, and provides a good starting point for studying related open problems.

The results of this chapter can be found in [Hartman and Shamir, 2004] (preliminary version in [Hartman, 2003]).

Chapter 4

In this chapter we study the problem of sorting permutations by transpositions, transreversals and revrevs (an operation reversing each of two consecutive segments). We show that the sorting problem is equivalent for linear and circular permutations. Moreover, we observe that for circular permutations revrev and transreversal are equivalent operations.

Thus, any sorting algorithm that uses transpositions, transreversals and revrevs (such as our algorithm and that of Lin and Xue [2001]) can be used to sort circular permutations by transpositions and transreversals, which are more biologically motivated operations. Then we derive our main result: A quadratic 1.5-approximation algorithm for sorting by transpositions, transreversals and revrevs, improving over the extant best known approximation ratio of 1.75 for this problem [Lin and Xue, 2001].

The results of this chapter can be found in [Hartman and Sharan, 2004a] (preliminary version in [Hartman and Sharan, 2004b]).

Chapter 5

We introduce a fast implementation of algorithms SortTranspos (Chapter 3) and SortTransrev (Chapter 4). We present an implementation of both algorithms that runs in time $O(n^{3/2}\sqrt{\log n})$, improving on the quadratic running time of previous algorithms. The improvement is achieved by exploiting an efficient data structure introduced by Kaplan and Verbin [2003] in the context of sorting by reversals.

The results of this chapter can be found in [Hartman and Shamir, 2004] and [Hartman and Sharan, 2004a] (preliminary version in [Hartman and Sharan, 2004b]).

1.1.2 Part II: Extracting Sequence Information from DNA Microarrays

Chapter 6

In this chapter we consider two main drawbacks of the sequencing by hybridization (SBH) method: lack of tools to handle realistic levels of hybridization errors, and an inherent limitation on the length of uniquely reconstructible sequence by standard universal arrays. We introduce a simple polynomial reconstruction algorithm which can be applied to spectra from standard arrays and has provable performance in the presence of both false negative and false positive errors. We also propose a novel design of chips containing universal bases, that differs from the one proposed by [Preparata et al., 1999, Preparata and Upfal, 2000]. We give a simple algorithm that uses spectra from such chips to reconstruct with high probability random sequences of length lower only by a squared log factor compared to the information theoretic bound. Our algorithm is very robust to errors, and has a provable performance even if there are both false negative and false positive errors. Simulations indicate that its sensitivity to errors is also very small in practice.

The results in this chapter were published in [Halperin et al., 2003] (preliminary version in [Halperin et al., 2002]).

Chapter 7

In this chapter we study a design and optimization problem that occurs, for example, when single nucleotide polymorphisms (SNPs) are to be genotyped using a universal DNA tag array. The problem of optimizing the universal array to avoid disruptive cross-hybridization between universal components of the system was addressed in previous work [Ben-Dor et al., 2000]. Cross-hybridization can, however, also occur assay-specifically, due to unwanted complementarity involving assay-specific components.

Here we examine the problem of identifying the most economic experimental configuration of the assay-specific components that avoids cross-hybridization. Our formulation translates this problem into that of covering the vertices of one side of a bipartite graph by a minimum number of balanced subgraphs of maximum degree 1. We show that the general problem is NP-complete. However, in the real biological setting the vertices that need to be covered have degrees bounded by d. Under this restriction we develop an $O(d)$-approximation algorithm for the problem. We also give an $O(d)$-approximation for a variant of the problem in which the covering subgraphs are required to be vertex-disjoint.

The results in this chapter were published in [Ben-Dor et al., 2004] (preliminary version in [Ben-Dor et al., 2003]).

Part I

Algorithms for Genome Rearrangements

Chapter 2

Introduction

When trying to determine evolutionary distance between two organisms using genomic data, one wishes to reconstruct the sequence of events that have occurred, transforming one genome into the other. One of the most promising ways to trace the evolutionary events is to compare the order of appearance of identical (or orthologous) genes in two different genomes. In late 1930's, Dobzhansky and Sturtevant [1938] have shown evidence of inversions in the genome of *Drosophila*. In the 1980's, Palmer and colleagues [Palmer and Herbon, 1986, 1987, 1988, Palmer et al., 1988, Hoot and Palmer, 1994] gave evidence that many different species have essentially the same set of genes, but their order may differ between species. This suggests that global rearrangement events (such as reversals and transpositions of genome segments) can be used to trace the evolutionary path between genomes. Such rare events may provide a more accurate and robust clue to the evolution than local point mutations (i.e. insertions, deletions, and substitutions of nucleotides).

The biological data for addressing a rearrangement problem may be obtained via various experiments, including sequence homology between genes, restriction maps, and other hybridization techniques (see [Li and Graur, 1991] for details). The data obtained from each technique reveals identical locations (points or segments) between the two genomes that can be mapped to each other. These can be extended to maximal similar segments between the two genomes. The mapping between the segments might have them running in the same direction, or in opposite directions, and hence, usually the direction (*sign*) of the mapping is known.

The field of genome rearrangements offers a wide range computational and combinatorial problems. This diversity results from the consideration of different rearrangement events, and of different models of the rearrangement process. Several types of questions are studied, including algorithmic, statistic, and combinatoric problems.

The possible global rearrangement events are:

- A reversal (inversion) of a chromosome segment, namely, cutting the segment from the chromosome and re-inserting it reversed at the same location.
- A transposition of a chromosome segment, namely, cutting the segment and pasting it (possibly reversed) in a new position.
- Chromosome fusion - joining two chromosomes into one chromosome.
- Chromosome fission - splitting one chromosome into two.
- Insertion/deletion of a chromosome segment.
- Reciprocal translocation between chromosomes, namely, cutting ending segments of two chromosomes, and exchanging them.

All these events apparently occur in evolution, as was demonstrated convincingly in the biological literature (cf. [Miller and Therman, 2001]. Due to the model and problem complexity, simplified problems have been studied. The solution to simplified problems may be an approximate solution to the real life problem, or may apply perfectly to specific instances of the problem. One usually assumes that the mapping between the two genomes is one-to-one, namely, with no repeating segments. For a single chromosome, i.e., when considering only intra-chromosomal events, the problem combinatorially formulates as the sorting of a permutation, or, when the sign is known, sorting of a *signed* permutation. One can also restrict the analysis only to inter-chromosomal gene-exchange events, and disregard the intra-chromosomal order of genes, resulting in the problem of *syntenic distance* between genomes.

The genomic distance and the metric it defines give rise to many types of problems:

- What is the distance (i.e. minimum number of events) between two genomes?
- What are the possible shortest sequences of events, that transform one genome to another? How many such sequences are there?
- What is the diameter of the genome space (S_n for the case of unsigned permutations) under the genomic distance metric?
- Given several genomes, find their most parsimonious evolutionary tree. Specifically, construct a tree with the known genomes at the leaves and additional genomes at internal nodes (which represent ancestral species). The edges are weighted by the distance between the genomes at the adjacent nodes. The problem can be studied

when the topology of the tree is known, or is not known. An interesting special case is the median tree for three genomes.

For more background on genome rearrangements refer to [Pevzner, 2000, Shamir, 2002, Setubal and Meidanis, 1997, Sankoff and El-Mabrouk, 2002].

2.1 Definitions

We will focus on comparing two single chromosomes, which will be modelled mathematically by permutations. Let π be a permutation of length n. Denote by id the identity permutation. For a particular operation X (e.g., reversals or transpositions), *sorting by* X is the problem of finding a shortest sequence of Xs that sorts a given permutation (i.e. transforms it into id). The distance of π, denoted $d(\pi)$, is the minimum number of operations required to sort π. Note that this formulation is equivalent to the realistic problem of finding the distance between two arbitrary permutations π and ϕ, since $d(\pi, \phi) = d(\phi^{-1} \cdot \pi, id)$. The *diameter*, $D(n)$, is the maximal value of $d(\pi)$, where π is a permutation of n elements (it is actually the diameter of the Cayley graph of the symmetric group, S_n, with respect to the specific operation).

A *reversal* $\rho = \rho(i, j)$ reverses the order of the segment $[i, j-1]$. Hence, it transforms the permutation $\pi = [\pi_1 \ \dots \ \pi_{i-1} \ \pi_i \ \pi_{i+1} \ \dots \ \pi_j \ \pi_{j+1} \ \dots \ \pi_n]$ into the permutation $[\pi_1 \ \dots \ \pi_{i-1} \ \pi_{j-1} \ \dots \ \pi_{i+1} \ \pi_i \ \pi_j \ \dots \ \pi_n]$. When considering *signed* permutations (which is biologically more relevant, since every gene has an orientation), each element of the permutation is given a sign, '+' or '-'. In signed permutations, the reversal flips the orientation of each element it is acting upon (in addition to reversing their order). Thus, it transforms π into $[\pi_1, \ \dots \ , \pi_{i-1}, -\pi_j, \ \dots \ , -\pi_{i+1}, \pi_j, \ \dots \ , \pi_n]$.

A *transposition* $\tau(i, j, k)$ (where $1 \leq i < j < k \leq n$) is an operation which 'cuts' the segment $[i, j-1]$ and 'pastes' it before the kth position. Formally, it transforms π into $[\pi_1 \ \dots \ \pi_{i-1} \, \pi_j \ \dots \ \pi_{k-1} \, \pi_i \ \dots \ \pi_{j-1} \, \pi_k \ \dots \ \pi_n]$. A *transreversal* (sometimes called *inverted transposition*) is a transposition in which the segment may be inserted in reverse order. From a biological point of view, a transversal is as probable as a transposition, since after the segment is cut, it can be inserted in reverse order with the same probability. As with ordinary reversals, when the permutation is signed the transreversal also flips the sign of each element in the inverted segment.

An important notion in the genome rearrangements literature is the notion of *breakpoint*. A pair of elements $(i, i+1)$ is called a breakpoint if $\pi_i - \pi_{i-1} \neq 1$ (for unsigned permutations the condition is $|\pi_i - \pi_{i-1}| \neq 1$).

2.2 Sorting by Reversals

A reversal can remove at most two breakpoints in a permutation. Therefore, a simple lower bound on the reversal distance is

$$d(\pi) \geq \lceil \frac{b(\pi)}{2} \rceil$$

where $b(\pi)$ is the number of breakpoints in permutation π. Kececioglu and Sankoff [1995] present a greedy 2-approximation algorithm which at each step chooses a reversal that removes the most breakpoints from the permutation. They also provide an exponential exact algorithm using a branch-and-bound approach.

Bafna and Pevzner [1996] introduce the notion of the *breakpoint graph* of a permutation. The breakpoint graph of $\pi = [\pi_1 \ \dots \ \pi_n]$ is an edge-colored graph on $n+2$ vertices $\{\pi_0, \pi_1, \dots, \pi_{n+1}\}$. Vertices π_i and π_j are joined by a *black edge* if (π_i, π_j) is a breakpoint in π and by a *gray edge* if (i, j) is a breakpoint in π^{-1}. An *alternating cycle* is a cycle that has edges of alternating colors. We ignore cycles of length 2. Since each vertex has an equal number of incident gray and black edges, the graph can be completely decomposed (usually non-uniquely) into edge-disjoint alternating cycles. Denote by $c(\pi)$ the number of alternating cycles in a *maximum alternating-cycle decomposition.* Note that the identity permutation is the only one who has no breakpoints and no alternating cycles. Bafna and Pevzner [1996] show that every reversal changes the value $b(\pi) - c(\pi)$ by at most one, implying that

$$d(\pi) \geq b(\pi) - c(\pi)$$

Bafna and Pevzner obtain an $O(n^2)$ 1.75-approximation algorithm for sorting unsigned permutations by reversals, based on properties of the breakpoint graph. The approximation ratio was improved to 1.5 [Christie, 1998] and further to 1.375 [Berman et al., 2002], which is the best known approximation guarantee. The problem of sorting unsigned permutations by reversals was proved to be NP-hard by Caprara [1997]. Furthermore, Berman and Karpinski [1999] prove that it is NP-hard to approximate the unsigned reversal distance to within a factor of 1.008.

As discussed above, the biologically more relevant problem is sorting signed permutations by reversals. Bafna and Pevzner [1996] note that the concept of breakpoint graph extends naturally to signed permutations. Define a transformation of the signed permutation π of length n into an unsigned permutation π' of length $2n$ by replacing the element $+i$ with $2i-1, 2i$ and $-i$ with $2i, 2i-1$. By using only reversals $\rho(i, j)$ such that i is odd and j is even, sorting the resulted unsigned permutation is equivalent to sorting the original signed one. In the resulting breakpoint graph every vertex has degree 0 or 2 and therefore it has a unique alternating-cycle decomposition.

In their seminal paper, Hannenhalli and Pevzner [1999] further investigate the breakpoint graph. They give a closed formula for the reversal distance of a permutation (based on parameters of the breakpoint graph) and provide a polynomial algorithm for sorting by signed reversals. A reversal is called *proper* if it decreases the value of $b(\pi) - c(\pi)$ by one. A cycle is *oriented* if a proper reversal can be performed on edges belonging to that cycle. Gray edges (π_i, π_j) and (π_k, π_l) are *interleaving* if the intervals $[i, j]$ and $[k, l]$ overlap but neither of them contains the other. Cycle C_1 *interleaves* with cycle C_2 if it contains an edge that interleaves with an edge in cycle C_2. An *interleaving graph* is a graph in which each vertex represents a cycle in the original graph, and the edges connect between interleaving cycles. A connected component of the interleaving graph is *oriented* if it contains a vertex that corresponds to an oriented cycle.

Hannenhalli and Pevzner [1999], show that oriented components can be resolved by proper reversals (the first proper reversal follows from the definition of oriented components, the fact that a proper reversal can be found at all of the other steps is not trivial). The unoriented components are divided into "good" and "bad" components. The good components can be resolved by using only proper reversals (roughly speaking, they can be transformed into oriented components while doing a proper reversal on a different component without "wasting" a reversal). Every bad component, which is called a *hurdle*, needs an extra reversal. Based on a deep analysis of these notions, the following theorem was obtained:

$$d(\pi) = b(\pi) - c(\pi) + h(\pi) + f(\pi)$$

where $h(\pi)$ is the number of hurdles in π and $f(\pi) \in \{0, 1\}$ equals 1 if and only if π is a special kind of permutation, called *fortress*. An $O(n^4)$ exact algorithm is obtained based on the theorem.

The bottleneck in the running time of the Hannenhalli-Pevzner algorithm is finding the next reversal in oriented components, which involves computing the connected components in the interleaving graph. Berman and Hannenhalli [1996] present a $O(n\alpha(n))$ algorithm (where α is the inverse Ackerman function) for computing connected components in such graphs, which in turn yields an $O(n^2\alpha(n))$ algorithm. Kaplan et al. [2000] simplify the combinatorial structures and improve the running time to $O(n^2)$ (it is actually $O(r \cdot n + n\alpha(n))$, where r is the number of reversals required to sort the permutation). Recently, Tannier and Sagot [2004] present an algorithm for sorting by reversals in time $O(n^{3/2}\sqrt{\log n})$.

All the above algorithms compute the reversal distance and also provide an optimal sequence of reversals that sorts the permutation. Bader et al. [2001] provide a linear time algorithm that computes only the reversal distance. A neat representation of the

Hannenahalli-Pevzner theory, along with an algorithm for sorting by reversals without using the breakpoint graph is given in [Bergeron, 2001, Bergeron et al., 2004].

The sorting algorithms provide an arbitrary sequence of reversals that optimally sorts the permutation, out of many possible sequences. Bergeron et al. [2002] investigate the space of all optimal sequences of reversals, and give a combinatorial structure to this space.

Bafna and Pevzner [1996] prove that the reversal diameter of S_n (unsigned permutations) is $n-1$. The signed reversal diameter for permutations of length at least 4 is $n+1$ (see [Christie, 1999]).

2.3 Sorting by Transpositions

Analogously to the reversal case, a transposition can remove at most 3 breakpoints from a permutation and, therefore, a simple lower bound on the transposition distance is

$$d(\pi) \geq \frac{b(\pi)}{3} \tag{2.1}$$

Bafna and Pevzner [1998] use the breakpoint graph (which was originally introduced for sorting by reversals) for the problem of sorting by transpositions. As in the reversal case, the identity permutation is the only permutation that has $n+1$ cycles and therefore, sorting can be viewed as a process that increases the number of cycles from $c(\pi)$ to $n+1$. Since a transposition may change the number of cycles by at most 2, a better lower bound is obtained [Bafna and Pevzner, 1998]

$$d(\pi) \geq \frac{n+1-c(\pi)}{2}$$

This lower bound and a careful analysis of the types of cycles in permutations leads Bafna and Pevzner to provide a polynomial time 1.5-approximation algorithm for sorting by transpositions. Christie [1999] further analyzes the structure of the transposition cycle graph and gains a simpler algorithm with the same performance guarantee. An $O(n^3)$ implementation of the latter algorithm, along with heuristics that improve its performance, are given in [Walter et al., 2003]. A considerably simpler 1.5-approximation algorithm for sorting by transpositions is given in Chapter 3. Eriksson et al. [2001] devise an algorithm which sorts any given permutation by at most $2n/3$ transpositions. Walter et al. [2000] introduce a simple structure, called *breakpoint diagram*, and obtain a 2.25-approximation algorithm. The complexity of sorting by transpositions is still open.

The transposition diameter of S_n is still an open problem. The algorithm of Eriksson et al. [2001] gives an upper bound of $2n/3$ on the diameter. Several works show that the transposition distance of the reverse permutation is $\lfloor n/2 \rfloor + 1$ [Christie, 1999, Eriksson et al., 2001, Walter et al., 1998] Therefore, $\lfloor n/2 \rfloor + 1 \leq D(n) \leq 2n/3$. Eriksson et al. [2001] show that for $1 \leq n \leq 15, D(n) = \lfloor n/2 \rfloor + 1$ except for $n = 13$ and $n = 15$.

2.4 Mixed Operations

In real life, genomes evolve by several types of rearrangement operations. Walter et al. [1998] describe a 3-approximation algorithm for sorting unsigned permutations by reversals and transpositions, and a 2-approximation algorithm for the signed case. When allowing also transreversals there is a 2-approximation algorithm as well [Gu et al., 1999]. Lin and Xue [2001] describe a 1.75-approximation algorithm when allowing also the revrev operation, in which two consecutive segments are reversed. The approximation ratio is improved to 1.5 in the algorithm introduced in Chapter 4. Walter et al. [1998] show that $\lfloor \frac{n}{2} \rfloor + 2$ is a lower bound on the reversal and transposition diameter for signed permutations. The complexity of all the above minimum distance problems, as well as the diameter of the symmetric group under these mixed operations, is unknown.

Blanchette et al. [1996] and Eriksen et al. [2001] investigate the relative frequency of reversals and transpositions in real genomic data. They conclude that the most biologically meaningful problem is to sort signed permutations by reversals and transpositions, where the weight of transpositions is roughly twice the weight of reversals. Eriksen [2002] gives a $(1 + \epsilon)$- approximation algorithm (for any $\epsilon > 0$) for the minimum weighted distance problem allowing reversals and transpositions of signed permutations, where transpositions are given double weight.

2.5 Circular Permutations

Circular genomes, such as bacterial and mitochondrial genomes, are represented by circular permutations. All the above problems can be defined analogously for circular permutations. Meidanis et al. [2000] prove that sorting circular signed permutations by reversals is equivalent to the linear case. The same result holds also for unsigned permutations [Solomon et al., 2003]. The analogous results for sorting by transpositions and for sorting by transpositions and transreversals are given in Chapters 3 and 4 respectively.

Chapter 3

Sorting by Transpositions

3.1 Introduction

In this chapter we study the problem of sorting permutations by transpositions. First, we prove that the problem of sorting circular permutations by transpositions is equivalent to the problem of sorting linear permutations by transpositions. Hence, all algorithms for sorting linear permutations by transpositions can be used to sort circular permutations. Then, we derive our main result: A novel 1.5-approximation algorithm, that is considerably simpler than the extant ones [Bafna and Pevzner, 1998, Christie, 1999]. Moreover, the analysis of the algorithm is significantly less involved, and provides a good starting point for studying related open problems.

The results of this chapter can be found in [Hartman and Shamir, 2004] (preliminary version in [Hartman, 2003]).

3.1.1 Organization of the Chapter

In Section 3.2 we first prove the equivalence between the problem of sorting linear and circular permutations by transpositions. Then, we review some classical genome rearrangement results, and show that every permutation can be transformed into a so-called simple permutation. Our main result, a new and simple 1.5-approximation algorithm for sorting permutations by transpositions, is introduced in Section 3.3. We conclude in Section 3.4 with a short discussion and some open problems.

3.2 Preliminaries

3.2.1 Linear and Circular Permutations

Let $\pi = [\pi_1 \ \dots \ \pi_n]$ be a permutation on n elements. Define a *segment* A in π as a consecutive sequence of elements $\pi_i, \dots, \pi_k$ $(k \geq i)$. Two segments $A = \pi_i, \dots, \pi_k$ and $B = \pi_j, \dots, \pi_l$ are *contiguous* if $j = k+1$ or $i = l+1$. A *transposition* τ on π is the exchange of two disjoint contiguous segments (Figure 3.1a). If the segments are $A = \pi_i, \dots, \pi_{j-1}$ and $B = \pi_j, \dots, \pi_{k-1}$, then by performing τ on π, the resulting permutation, denoted $\tau \cdot \pi$, is $[\pi_1 \ \dots \ \pi_{i-1} \ \pi_j \ \dots \ \pi_{k-1} \ \pi_i \ \dots \ \pi_{j-1} \ \pi_k \ \dots \ \pi_n]$ (note that the end segments can be empty if $i = 1$ or $k - 1 = n$). We shall say that τ *cuts* π *before* positions i, j and k. We say that τ *operates* on index l if $i \leq l < k$, i.e., if l belongs to one of the two exchanged segments.

In circular permutations, one can define a transposition analogously as the exchange of two contiguous segments. Note that here the indices are cyclic, so the disjointness of the exchanged segments is a meaningful requirement. The transposition partitions a circular permutation into three segments, as opposed to at most four in a linear permutation (see Figure 3.1). Note that for circular permutations we can assume w.l.o.g. that all three segments are nonempty, since otherwise the original and transformed permutations are the same. Since there are only two cyclic orders on three segments, and each two of the three segments are contiguous, the transposition can be represented by exchanging any two of them. Note that the number of possible transpositions on a linear n-permutation is $\binom{n+1}{3}$, since there are $n + 1$ possible cut points of segments. In contrast, in a circular n-permutation there are only $\binom{n}{3}$ possibilities.

The problem of finding a shortest sequence of transpositions, which transforms a (linear or circular) permutation into the identity permutation, is called *sorting by transpositions*. The *transposition distance* of a permutation π, denoted by $td(\pi)$, is the length of the shortest sorting sequence.

Theorem 1 *The problem of sorting linear permutations by transpositions is linearly equivalent to the problem of sorting circular permutations by transpositions.*

Proof: Given a linear n-permutation, circularize it by adding an $n+1'$st element $\pi_{n+1} = x$, and closing the circle (see Figure 3.1c). Call the new circular permutation π^c. By the discussion above, any transposition on π^c can be represented by the two segments that do not include x. Hence, there is an optimal sequence of transpositions that sorts π^c, and none of them operates on x. The same sequence can be viewed as a sequence of transpositions on the linear permutation π, by ignoring x. This implies that $td(\pi) \leq td(\pi^c)$. On the

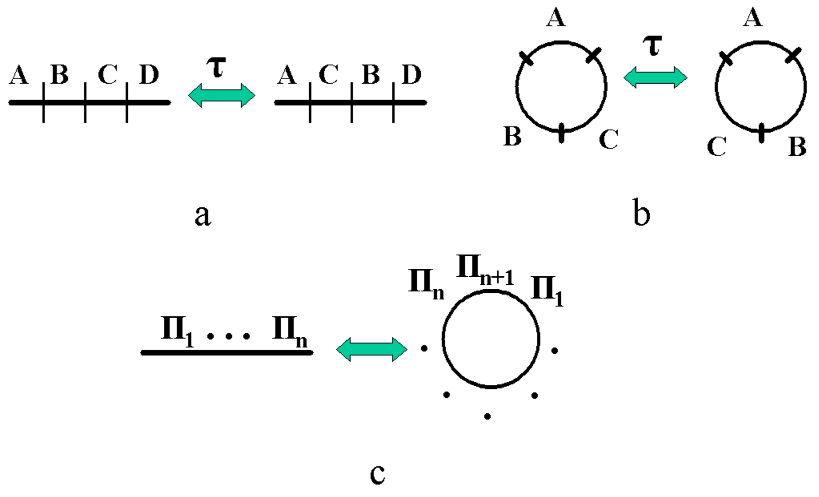


Figure 3.1: (a) A transposition τ, which is applied on a linear permutation, and exchanges segments B and C. (b) A transposition τ, which is applied on a circular permutation. τ can be viewed as exchanging A and B, or B and C, or A and C. (c) A one-to-one transformation between linear and circular permutations. In the circular permutation, a new element, π_{n+1}, is introduced.

other hand, any sequence of transpositions on π is also a sequence of transpositions on π^c, so $td(\pi^c) \leq td(\pi)$. Hence, $td(\pi) = td(\pi^c)$. Moreover, an optimal sequence for π^c provides an optimal sequence for π.

For the other direction, starting with a circular permutation, we can linearize it by removing an arbitrary element, which plays a role of x above (see Figure 3.1c). The same arguments imply that an optimal solution for the linear permutation translates to an optimal solution for the circular one. ■

In the rest of the chapter, we will discuss only circular permutations. As implied by Theorem 1, all the results on circular permutations hold also for linear ones. We prefer to work with circular permutations since it simplifies the analysis.

3.2.2 The Circular Breakpoint Graph

We transform a permutation π on n elements into a permutation $f(\pi)$ on $2n$ elements, by replacing each element i by two elements $2i-1, 2i$. On the doubled permutation $f(\pi)$, we allow only transpositions that cut before odd positions. This ensures that no transposition cuts between $2i-1$ and $2i$, and therefore every transposition on π can be mimicked by a transposition on $f(\pi)$. We call such transpositions *legal*. We now define

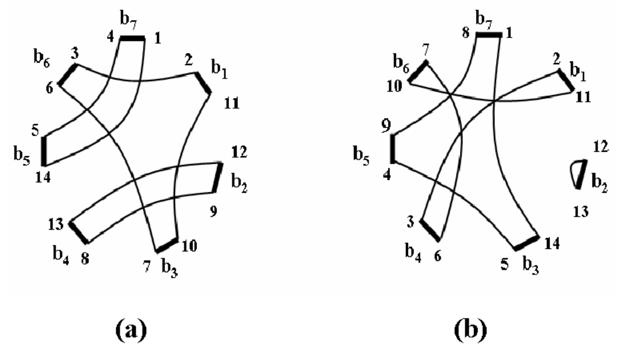


Figure 3.2: (a) The circular breakpoint graph of the permutation $\pi = [1\ 6\ 5\ 4\ 7\ 3\ 2]$, for which $f(\pi) = [1\ 2\ 11\ 12\ 9\ 10\ 7\ 8\ 13\ 14\ 5\ 6\ 3\ 4]$. Black edges are represented as thick lines on the circumference, and gray edges are chords.(b) The circular breakpoint of π after applying the transposition that acts on black edges b_2, b_4 and b_7.

the circular breakpoint graph, which is the circular version of the breakpoint graph [Bafna and Pevzner, 1996]. Throughout, in both indices and elements, we identify $2n+1$ and 1.

Definition 1 *Let* $\pi = (\pi_1\ \dots\ \pi_n)$ *be a circular permutation, and* $f(\pi) = \pi' = (\pi'_1\ \dots\ \pi'_{2n})$. *The* breakpoint graph $G(\pi)$ *is an edge-colored graph on* $2n$ *vertices* $\{1, 2, \dots, 2n\}$. *For every* $1 \leq i \leq n$, π'_{2i} *is joined to* π'_{2i+1} *by a black edge (denoted by* b_i*), and* $2i$ *is joined to* $2i+1$ *by a gray edge.*

Note that unlike previous studies of transpositions [Bafna and Pevzner, 1998, Christie, 1999], we chose to double the number of vertices and work with an undirected graph, as done in the signed case [Bafna and Pevzner, 1996]. It is convenient to draw the breakpoint graph on a circle, such that black edges are on the circumference and gray edges are chords (see Figure 3.2(a)). We shall use this representation throughout the chapter.

Since the degree of each vertex is exactly 2, the graph uniquely decomposes into cycles. Denote the number of cycles in $G(\pi)$ by $c(\pi)$. The *length* of a cycle is the number of black edges it contains. A *k-cycle* is a cycle of length k, and it is *odd* if k is odd. The number of odd cycles is denoted by $c_{odd}(\pi)$. Define $\Delta c(\pi, \tau) = c(\tau \cdot \pi) - c(\pi)$, and $\Delta c_{odd}(\pi, \tau) = c_{odd}(\tau \cdot \pi) - c_{odd}(\pi)$.

Bafna and Pevzner proved the following useful lemma (This - and other results we quote - was proved for linear permutations, but holds also for circular ones):

Lemma 2 (Bafna and Pevzner [1998]) *For all permutations* π *and transpositions* τ, *it holds that* $\Delta c(\pi, \tau) \in \{-2, 0, 2\}$, *and* $\Delta c_{odd}(\pi, \tau) \in \{-2, 0, 2\}$.

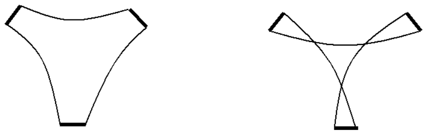


Figure 3.3: The only two possible configurations of 3-cycles. The left one is unoriented, and the right one is oriented.

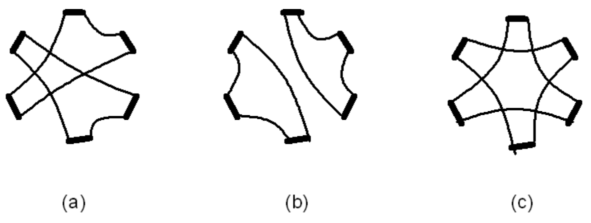


Figure 3.4: (a) Intersecting 3-cycles. (b) Non-intersecting 3-cycles. (b) Interleaving 3-cycles.

Let $n(\pi)$ denote the number of black edges in $G(\pi)$. The maximum number of cycles is obtained iff π is the identity permutation. In that case, there are $n(\pi)$ cycles, and all of them are odd (in particular, they are all of length 1) . Starting with π with c_{odd} odd cycles, Lemma 2 implies the following lower bound on $td(\pi)$:

Theorem 3 (Bafna and Pevzner [1998]) *For all permutations π, $td(\pi) \geq (n(\pi) - c_{odd}(\pi))/2$.*

By definition, every legal transposition must cut three black edges. The transposition that cuts black edges b_i, b_j and b_k is said to *act on* these edges (see Figure 3.2(b)). A transposition τ is a *k-transposition* if $\Delta c_{odd}(\pi, \tau) = k$. A cycle is called *oriented* if there is a 2-transposition that acts on three of its black edges; otherwise, it is *unoriented.*

Observation 4 *There are only two possible configurations of 3-cycles that can be obtained by legal transpositions.*

The two possibilities are shown in Figure 3.3. It is easy to verify that the left 3-cycle is unoriented, and the right one is oriented.

Given a cyclic sequence of elements $i_1, \ldots, i_k$, an *arc* is an interval in the cyclic order, i.e., a set of contiguous elements in the sequence. The pair (i_j, i_l) $(j \neq l)$ defines two disjoint arcs: $i_j, \ldots, i_{l-1}$ and $i_l, \ldots, i_{j-1}$. Similarly, a triplet defines a partition of the cycle into three disjoint arcs. We say that two pairs of black edges (a, b) and (c, d) are *intersecting* if a and b belong to different arcs defined by the pair (c, d). A pair of black

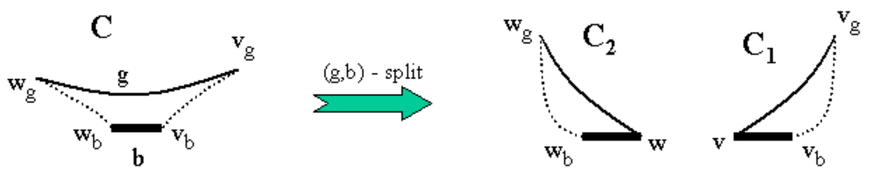


Figure 3.5: A (g, b)-split. A dashed line indicates a path.

edges intersects with cycle C, if it intersects with a pair of black edges that belong to C. Cycles C and D intersect if there is a pair of black edges in C that intersect with D (see Figure 3.4a). Two triplets of black edges are *interleaving* if each of the edges of one triple belongs to a different arc of the second triple. Two 3-cycles are interleaving if their edges interleave (see Figure 3.4c).

Throughout the chapter, we use the term permutation also when referring to the breakpoint graph of the permutation (as will be clear from the context). For example, when we say that π contains an oriented cycle, we mean that $G(\pi)$ contains an oriented cycle.

3.2.3 Transformation into Equivalent Simple Permutations

A k-cycle in the breakpoint graph is called *short* if $k \leq 3$; otherwise, it is called *long*. A breakpoint graph is called *simple* if it contains only short cycles. A permutation π is called *simple* if $G(\pi)$ is simple. Following [Hannenhalli and Pevzner, 1999, Lin and Xue, 2001], we show how to transform an arbitrary permutation into a simple one, while maintaining the lower bound of Theorem 3.

Let $b = (v_b, w_b)$ be a black edge and let $g = (v_g, w_g)$ be a gray edge belonging to the same cycle $C = (\ldots, v_b, w_b, \ldots, w_g, v_g, \ldots)$ in $G(\pi)$. A (g, b)-*split* of $G(\pi)$ is a sequence of operations on $G(\pi)$, resulting in a new graph $\hat{G}(\pi)$ with one more cycle, as follows:

- Removing edges b and g.
- Adding two new vertices v and w.
- Adding two new black edges (v_b, v) and (w, w_b).
- Adding two new gray edges (w_g, w) and (v, v_g).

Figure 3.5 shows a (g, b)-split transforming a cycle C in $G(\pi)$ into two cycles C_1 and C_2 in $\hat{G}(\pi)$. Note that the order of the nodes of each edge along the cycle is important, as other orders may not split the cycle. Hannenhalli and Pevzner [1999] show that for every (g,b)-split on a permutation π of n elements, there is a permutation $\hat{\pi}$ of $n+1$ elements, that is obtained by inserting an element into π, such that $\hat{G}(\pi) = G(\hat{\pi})$. Thus, a (g,b)-split can be viewed as a transformation from π to $\hat{\pi}$. A (g,b)-split is called *safe* if $n(\pi) - c_{odd}(\pi) = n(\hat{\pi}) - c_{odd}(\hat{\pi})$, i.e., if it maintains the lower bound of Theorem 3.

Lemma 5 (Lin and Xue [2001]) *Every permutation can be transformed into a simple one by safe splits.*

Proof: Let π be a permutation that contains a long cycle C. Let b_1 be a black edge in C. Denote by b_2 and b_3 the black edges that are connected to b_1 via a gray edge. Let g be the gray edge that is connected to b_2 but not to b_1. Then a (g,b_3)-split breaks C into a 3-cycle and a $(k-2)$-cycle in $\hat{\pi}$. Clearly, $n(\hat{\pi}) = n(\pi) + 1$, and $c_{odd}(\hat{\pi}) = c_{odd}(\pi) + 1$, so the split is safe. This process can be repeated until a simple permutation is eventually obtained. ■

We say that permutation π is *equivalent* to permutation $\hat{\pi}$ if $n(\pi) - c_{odd}(\pi) = n(\hat{\pi}) - c_{odd}(\hat{\pi})$.

Lemma 6 (Hannenhalli and Pevzner [1999]) *Let $\hat{\pi}$ be a simple permutation that is equivalent to π, then every sorting of $\hat{\pi}$ mimics a sorting of π with the same number of operations.*

In the following, we show how to sort a simple permutation by transpositions. We prove that the number of transpositions is within a factor of 1.5 from the lower bound of Theorem 3. Thus, we obtain a 1.5-approximation algorithm for sorting simple permutations. The above discussion implies that this algorithm translates into a 1.5-approximation algorithm for an arbitrary permutation: Transform the permutation into an equivalent simple permutation (Lemma 5), sort it, and then mimic the sorting on the original permutation (Lemma 6).

3.3 The Algorithm

In this section we provide a 1.5-approximation algorithm for sorting permutations by transpositions. We first develop an algorithm for simple permutations, and then use the results of Section 3.2.3 to prove the general case. Recall that the breakpoint graph of a simple permutation contains only 1-, 2- and 3-cycles. Our goal is to obtain a graph with 1-cycles only, which is the breakpoint graph of the identity permutation. Thus, the sorting can be viewed as a process of transforming the 2- and 3-cycles into 1-cycles.

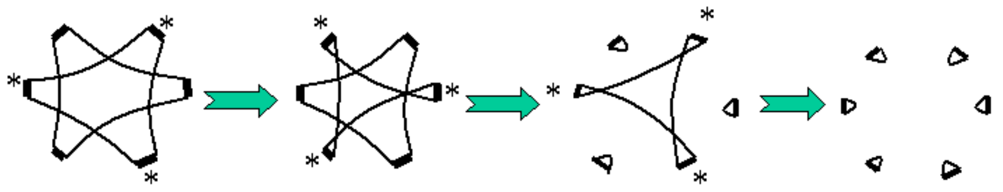


Figure 3.6: A $(0, 2, 2)$-sequence of transpositions for two interleaving unoriented 3-cycles. At each step the transposition acts on the three black edges that are marked by a star.

First we deal with the case that the permutation contains a 2-cycle:

Lemma 7 (Christie [1999]) *If π is a permutation that contains a 2-cycle, then there exists a 2-transposition on π.*

By definition, an oriented 3-cycle can be eliminated by a 2-transposition that acts on its black edges. Suppose from now on that all 2-cycles were eliminated by applying Lemma 7, and all oriented 3-cycles were eliminated. The only remaining problem is how to handle unoriented 3-cycles. This is the case we analyze henceforth.

A *(0,2,2)-sequence* is a sequence of three transpositions, of which the first is a 0-transposition, and the next two are 2-transpositions. Note that a $(0, 2, 2)$-sequence increases the number of odd cycles by 4 out of 6 that are the maximum possible in 3 steps, and thus a series of $(0, 2, 2)$-sequences preserves a 1.5 approximation ratio. We shall show below that such a sequence is always possible.

Lemma 8 *Let π be a permutation that contains two interleaving unoriented 3-cycles. Then, there exists a $(0, 2, 2)$-sequence of transpositions on π.*

Proof: The $(0, 2, 2)$-sequence is described in Figure 3.6. ■

Lemma 9 *Let C and D be two intersecting unoriented 3-cycles that are not interleaving. Then, there exists a transposition which transforms C and D into a 1-cycle and an oriented 5-cycle.*

Proof: Let c_1, c_2 and c_3 be the three black edges of C. Assume, without loss of generality, that (c_1, c_2) intersects with D. We shall in fact prove a stronger statement, namely, for any choice of a black edge $d \in D$ such that (d, c_3) intersects with (c_1, c_2), the transposition on c_1, c_2 and d satisfies the lemma. There are three possible cases to consider, which are shown in Figure 3.7. In each case, the first transposition, which acts on c_1, c_2 and d,

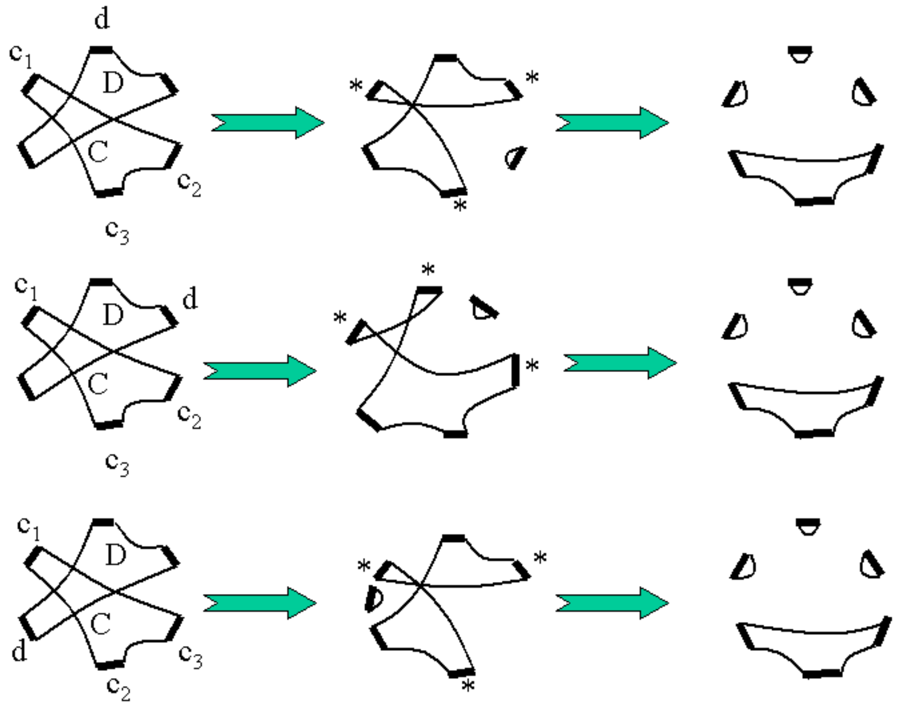


Figure 3.7: The three possible cases of two intersecting unoriented 3-cycles that are not interleaving. In each case, the transposition that acts on edges c_1, c_2, and d, transforms C and D into a 1-cycle and an oriented 5-cycle.

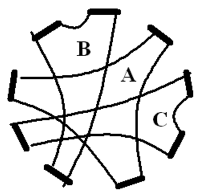


Figure 3.8: Cycle shattering. Cycle A is shattered by B and C. Cycle C is not shattered by A and B.

transforms 3-cycles C and D into a 1-cycle and a 5-cycle. Then, in order to show that the 5-cycle is oriented, a 2-transposition which acts on three of its edges is shown. ■

We say that cycle E is *shattered* by cycles C and D if every pair of edges in E intersects with a pair of edges in C or with a pair of edges in D (see Figure 3.8).

Lemma 10 *Let π be a permutation that contains three unoriented 3-cycles C, D and E, such that E is shattered by C and D. Then, there exists a $(0, 2, 2)$-sequence of transpositions on π.*

Proof: If two of the three cycles are interleaving, the (0,2,2)-sequence follows from Lemma 8. Otherwise, there are two general cases:

1. Two out of the three cycles are non-intersecting. In this case, there are three possible configurations of the cycles, which are shown in Figure 3.9. For every sub-case, a $(0, 2, 2)$-sequence is shown.

2. The three cycles are mutually intersecting. The general case is illustrated in Figure 3.10. Since cycles C and D are unoriented, the condition of the proof of Lemma 9 is fulfilled. Thus, we can apply a 0-transposition that acts on edges c_1, c_2, and d, and obtain a new oriented cycle F. Now we apply a 2-transposition on E (which has also become oriented). Cycle F remains oriented, since the latter transposition does not change its structure. Thus, another 2-transposition is possible on the edges of F, which completes the $(0, 2, 2)$-sequence.

■

A pair of black edges is said to be *connected* if they are connected by a gray edge.

Lemma 11 (Bafna and Pevzner [1998]) *Let (b_i, b_j) be a connected pair in an unoriented cycle. Then, (b_i, b_j) intersects with some other cycle.*

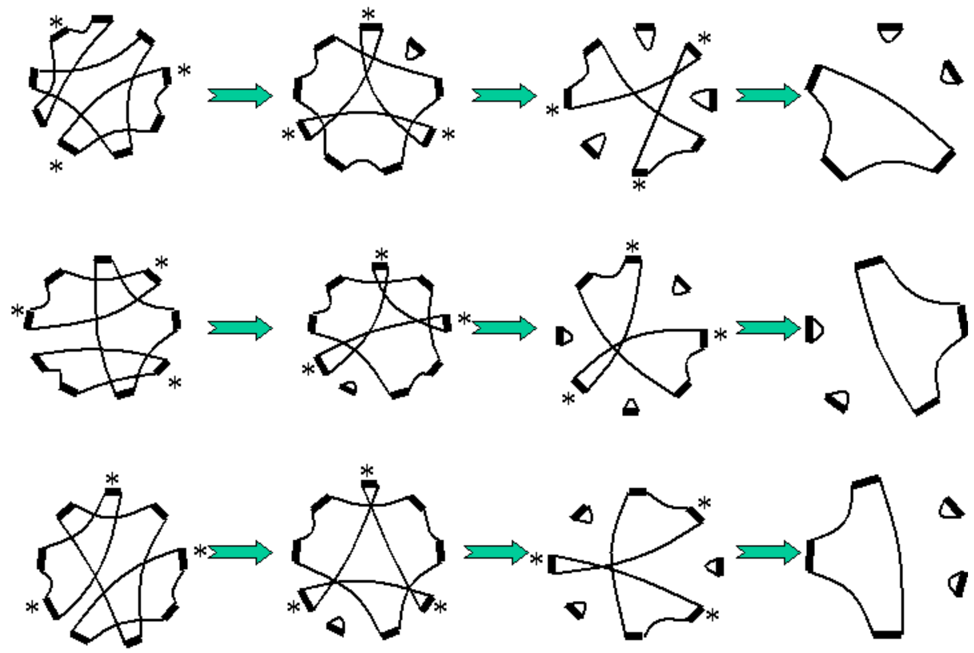


Figure 3.9: The three possible cases of three unoriented 3-cycles, such that one of them is shattered by the other two, no pair is interleaving and two of them are non-intersecting. In each case, a $(0, 2, 2)$-sequence of transpositions is shown. For simplicity, every 1-cycle is shown only when it is formed and not in subsequent graphs (since it is not affected by transpositions in later steps).

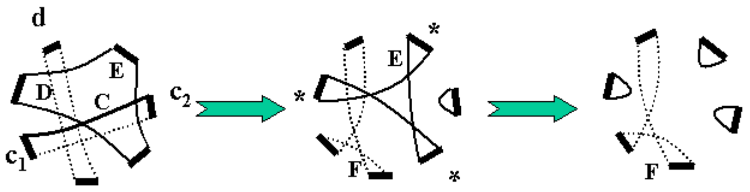


Figure 3.10: Three mutually intersecting unoriented cycles such that no pair is interleaving, and one is shattered by the other two. A dashed line represents either a single gray edge or a path of length 3. Note that edges c_1 and c_2 are connected by a single gray edge.

We are now ready to present the full algorithm. It is described in Figure 3.11. Note that in steps 2 - 3 it is impossible to create a long cycle, and thus the permutation remains simple throughout the algorithm. Note also that in step 3 we do not create 2-cycles, and hence, there is no need to iterate over step 2.

```
Algorithm SortTranspos ($\pi$)
  1. Transform permutation $\pi$ into an equivalent simple permutation $\hat{\pi}$ (Lemma 5).
  2. While $G(\hat{\pi})$ contains a 2-cycle, apply a 2-transposition (Lemma 7).
  3. While $G(\hat{\pi})$ contains a 3-cycle $C$, do:
       • Pick a connected pair of black edges $c$ from cycle $C$. If $C$ is oriented - apply
         a 2-transposition.
       • Otherwise, pick a connected pair $d$ (from some cycle $D$) that intersects with
         $c$ (the existence of $d$ is guaranteed by Lemma 11). If $C$ and $D$ are interleaving
         - apply a $(0,2,2)$-sequence (Lemma 8).
       • Otherwise, there is a connected pair $c'$ in $C$ that does not intersect with
         $D$. Pick a connected pair $e$ (from some cycle $E$) that intersects with $c'$ (the
         existence of $e$ is guaranteed by Lemma 11). As cycle $C$ is shattered by cycles
         $D$ and $E$, apply a $(0,2,2)$-sequence (Lemma 10).
  4. Mimic the sorting of $\pi$ using the sorting of $\hat{\pi}$ (Lemma 6).
```

Figure 3.11: SortTranspos: A 1.5-approximation algorithm for sorting by transpositions.

Performing only steps 2 - 3 is an algorithm in its own, and is denoted by *Algorithm SortSimple*. The following lemma claims that SortSimple is a quadratic time 1.5-approximation algorithm for sorting simple permutations:

Lemma 12 *Algorithm* SortSimple *is a 1.5-approximation algorithm for simple permutations, and it runs in time $O(n^2)$.*

Proof: The sequence of transpositions that is generated by the algorithm contains only 2-transpositions and $(0, 2, 2)$-sequences of transpositions. Therefore, every sequence of three transpositions increases the number of odd cycles by at least 4 out of 6 possible in 3 steps (as implied from the lower bound of Theorem 3). Hence, the approximation ratio is 1.5.

We now analyze the running time of the algorithm. Step 2 can be done in linear time.

The number of iterations in step 3 is linear, since every iteration we break a 3-cycle into three 1-cycles. The main operations in each iteration is to find a connected pair that intersects with a given pair, and to apply a transposition (the other operations can be done in constant time). These operations can be done trivially in linear time. Hence, the algorithm is quadratic. ■

Now we are ready to prove the correctness of Algorithm *Sort*:

Theorem 13 *Algorithm* SortTranspos *is a 1.5-approximation algorithm for general permutations, and it runs in time $O(n^2)$.*

Proof: By Lemma 12, we are guaranteed that $alg(\hat{\pi}) \leq 1.5 \cdot td(\hat{\pi})$, where $alg(\hat{\pi})$ is the number of transpositions used by Algorithm SortSimple to sort $\hat{\pi}$. Thus, by Theorem 3,

$$alg(\hat{\pi}) \leq 1.5td(\hat{\pi}) \leq 1.5\left(\frac{n(\hat{\pi}) - c_{odd}(\hat{\pi})}{2}\right) = 1.5\left(\frac{n(\pi) - c_{odd}(\pi)}{2}\right) \leq 1.5 \cdot td(\pi)$$

Using Lemma 6, we can sort π by $alg(\hat{\pi})$ transpositions, which implies an approximation ratio of 1.5.

Since steps 1 and 4 can be done in linear time, Lemma 12 implies that the running time of Algorithm Sort is $O(n^2)$. ■

3.4 Discussion and Subsequent Work

In this chapter we studied the problem of sorting permutations by transpositions, simplified the underlying theory, and gave a simple 1.5-approximation algorithm for the problem. We believe that this is an important step towards solving some related open problems. The main open problem is to determine the complexity of sorting by transpositions. Devising algorithms with better approximation ratio and/or faster running time is also desirable. Another direction, which is more biologically relevant, is to consider algorithms for sorting permutations by a set of rearrangement operations (such as reversals, transpositions and translocations). A step towards this direction is made in Chapter 4.

An implementation of our algorithm, along with a comparison to the previous ones is reported in [Honda, 2004, Walter et al.].

Chapter 4

Sorting by Transpositions and Transreversals

4.1 Introduction

In this chapter we study the problem of sorting permutations by transpositions, transreversals and revrevs (an operation reversing each of two consecutive segments). We show that the sorting problem is equivalent for linear and circular permutations. Moreover, we observe that for circular permutations revrev and transreversal are equivalent operations. Thus, any sorting algorithm that uses transpositions, transreversals and revrevs (such as our algorithm and that of Lin and Xue [2001]) can be used to sort circular permutations by transpositions and transreversals, which are more biologically motivated operations. Then we derive our main result: A quadratic 1.5-approximation algorithm for sorting by transpositions, transreversals and revrevs, improving over the extant best known approximation ratio of 1.75 for this problem [Lin and Xue, 2001].

The results of this chapter can be found in [Hartman and Sharan, 2004a] (preliminary version in [Hartman and Sharan, 2004b]).

4.1.1 Organization of the Chapter

Background on rearrangement operations, permutations and their representation is given in Section 4.2, where we also describe the reduction to sorting simple circular permutations, and show that every permutation can be transformed into a so-called 3-permutation. The approximation algorithm is given in Section 4.3.

4.2 Preliminaries

A *signed permutation* $\pi = [\pi_1 \ \dots \ \pi_n]$ on $n(\pi) \equiv n$ elements is a permutation in which each element is labeled by a sign of plus or minus. A *segment* of π is a consecutive sequence of elements $\pi_i, \dots, \pi_k$ $(k \geq i)$. We focus on four rearrangement *operations*. A *reversal* ρ is an operation that reverses the order of the elements in a segment and flips their signs. If the segment is $\pi_i, \dots, \pi_{j-1}$ then $\rho \cdot \pi = [\pi_1 \ \dots \ \pi_{i-1} \ \ -\pi_{j-1} \ \dots \ -\pi_i \ \pi_j \ \dots \ \pi_n]$. Two segments $\pi_i, \dots, \pi_k$ and $\pi_j, \dots, \pi_l$ are *contiguous* if $j = k+1$ or $i = l+1$. A *transposition* τ exchanges two contiguous (disjoint) segments. If the segments are $A = \pi_i, \dots, \pi_{j-1}$ and $B = \pi_j, \dots, \pi_{k-1}$ then $\tau \cdot \pi = [\pi_1 \ \dots \ \pi_{i-1} \ \pi_j \ \dots \ \pi_{k-1} \ \pi_i \ \dots \ \pi_{j-1} \ \pi_k \ \dots \ \pi_n]$ (note that the end segments can be empty if $i = 1$ or $k = n+1$). A *transreversal* $\tau\rho_{A,B}$ is a transposition that exchanges segments A and B and also reverses A, i.e., $\tau\rho_{A,B} \cdot \pi = [\pi_1 \ \dots \ \pi_{i-1} \ \pi_j \ \dots \ \pi_{k-1} \ \ -\pi_{j-1} \ \dots \ \ -\pi_i \ \pi_k \ \dots \ \pi_n]$, and $\tau\rho_{B,A} \cdot \pi = [\pi_1 \ \dots \ \pi_{i-1} \ \dots \ \ -\pi_{k-1} \ \dots \ \ -\pi_j \ \pi_i \ \dots \ \pi_{j-1} \ \pi_k \ \dots \ \pi_n]$. A *revrev* operation reverses each of the two segments (without transposing them). Thus, $\rho\rho \cdot \pi = [\pi_1 \ \dots \ \pi_{i-1} \ \ -\pi_{j-1} \ \dots \ \ -\pi_i \ \ -\pi_{k-1} \ \dots \ \ -\pi_j \ \pi_k \ \dots \ \pi_n]$.

The problem of finding a shortest sequence of transposition, transreversal and revrev operations that transforms a permutation into the identity permutation is called *sorting by transpositions and transreversals*[1]. The *distance* of a permutation π, denoted by $trd(\pi)$, is the length of the shortest sorting sequence.

4.2.1 Linear vs. Circular Permutations

Key to our approximation algorithm is a reduction from the problem of sorting linear permutations to that of sorting *circular* permutations (indices are cyclic), on which the analysis is simpler. An operation is said to *operate* on the segments that are affected by it and on the elements in those segments. We say that two operations μ and μ' are *equivalent* if they have the same effect, i.e., $\mu \cdot \pi = \mu' \cdot \pi$ for all π. The following lemma is the basis for the reduction, and is used to prove the subsequent theorem on the equivalence of the sorting problem for linear and circular permutations, similarly to the transpositions only case (Section 3.2.1).

Lemma 14 *Let x be an element of a circular permutation π, and let μ be an operation that operates on x. Then there exists an equivalent operation μ' that does not operate on x.*

[1]We do not include revrevs in the problem name, as we provide in the next section a reduction of the problem that allows us to mimic revrevs using transreversals.

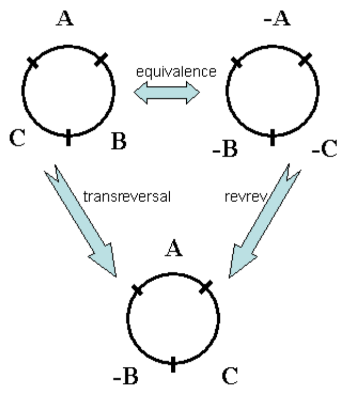


Figure 4.1: The equivalence of operations on circular permutations.

Proof: For reversals, this result was proven by Meidanis et al. [2000]; for transpositions it follows from the discussion in Section 3.2.1. For transreversals and revrevs, the claim relies on the observation that a chromosome is equivalent to its *reflection*, i.e., the reversed sequence of elements with their signs flipped [Meidanis et al., 2000] (see the upper part of Figure 4.1). Consider a permutation with three segments: A, B and C, where $x \in A$. Then a transreversal that operates on segments A and B and reverses B (resp., A) is equivalent to a revrev that operates on A and C (resp., B and C), since the result is a reflection of the permutation (as illustrated in Figure 4.1). Similarly, a revrev that operates on A and B (or C) is equivalent to a transreversal that operates on B and C. ■

Invoking Lemma 14 in the proof of Theorem 1 yields the following result:

Theorem 15 *The problem of sorting linear permutations by transpositions and transreversals is linearly equivalent to the problem of sorting circular permutations by transpositions and transreversals.*

We observe that for circular permutations revrevs and transreversals are equivalent operations. Thus, for circular permutations we can restrict attention to transpositions and transreversals, which are better motivated biologically compared to revrevs. Moreover, combined with Theorem 15, this observation implies that one can reduce the problem of sorting a linear permutation by transpositions, transreversals and revrevs to that of sorting a circular permutation by transpositions and transreversals only. We note that the problem of sorting circular permutations is important on its own, since many genomes including mitochondrial and bacterial ones are circular.

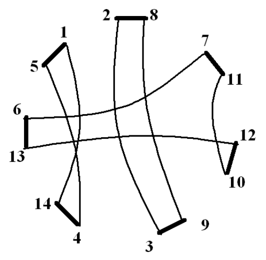


Figure 4.2: The circular breakpoint graph of the permutation $\pi = [1\ -4\ 6\ -5\ 2\ -7\ -3]$, for which $f(\pi) = [1\ 2\ 8\ 7\ 11\ 12\ 10\ 9\ 3\ 4\ 14\ 13\ 6\ 5]$. Black edges are represented as thick lines on the circumference, and gray edges are chords.

4.2.2 The Breakpoint Graph

In the following we define the breakpoint graph for signed permutations, which is a generalization of the graph defined in Section 3.2.2 for unsigned permutations. Some of the definitions are the same as in the unsigned case. For completeness, we give here all relevant definitions. We follow the construction of Bafna and Pevzner [1996] for representing signed permutations. First, a permutation π on n elements is transformed into a permutation $f(\pi) = \pi' = (\pi'_1\ \dots\ \pi'_{2n})$ on $2n$ elements. $f(\pi)$ is obtained by replacing each positive element i with two elements $2i-1, 2i$ (in this order), and each negative element with $2i, 2i-1$. For the extended permutation $f(\pi)$, only operations that cut before odd positions are allowed. This ensures that every operation on $f(\pi)$ can be mimicked by an operation on π. In the rest of the chapter we identify, in both indices and elements, $2n+1$ and 1.

Definition 2 *The* circular breakpoint graph $G(\pi)$ *is an edge-colored graph on* $2n$ *vertices* $\{1, 2, \dots, 2n\}$. *For every* $1 \le i \le n$, π'_{2i} *is joined to* π'_{2i+1} *by a black edge, and* $2i$ *is joined to* $2i+1$ *by a gray edge.*

It is convenient to draw the breakpoint graph on a circle, such that black edges are on the circumference and gray edges are chords (see Figure 4.2). Since the degree of each vertex is exactly 2, the graph uniquely decomposes into cycles. A *k-cycle* is a cycle with k black edges, and it is *odd* if k is odd. k is called the *length* of the cycle. The number of odd cycles in $G(\pi)$ is denoted by $c_{odd}(\pi)$. Gu et al. [1999] have shown that for all linear permutations π and operations μ (reversals, transpositions, transreversals or revrevs), it holds that $c_{odd}(\mu \cdot \pi) \le c_{odd}(\pi) + 2$. Their result holds also for circular permutations and can be used to prove the following lower bound on $trd(\pi)$:

Theorem 16 (Gu et al. [1999]) *For all permutations π, $trd(\pi) \geq (n(\pi) - c_{odd}(\pi))/2$.*

4.2.3 Transformation into 3-Permutations

Our goal in this section is to transform the input permutation into a permutation with simple structure, to which we can apply our algorithm and mimic its steps on the original permutation. A permutation is called *simple* if its breakpoint graph contains only k-cycles, where $k \leq 3$. It is called a *3-permutation* if it contains only 1-cycles and 3-cycles. A transformation from π to $\hat{\pi}$ is called *safe* if $n(\pi) - c_{odd}(\pi) = n(\hat{\pi}) - c_{odd}(\hat{\pi})$, i.e., if it maintains the lower bound of Theorem 16. Next, we show how to transform an arbitrary permutation into a 3-permutation using safe transformations. We note that the transformation maintains only the lower bound, not the exact distance. Our starting point is the standard safe transformation into simple permutations (cf. Section 3.2.3). Hence, to obtain a 3-permutation it suffices to show how to convert 2-cycles into 3-cycles using safe transformations.

Let C be a 2-cycle and let $b = (\pi'_{2i}, \pi'_{2i+1})$ be one of its black edges. A (C, b)*-padding* extends the original permutation π by adding a new element $\pi_i + 1$, and renaming all elements $j > \pi_i + 1$ by $j + 1$ (the renaming is done on the absolute values of the elements and then their signs are reintroduced, e.g., -3 is renamed to -4). The new element $\pi_i + 1$ has the same sign as π_i, and is placed after (resp., before) π_i if it is positive (resp., negative). Finally, the sign of π_i is flipped. The effect on the breakpoint graph is that C is transformed into a 3-cycle (see Figure 4.3 for an example). Overall, the permutation after the padding has an additional element and one more odd cycle.

Lemma 17 *Every simple permutation π can be transformed into a 3-permutation $\hat{\pi}$ by safe paddings. Moreover, every sorting of $\hat{\pi}$ mimics a sorting of π with the same number of operations.*

Proof: Let π be a simple permutation that contains a 2-cycle C, and let $b \in C$. Let $\overline{\pi}$ be the permutation obtained by applying a (C, b)-padding to π. Clearly, $n(\overline{\pi}) = n(\pi) + 1$ and $c_{odd}(\overline{\pi}) = c_{odd}(\pi) + 1$, so the padding is safe. This process can be repeated until a 3-permutation $\hat{\pi}$ is obtained. Since $\hat{\pi}$ is obtained from π by padding new elements, every operation on $\hat{\pi}$ can be mimicked on π by ignoring the padded elements. ■

In the rest of the chapter we shall restrict attention to circular 3-permutations and often refer to the 3-cycles in our breakpoint graph simply as cycles. In Section 4.3 we show how to sort a 3-permutation using at most $1.5l$ operations, where l is the lower bound of Theorem 16. By Theorem 15 and Lemma 17 this implies a 1.5-approximation algorithm for sorting arbitrary circular and linear permutations.

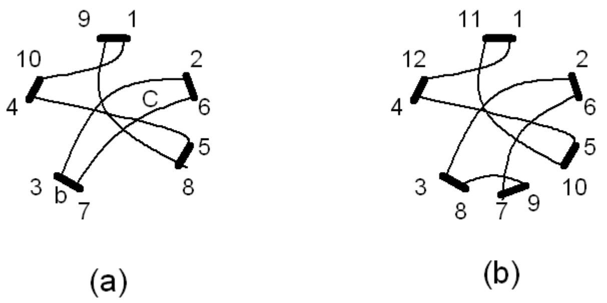


Figure 4.3: (a) The breakpoint graph of the permutation $\pi = [1\ -3\ -4\ 2\ -5]$. (b) The graph of $[1\ -3\ -5\ 4\ 2\ -6]$, which is obtained by a (C, b)-padding.

Cycle Configurations

An operation that cuts some black edges is said to *act on* these edges. It is called a *k-operation* if it increases the number of odd cycles by k. A $(0, 2, 2)$*-sequence* is a sequence of three operations, of which the first is a 0-operation and the next two are 2-operations. Since a 2-operation is the best possible in one step, a series of $(0, 2, 2)$-sequences guarantees a 1.5 approximation ratio.

Definition 3 *An odd cycle is called* oriented *if there is a 2-operation that acts on three of its black edges; otherwise, it is* unoriented.

A *configuration* of cycles is a subgraph of the breakpoint graph that contains one or more cycles. There are four possible configurations of single 3-cycles, which are shown in Figure 4.4(a-d). It is easy to verify that cycles a and b are unoriented, whereas c and d are oriented.

Definition 4 *A black edge is called* twisted *if its two adjacent gray edges cross each other as chords in the circular breakpoint graph. A cycle is k-*twisted *if k of its black edges are twisted. For example, in Figure 4.4 cycle a is 0-twisted and c is 2-twisted.*

Observation 18 *A 3-cycle is oriented iff it is 2- or 3-twisted.*

Proof: For a 3-twisted cycle, a transposition is a 2-operation; for a 2-twisted cycle, a transreversal that reverses the segment between the two twists is a 2-operation. ■

We now give terminology to describe certain configurations of cycles. A pair of black edges is said to be *connected* if they are connected by a gray edge. A pair of connected black edges is *coupled* if they are read in the same direction when reading the edges along

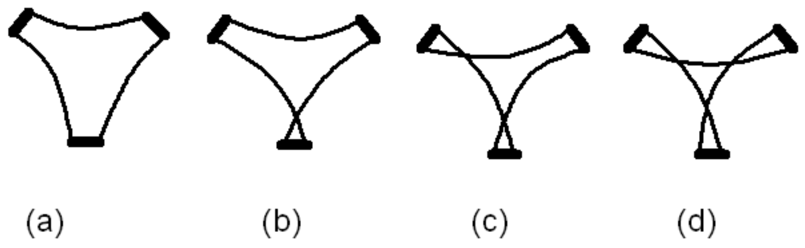


Figure 4.4: Configurations of 3-cycles. (a-b) Unoriented 3-cycles. (c-d) Oriented 3-cycles.

the cycle (For example, the top edges in Figure 4.4b are coupled, and so are all pairs of edges in Figure 4.4a). Let $b = (i_1, i_2)$ and $b' = (j_1, j_2)$ be two black edges in the breakpoint graph such that i_1,i_2,j_1 and j_2 occur in this order along the circle. Then b_1 and b_2 *induce* two disjoint *arcs* on the circle, one between i_2 and j_1 and the other between j_2 and i_1. Given a cycle C and two of its black edges b_1, b_2 that occur consecutively on C, we shall refer to any black edge (of another cycle) as lying *between* b_1 and b_2, if this edge occurs along the arc induced by b_1 and b_2 that does not involve any other edge of C. Two pairs of black edges are called *intersecting* if they alternate in the order of their occurrence along the circle. A pair of black edges intersects with cycle C, if it intersects with a pair of black edges that belong to C. Cycles C and D intersect if there is a pair of black edges in C that intersect with D (see Figure 3.4a). Two cycles are *interleaving* if their black edges alternate in their order of occurrence along the circle (see Figure 3.4b). A *1-twisted pair* is a pair of 1-twisted cycles, whose twists are consecutive on the circle in a configuration that consists of these two cycles only.

Two arcs are called *adjacent* if both endpoints of each arc are connected by gray edges to the endpoints of the other arc. (For example, the arcs induced by the pairs $(1, 2)$ and $(4, 5)$ in Figure 4.5(a)). Given a breakpoint graph $G(\pi)$, we define its *complement* as the graph formed from $G(\pi)$ by replacing each black edge that connects π'_{2i} to π'_{2i+1} with a black edge that connects π'_{2i-1} to π'_{2i} (this notion is related to Caprara's Hamiltonian Matching [Caprara, 1999]). By construction, every vertex i is connected in the complement graph to $i - 1$ and $i + 1$, hence:

Observation 19 *The complement breakpoint graph of a permutation is a cycle of length* $2n$*.*

The following lemma, proved originally by Gu et al. [1999], follows from the latter observation:

Lemma 20 (Gu et al. [1999]) *Let (b_1, b_2) be a pair of coupled black edges. Then there exists another pair of black edges that intersects with b.*

Proof: Suppose to the contrary that no pair intersects (b_1, b_2). Then the complement graph contains at least two disjoint cycles, one in each of the arcs induced by the endpoints of b_1 and b_2, in contradiction to Observation 19. ■

Lemma 21 *Let i_1 and i_2 be a pair of adjacent arcs. Then there exist two connected black edges b_1 and b_2 such that: (1) b_1 is either in i_1 or in i_2; and (2) b_2 is neither in i_1 nor in i_2.*

Proof: Suppose to the contrary that no such gray edge exists. Then in the complement graph there is a cycle that lies in arcs i_1 and i_2, but does not include all the vertices, a contradiction. ■

A 1-twisted cycle is called *closed* (w.r.t. a configuration) if its two coupled edges intersect with some other cycle in the configuration. A configuration is *closed* if at least one of its 1-twisted cycles is closed; otherwise it is called *open*. As we show next, any graph with a 1-twisted cycle has a closed configuration.

Observation 22 *Let $G(\pi)$ be a breakpoint graph that contains a 1-twisted cycle C. Then $G(\pi)$ contains a closed configuration.*

Proof: By Lemma 20 there exists another cycle D that intersects with the coupled edges of C. The configuration which consists of cycles C and D is closed. ■

4.2.4 Canonical Labeling of Cycles

In this section we develop a characterization of oriented cycles that allows us to borrow some of the theory developed in Section 3.3 for unsigned permutations. A useful tool that we will require is the signed canonical labeling[2] of a cycle in a breakpoint graph, which we present next.

For a given cycle C (of any length), consider the labeling of its black edges obtained by labeling an arbitrary edge by 1, and labeling the rest of the cycle's black edges according to their occurrence in clockwise order along the circle. The *signed canonical labeling* of C is the signed permutation obtained by starting with the edge labeled 1 and reading the labels in the order they appear along the cycle, where the signs stand for the direction in

[2]A generalization of the notion of canonical labeling in [Christie, 1999].

which the edge is read: An edge that is visited in the same direction as the edge labeled 1 is positive, and otherwise it is negative (see Figure 4.5). This definition captures the notion of twists in 3-cycles; indeed, a 0-twisted 3-cycle has labeling $(1, 2, 3)$, a 3-twisted cycle has labeling $(1, -3, -2)$, etc. Note that a cycle typically has more than one possible canonical labeling, since it depends on the choice of the first edge.

A canonical labeling of a 5-cycle is called *oriented* if it starts with $1, b, a$ or $1, -a, -b$ or $1, -b, a$ or $1, b, -a$, where $1 < a < b$. The motivation for this definition comes from the following observation:

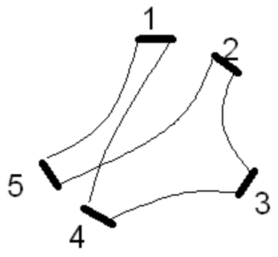


Figure 4.5: A 5-cycle with signed canonical labeling $(1, -4, -3, -2, 5)$.

Lemma 23 *A 5-cycle is oriented iff it has an oriented canonical labeling.*

Proof: A general 5-cycle that has a canonical labeling that starts with $1, b, a$ (where $1 < a < b$) is depicted in Figure 4.6. A transposition that acts on $1, a$ and b transforms the cycle into two 1-cycles and one 3-cycle, showing that the 5-cycle is oriented. A general 5-cycle that has a canonical labeling that starts with $1, -a, -b$ (resp., $1, -b, a$ or $1, b, -a$) is depicted in Figure 4.7. In these cases we consider a transreversal that acts on these three edges, while reversing the segment between edges a and b (resp., 1 and a, or b and 1). This operation breaks the 5-cycle into two 1-cycles and one 3-cycle, implying that the 5-cycle is oriented.

Conversely, consider a 5-cycle C that admits a 2-operation. By definition, this operation creates two 1-cycles and one 3-cycle. An example for the case of a 2-transposition is given in Figure 4.6. Thus, starting clockwise with the edge that will be part of the 3-cycle as edge 1, we obtain the requested canonical labeling (see, e.g., Figure 4.6). ■

Lemma 24 *Let C be a 5-cycle that admits a 2-transposition, and let D be a cycle that has the same canonical labeling as C, up to flipping the sign of one element. Then D is also oriented.*

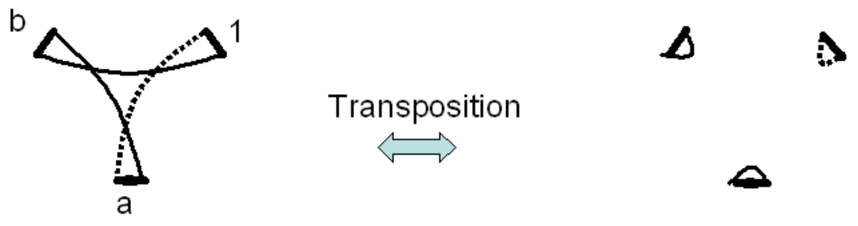


Figure 4.6: A general 5-cycle that has a canonical labeling that starts with $1, b, a$. The dashed line stands for a path of two black edges and two gray edges (the black edges can be located anywhere along the circle).

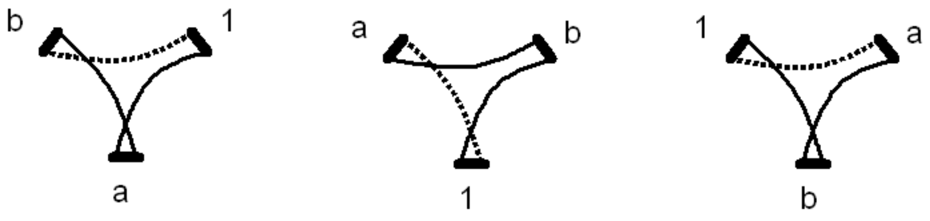


Figure 4.7: General 5-cycles that have a canonical labeling that starts with $1, -a, -b$ and $1, -b, a$ and $1, b, -a$.

Proof: By the proof of Lemma 23, C has a canonical labeling that starts with $1, b, a$, where $1 < a < b$. Let x be the element whose sign is flipped. If x is one of the last two elements in the canonical labeling then D still has a labeling that starts with $1, b, a$ and, thus, admits a 2-transposition. If x is the second (resp., third) element of D then D has a canonical labeling that starts with $1, -b, a$ (resp., $1, b, -a$). Hence, by Lemma 23 the 5-cycle is oriented. If x is the first element, we observe that the reflection of D has a canonical labeling that starts with $1, -a, -b$. ■

4.3 The Algorithm

In this section we provide a 1.5 approximation algorithm for sorting by transpositions and transreversals. We first develop an algorithm for sorting 3-permutations, and then use the results of Section 4.2.3 to generalize it to arbitrary permutations. By definition, an oriented cycle can be eliminated by a 2-operation that acts on its black edges. Thus, from now on, we will only consider unoriented cycles. Since configurations involving only 0-twisted cycles were handled in Section 3.3, by Observation 22 we may restrict attention to closed configurations. For each possible closed configuration we shall prove the existence of a $(0, 2, 2)$-sequence of operations.

The following lemma deals with interleaving cycle pairs:

Lemma 25 *Let π be a permutation that contains two unoriented, interleaving cycles C and D that do not form a 1-twisted pair. Then π admits a $(0, 2, 2)$-sequence.*

Proof: If both cycles are 0-twisted then a $(0, 2, 2)$-sequence of transpositions is given in Section 3.3. Suppose that C is 0-twisted and D 1-twisted (resp., both are 1-twisted and their twists are not consecutive on the circle). First apply a 0-transposition that acts on the black edges of C. This makes D 2-twisted, so it is possible to eliminate it using a 2-transreversal. The latter operation makes C 2-twisted (resp., 3-twisted). A 2-transreversal (resp., 2-transposition) on C completes the $(0, 2, 2)$-sequence. The $(0, 2, 2)$-sequences are depicted in Figure 4.8. ■

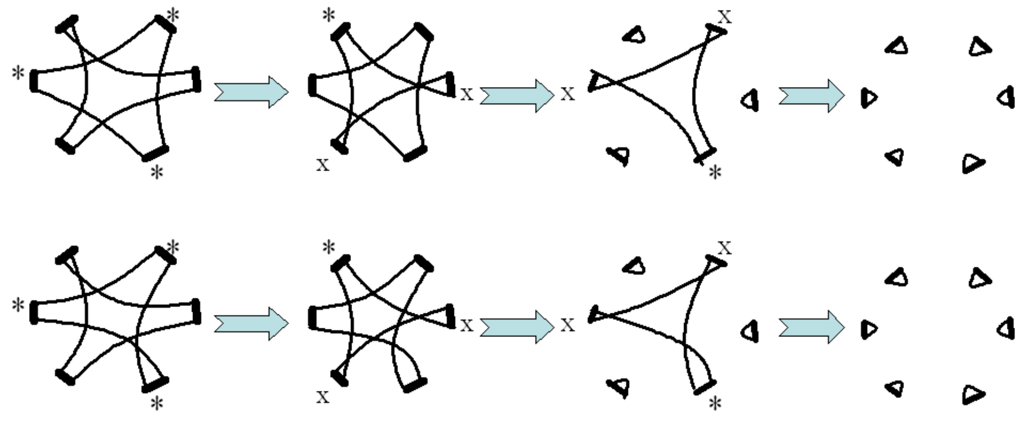


Figure 4.8: $(0, 2, 2)$-sequences for the two cases of two interleaving cycles considered in Lemma 25. Here and throughout the chapter, three asterisks represent a transposition that acts on the three marked black edges. Two x's and a asterisk stand for a transreversal that acts on the three marked edges and reverses the segment between the two x's.

In order to deal with intersecting cycles we use the notion of canonical labeling of cycles, defined in Section 4.2.4. The following lemma handles the case of two intersecting 0-twisted cycles.

Lemma 26 *Let π be a permutation that contains a closed configuration with two intersecting, 0-twisted cycles C and D. Then π admits a $(0, 2, 2)$-sequence.*

Proof: Since C and D are intersecting, C has a pair of coupled edges that do not intersect with D. By Lemma 20 there exists a cycle E that intersects with this pair of edges. The case in which E is 0-twisted was treated in Section 3.3. If E is 1-twisted there are two cases to consider:

1. D and E are non-intersecting. Our starting point is the $(0, 2, 2)$-sequences for configurations of three 0-twisted cycles given in Figure 3.9, where two of the cycles are non-intersecting, and the third one intersects both. In our case, one of the non-intersecting cycles corresponds to E and is 1-twisted. Depending on the location of the twist in E, it is always possible to apply the first two transpositions shown in Figure 3.9 to the closed configuration–the first transposition is applied to the edges shown in the figure, if all are non-twisted, or to a symmetric set of edges. Since the three configurations given here are symmetric with respect to the two non-intersecting cycles, we can ensure that the black edge(s) from cycle E that are involved in the transposition do not include a twist. Indeed, if this is not the case, we simply exchange the choice of edges between D and E, choosing in each case symmetric edges from the other cycle. By Lemma 24, the resulting 5-cycle is oriented, which completes the $(0, 2, 2)$-sequence.

2. D and E are intersecting. Consider the $(0, 2, 2)$-sequences for three mutually intersecting 0-twisted cycles given in Figure 3.10. In our case either D or E are 1-twisted. If all three edges d, e_1 and e_2 that are cut by the first transposition are non-twisted, we apply the first two transpositions as in Figure 3.10. By Lemma 24, the resulting 5-cycle F is oriented. The same holds for any set of symmetric edges that are non-twisted. The only closed configurations in which no such symmetric set is possible is when some arc induced by a pair of black edges of C contains a single twist. There are three such configurations, for which $(0, 2, 2)$-sequences are described in Figure 4.9.

■

Next, we deal with closed configurations that include two intersecting, 1-twisted cycles. We need the following observations:

Observation 27 *Let π be a permutation that contains a 2-twisted cycle C and a 1-twisted cycle D, such that C and D are intersecting and none of the arcs induced by the two twists of C contains both non-twists of D. Then π admits two consecutive 2-operations.*

Proof: Applying a 2-transreversal on C eliminates it, while making D 2-twisted. Thus, two consecutive 2-operations are possible. ■

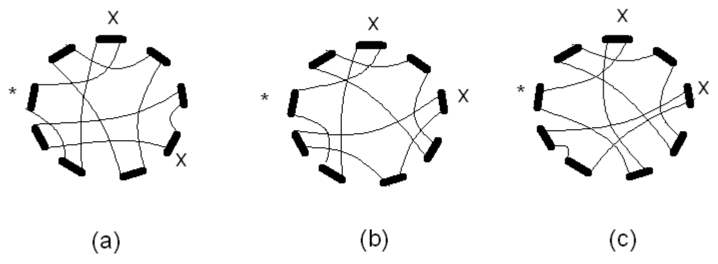


Figure 4.9: $(0,2,2)$-sequences for some cycle configurations that contain two intersecting 0-twisted cycles. First we apply a 0-transreversal on the three marked edges, such that the segment between the two x's is reversed, resulting in an oriented 3-cycle and a 5-cycle. Next, we eliminate the oriented 3-cycle and are left with a 5-cycle, which can be verified to be oriented by Lemma 23. Hence, a $(0,2,2)$-sequence is possible.

Observation 28 *Let C be a 2-twisted cycle such that in a given configuration there are no black edges from other cycles between its two twists. Then it is possible to apply a 2-operation on C that does not affect other cycles in the configuration.*

Proof: A 2-transreversal on C switches the segment between its two twists with a segment between a twist and a non-twist of C. Since the former segment does not involve black edges from other cycles in the configuration, the claim follows. ■

Observation 29 *Let C and D be two 2-twisted, interleaving cycles. Then these cycles admit two consecutive 2-operations iff at least three of their twists are consecutive on the circle.*

Proof: If C and D have at least three consecutive twists then a 2-operation on any of them leaves the other cycle 2-twisted, and the claim follows. Conversely, suppose to the contrary that the non-twist of each cycle lies between the two twists of the other cycle (which is the only configuration in which there are no three consecutive twists). Then a 2-operation on one cycle makes the other one 0-twisted, a contradiction. ■

Lemma 30 *Let π be a permutation that contains a closed configuration with two intersecting, 1-twisted cycles C and D. Then π admits a* $(0,2,2)$*-sequence.*

Proof: There are six possible cases, shown in Figure 4.10. For cases (a-d), we first apply a 0-reversal that acts on the black edges that are marked by an asterisk. This makes the other cycle 2-twisted, and two additional 2-operations follow from Observation 27. For cases (e-f), we observe that by Lemma 20 there is another cycle that has a black edge in the arc denoted x, and a black edge in one of the other five arcs. We apply a reversal that acts on these two edges of the third cycle. If the edges are coupled then the reversal does not affect the cycle. Otherwise, it breaks this 3-cycle into a 1-cycle and a 2-cycle (which we safely transform into a 3-cycle). Thus, in both cases the reversal is a 0-operation. To show that two 2-operations follow, we consider three possible cases with respect to the resulting configuration of C and D:

- The resulting configuration consists of a 1-twisted cycle and a 2-twisted cycle. In all cases the configuration fulfills the condition of Observation 27 and, thus, two 2-operations follow.
- C and D intersect and are both 2-twisted. In all cases one of the cycles fulfills the condition of Observation 28 and can be eliminated, while leaving the other cycle 2-twisted. Thus, two 2-operations are possible.
- C and D interleave and are both 2-twisted. In all cases C and D have at least three consecutive twists on the circle, and two 2-operations follow from Observation 29.

■

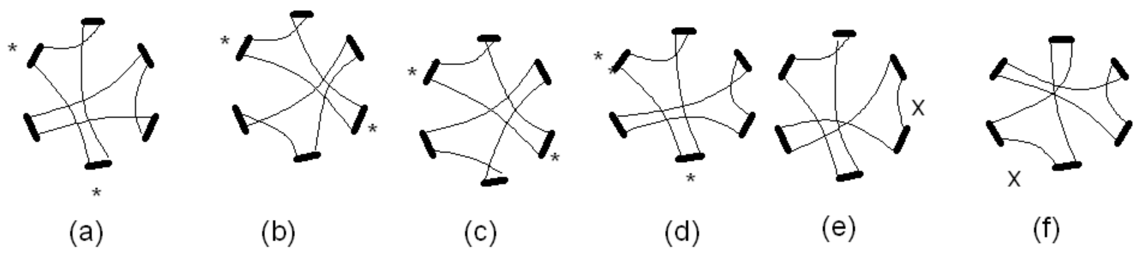


Figure 4.10: Closed configurations of two intersecting 1-twisted cycles.

The following two lemmas deal with closed configurations that involve two intersecting cycles, one of which is 0-twisted and the other 1-twisted.

Lemma 31 *Let π be a permutation that contains a 0-twisted cycle C that intersects with the coupled edge pairs of two non-intersecting, 1-twisted cycles D and E. Then π admits a $(0, 2, 2)$-sequence.*

Proof: Since cycles D and E both intersect with C, there is an arc a induced by two black edges of C that includes edges from both D and E. We apply a 0-transreversal

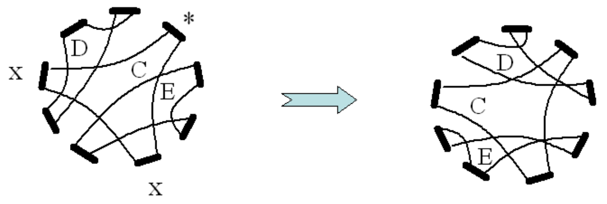


Figure 4.11: One possible scenario of Lemma 31. C is a 0-twisted cycle that intersects with the coupled edge pairs of two non-intersecting, 1-twisted cycles D and E. First we apply a 0-transreversal on the three marked edges. Now both D and E become oriented and remain non-intersecting, allowing two 2-transreversals.

on the edges of C that reverses a. In the resulting permutation, D and E are both 2-twisted and remain non-intersecting, implying two 2-transreversals. An example of such a $(0, 2, 2)$-sequence is given in Figure 4.11. ■

Lemma 32 *Let π be a permutation that contains a 0-twisted cycle, which intersects with the coupled edges of a 1-twisted cycle. Then π admits a* $(0, 2, 2)$*-sequence.*

Proof: There are two possible configurations of a 0-twisted cycle C that intersects the coupled edges of a 1-twisted cycle D, depicted in Figure 4.12. By Lemma 20, there is another cycle E that has a black edge in the arc marked by x, and in some other arc(s) (see Figure 4.12). If E is 0-twisted then a $(0, 2, 2)$-sequence follows from Lemma 26. Otherwise, E is 1-twisted and $(0, 2, 2)$-sequences for all possible configurations are summarized in Table 4.1. ■

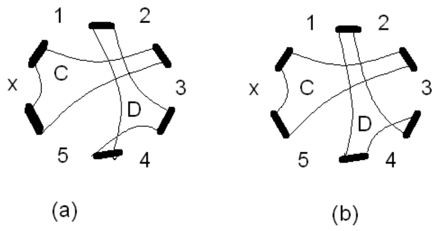


Figure 4.12: Closed configurations consisting of two intersecting cycles, such that one is 0-twisted and the other is 1-twisted.

The final configuration that needs to be taken care of involves 1-twisted pairs of interleaving cycles. This case is handled by the following lemma:

Lemma 33 *Let π be a permutation that contains $k \geq 2$ mutually interleaving 1-twisted*

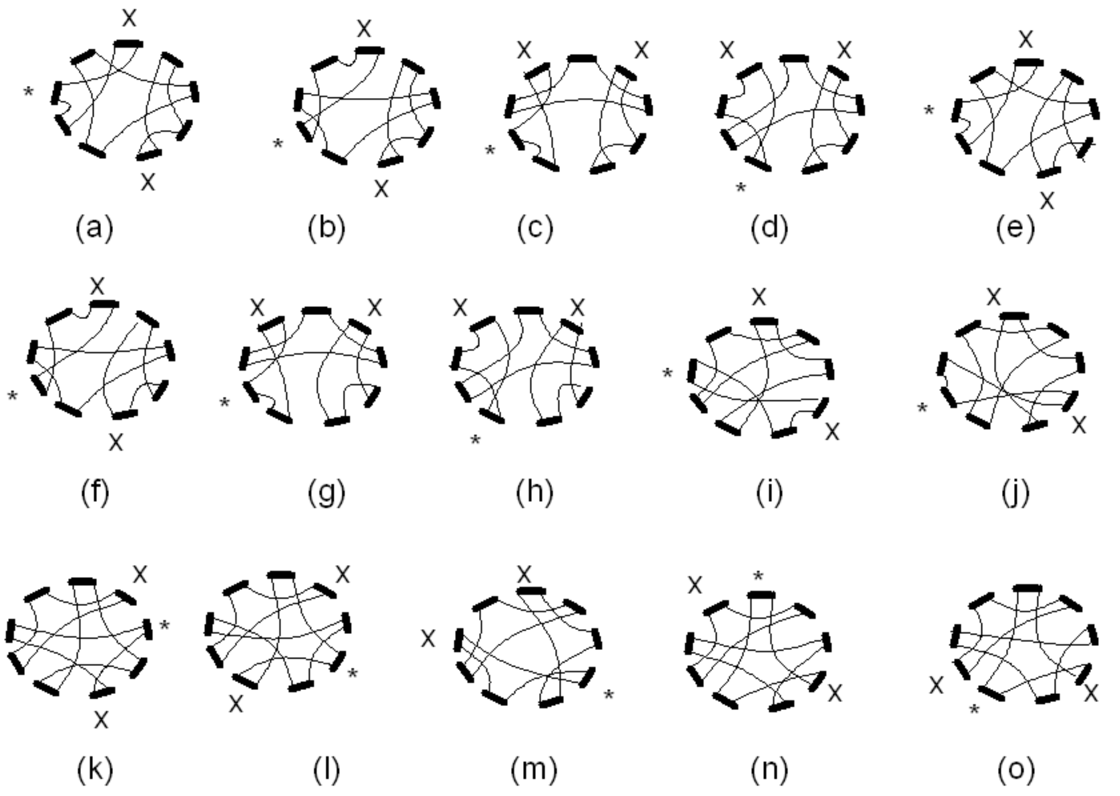


Figure 4.13: $(0, 2, 2)$-sequences for configurations described in Table 4.1.

cycles, such that all their twists are consecutive on the circle and k is maximal with this property. Then π admits a $(0, 2, 2)$-sequence.

Proof: Such an arrangement is possible iff the edges of all cycles alternate along the circle (see Figure 4.14). Consider the arc induced by the twist of the k-th cycle and the first non-twisted edge of the first cycle; further consider the arc induced by the twist of the first cycle and the second non-twisted edge of the k-th cycle. These two arcs (marked by an asterisk in Figure 4.14) are adjacent, so by Lemma 21 there is a cycle C that has a black edge in one of these arcs, and another black edge in some other arc.

Now, if C interleaves with one of the k interleaving cycles then a $(0, 2, 2)$-sequence follows from Lemma 25 (the case in which C is 1-twisted and interleaves with all k cycles is impossible, since it contradicts the maximality of k). If C has an edge that lies between the two non-twists of one of the k cycles, then these two intersecting cycles form a closed configuration; a $(0, 2, 2)$-sequence then follows from Lemma 32, if C is 0-twisted (resp., Lemma 30, if C is 1-twisted). If C is 1-twisted, has an edge between two twists of the k

cycles and forms a closed configuration with one of the cycles it intersects, then a $(0, 2, 2)$-sequence follows Lemma 30. There are five other configurations that are not covered by these cases. The five configurations and $(0, 2, 2)$-sequences for them are depicted in Figure 4.15. ■

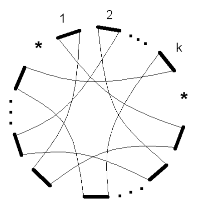


Figure 4.14: Mutually interleaving 1-twisted cycles.

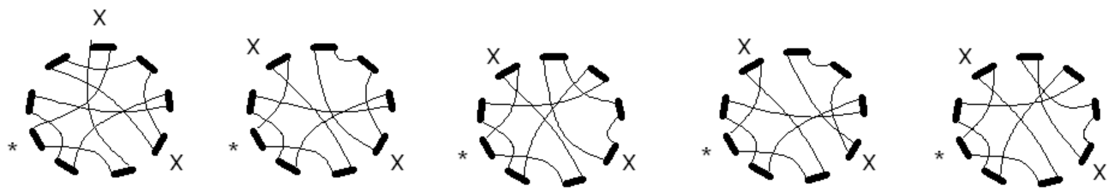


Figure 4.15: $(0, 2, 2)$-sequences for some closed configurations that involve a 1-twisted pair and a third cycle that intersects at least one of the other cycles.

We are now ready to describe our sorting algorithm, which is given in Figure 4.16. For 3-permutations, Step 2 suffices for the sorting and we call this part *Algorithm Sort3Perm*. The following lemma shows that Algorithm Sort3Perm is a quadratic 1.5-approximation algorithm for sorting 3-permutations:

Lemma 34 Sort3Perm *is a 1.5-approximation algorithm for sorting 3-permutations, running in time $O(n^2)$.*

Proof: First, we observe that it suffices to consider only closed configurations, since for configurations that contain only 0-twisted cycles a $(0, 2, 2)$-sequence is given in Section 3.3; in all other cases, by Observation 22 there must exist a closed configuration.

The sequence of operations that is generated by the algorithm contains only 2-operations and $(0, 2, 2)$-sequences of operations. Therefore, every sequence of three operations increases the number of odd cycles by at least 4 out of 6 possible in 3 steps (as implied from the lower bound of Theorem 16). Hence, the approximation ratio is 1.5.

```
Algorithm SortTransrev ($\pi$)

1. Transform $\pi$ into a 3-permutation $\hat{\pi}$ (Lemma 17).
2. While $G(\hat{\pi})$ contains a 3-cycle $C$ do:
    (a) If $C$ is oriented, apply a 2-operation.
    (b) Otherwise, find a cycle $D$ that intersects with a coupled pair of $C$.
    (c) If $C$ and $D$ interleave, apply a $(0,2,2)$-sequence (Lemmas 25, 33).
    (d) Else if $C$ or $D$ are 1-twisted, apply a $(0,2,2)$-sequence (Lemmas 30, 32).
    (e) Otherwise, apply a $(0,2,2)$-sequence (Lemma 26).
    (f) If new 2-cycles were introduced by the last operations, transform $\hat{\pi}$
        into a 3-permutation $\hat{\pi}'$ by safe paddings (Lemma 17), and let $\hat{\pi} = \hat{\pi}'$.
3. Mimic the sorting of $\pi$ using the sorting of $\hat{\pi}$ (Lemma 17).
```

Figure 4.16: Algorithm SortTransrev. After Step 2(a) we assume that all cycles involved in the configuration of C are unoriented. Obviously, if an oriented cycle is involved, then a 2-operation can be applied on it.

We now analyze the running time of the algorithm. The number of iterations in step 2 is linear, since in every iteration two cycles are eliminated and there is a linear number of cycles. Deciding whether two given cycles are interleaving or intersecting can be done in constant time. Thus, the bottleneck in each iteration is to apply an operation to the permutation, and to find an arbitrary cycle that intersects a given coupled pair of black edges (Step 2b). The latter task can be performed by a procedure that finds an arbitrary pair of connected black edges that intersects with a given pair of connected black edges. These tasks can be done easily in linear time. The number of new 2-cycles introduced in each iteration is constant and, therefore, Step 2f takes constant time. Overall, the algorithm can be implemented in quadratic time. ■

Now we are ready to prove the correctness of Algorithm SortTransrev:

Theorem 35 *Algorithm* SortTransrev *is a 1.5-approximation algorithm for sorting arbitrary permutations by transpositions and transreversals, and it runs in time $O(n^2)$.*

Proof: By Lemma 34, we are guaranteed that $alg(\hat{\pi}) \leq 1.5 \cdot trd(\hat{\pi})$, where $alg(\hat{\pi})$ is the number of operations used by Algorithm Sort3Perm to sort $\hat{\pi}$. Thus, by Theorem 16

$$alg(\hat{\pi}) \leq 1.5trd(\hat{\pi}) \leq 1.5\left(\frac{n(\hat{\pi}) - c_{odd}(\hat{\pi})}{2}\right) = 1.5\left(\frac{n(\pi) - c_{odd}(\pi)}{2}\right) \leq 1.5 \cdot trd(\pi)$$

Using Lemma 17, we can sort π by $alg(\hat{\pi})$ operations, which implies an approximation ratio of 1.5.

Since the transformation into 3-permutations can be done in linear time, Lemma 34 implies that the running time of Algorithm Sort is $O(n^2)$. $\blacksquare$

Config. of cycles C & D	2nd black edge of E in arc	3rd black edge of E in arc	Twisted edge in arc	$(0, 2, 2)$-sequence by
a,b	1	2	all cases	Lemma 30
a,b	1 or 2	3 or 4 or 5	all cases	Lemma 25
a	3	4	4	Figure 4.13(k)
a	3	4	x	Figure 4.13(l)
a	3	4	3	Lemma 25
b	3	4	4	Figure 4.13(n)
b	3	4	x	Lemma 25
b	3	4	3	Figure 4.13(o)
a,b	3	5	all cases	Lemma 30
a	4	5	4	Figure 4.13(m)
a	4	5	x or 5	Lemma 30
b	4	5	all cases	Lemma 30
a,b	1,5	same arc	E closed by C	Lemma 31
a	1,5	same arc	E not closed by C	Figure 4.13(a-d)
b	1,5	same arc	E not closed by C	Figure 4.13(e-h)
a,b	2,3	same arc	all cases	Lemma 30
a	4	same arc	E not closed by C	Figure 4.13(i-j)
a	4	same arc	E closed by C	Lemma 30
b	4	same arc	all cases	Lemma 30

Table 4.1: $(0, 2, 2)$-sequences for the two configurations shown in Figure 4.12. Each row of the table describes possible configurations of cycles C, D and E. The first column indicates to which of the two configurations from Figure 4.12 this row refers. As explained in the proof of Lemma 32, cycle E has black edges in arc x and in some other arc(s), with the possible locations listed in the second and third columns. The fourth column specifies the location of the twist of E, completing the description of the configuration of cycles C, D and E. The last column points to a reference for a $(0, 2, 2)$-sequence in each case.

Chapter 5

An $O(n^{3/2}\sqrt{\log n})$ Implementation

In this chapter we introduce a fast implementation of algorithms SortTranspos (Chapter 3) and SortTransrev (Chapter 4). We present an implementation of both algorithms that runs in time $O(n^{3/2}\sqrt{\log n})$, improving on the quadratic running time of previous algorithms. The improvement is achieved by exploiting an efficient data structure introduced by Kaplan and Verbin [2003] in the context of sorting by reversals.

The results of this chapter can be found in [Hartman and Shamir, 2004] and [Hartman and Sharan, 2004a] (preliminary version in [Hartman and Sharan, 2004b]).

As discussed in the proofs of Lemmas 12 and 34, the main tasks in each iteration of both algorithms are finding an arbitrary connected pair that intersects with a given connected pair, and applying an operation. In the sequel we describe a data structure that allows to perform these tasks in sub-linear time. Based on this data structure, each algorithm can be implemented in sub-quadratic time. This data structure is similar to the one introduced by Kaplan and Verbin [2003], although here the required tasks are slightly different. For completeness we give here a full description of the data structure.

By Theorems 1 and 15, the data structure can be presented for linear permutations (we prefer doing that since it makes the presentation clearer). We consider the doubled permutation $f(\pi)$ (see Sections 3.2.2 and 4.2.2), which is denoted here simply by π. A connected pair of black edges (b_1, b_2) is represented by the pair $(2i, 2i+1)$ which corresponds to the gray edge that connects b_1 and b_2. Thus, π is a union of disjoint pairs. Two elements that form a pair are called *mates*. We need a data structure that supports the following tasks in sub-linear time:

- $IntersectionQuery(\pi, e_1, e_2)$: Find a pair that intersects in π with the pair of elements (e_1, e_2).

$$\begin{array}{cccccccccccccccccc}
(& 1 & 2 & 11 & 12 & 9 & 10 & | & 7 & 8 & 13 & 14 & | & 5 & 6 & 3 & 4 &) \\
 & 10 & 13 & 6 & 9 & 8 & 3 & & 12 & 5 & 4 & 1 & & 14 & 7 & 2 & 11 &
\end{array}$$

(a)

$$[\,10\ 11\ 9\ 12\ 1\ 2\,]\quad[\,14\ 13\ 8\ 7\,]\quad[\,3\ 6\ 4\ 5\,]$$

(b)

Figure 5.1: (a) Partition of the permutation $\pi = [1\ 2\ 11\ 12\ 9\ 10\ 7\ 8\ 13\ 14\ 5\ 6\ 3\ 4]$ into blocks. Below each element, the location of its mate in π is indicated. (b) The internal order of each block (according to the order of the mates in π).

- *Operation*(π, e_1, e_2, e_3): Apply an operation on π that cuts before elements e_1, e_2 and e_3 (for reversals $e_2 = e_3$).

A query is said to *act* on the elements e_1 and e_2. Similarly, an operation acts on elements e_1, e_2 and e_3. The *location* of an element e is its number in a left-to-right ordering of π, and is denoted by $loc(e)$.

Now we describe the data structure. The permutation π is divided into $\Theta(\sqrt{\frac{n}{\log n}})$ blocks of size $\Theta(\sqrt{n \log n})$ each, which are maintained in a list. The elements in each block are ordered according to the order of their mates in π (see example in Figure 5.1). Attached to each block is a splay tree [Sleator and Tarjan, 1985] in which the elements of the block are maintained. This data structure is a balanced binary search tree that is re-balanced via rotations, and supports split and concatenate operations in logarithmic time. [1] We also maintain a lookup table that contains for each element a pointer to its block.

In order to maintain signed permutations we introduce a *"reverse" flag* for each node in the splay tree. This flag indicates whether the elements of its subtree should be reversed (in order and sign). The reverse flag of a node can be flipped by exchanging the order of its two children, and flipping their own flags. The ability to clear the reverse flag allows us to implement splits and concatenations, while correctly maintaining the permutation [Kaplan and Verbin, 2003].

[1] As in [Kaplan and Verbin, 2003], we use splay trees since the implementation of the split and concatenate operations is very elegant. Therefore, our running times will be amortized. Amortization can be avoided by using some other type of search tree.

For simplicity, we assume that queries and operations act only on elements that are on block boundaries (Lemma 37 will show why we can make this assumption). More specifically, assume that e_1 and e_2 (for transpositions and transreversals also e_3) are all first elements in their blocks.

Lemma 36 *Tasks IntersectionQuery and Operation can be performed in time $O(\sqrt{n \log n})$, assuming that they act only on elements that are on block boundaries.*

Proof:

- IntersectionQuery(π, e_1, e_2): Let B_1 and B_2 be the blocks that contain e_1 and e_2 respectively, (these blocks are found by using the lookup table) and assume w.l.o.g that B_1 is located before B_2. For each block that is before B_1 or after B_2 do the following. Find, by binary search, the first element that is located after $loc(e_1)$ and the element that is located right before $loc(e_2)$. Split the corresponding tree in the locations of these two elements, and consider the subtree that is bounded by these two elements. If this subtree is not empty, then pick an arbitrary element in it. By construction, this element and its mate are intersecting with (e_1, e_2), i.e., the query is answered. Otherwise, continue to the next block. The split operation is done in logarithmic time. Since there are $O(\sqrt{\frac{n}{\log n}})$ blocks of size $\Theta(\sqrt{n \log n})$, the total time is $O(\sqrt{n \log n})$.
- *Operation*(π, e_1, e_2, e_3): A transposition can be done by applying three reversals, and a transreversal or a revrev can be mimicked by two reversals. Since a reversal takes time $O(\sqrt{n \log n})$ [Kaplan and Verbin, 2003], all operations can be done within the same time bound.

■

The following lemma is based on Kaplan and Verbin [2003], and shows why we can assume that all tasks act only on elements that are at block boundaries:

Lemma 37 *Suppose that it is possible to perform IntersectionQuery and Operation in time $O(\sqrt{n \log n})$, assuming that the tasks act only on block boundaries. Then, it is possible to perform these tasks with the same time complexity, even if they act on arbitrary elements.*

Proof: The idea is to (1) add for each task a pre-processing step that splits up to three blocks, such that in the new partition of blocks the task acts only on boundaries of blocks;

(2) apply procedures *IntersectionQuery* and *Operation* on these blocks, and (3) add a post-processing step that ensures that at the end of the task the blocks are of size between $\frac{\sqrt{n \log n}}{2}$ and $2\sqrt{n \log n}$, and hence, there are still $\Theta(\sqrt{n \log n})$ blocks. We now describe steps (1) and (3), and show that they can be performed in time $O(\sqrt{n \log n})$.

- Pre-processing: Splitting a block is done by splitting the corresponding splay tree in the appropriate location(s), and updating the lookup table accordingly. The splitting locations are the elements that are right before $loc(e_i)$ for $i = 1, 2, 3$, and are found by binary search (for queries we split in two locations). The number of splits is at most three, each split is logarithmic and the number of elements to update is $O(\sqrt{n \log n})$. Thus, this step can be done in time $O(\sqrt{n \log n})$.
- Post-processing: Due to the pre-processing step, there may be up to six blocks that are smaller than $\frac{\sqrt{n \log n}}{2}$. If there is such a block, we first concatenate it to the preceding block (if it is the first block, we concatenate to the last one). Now, it is possible that a block that is bigger than $2\sqrt{n \log n}$ is created. However, since the size of this new block is bounded by $2.5\sqrt{n \log n}$, another single split in this block ensures that the new blocks are of legal size. Concatenating two blocks is done by concatenating the corresponding splay trees, and updating the lookup table accordingly. Since the total number of splits and concatenations in this step is constant and the number of updates is $O(\sqrt{n \log n})$, it can be performed in time $O(\sqrt{n \log n})$.

■

By combining Lemmas 36 and 37 we get:

Corollary 38 *Step 3 of Algorithm SortTranspos (and Step 2 of Algorithm SortTransrev) can be implemented in time $O(\sqrt{n \log n})$.*

Now we are ready for the main result of this chapter :

Theorem 39 *Algorithm SortTranspos (resp. Algorithm Transrev) guarantees a 1.5-approximation ratio to sorting by transpositions (sorting by transpositions and transreversals), and runs in time $O(n^{3/2}\sqrt{\log n})$.*

Proof: The number of iterations in each algorithm is at most n. By Corollary 38, each iteration can be implemented in time $O(\sqrt{n \log n})$. Thus, both algorithms runs in time $O(n^{3/2}\sqrt{\log n})$. ■

Part II

Extracting Sequence Information from Oligonucleotide Arrays

Chapter 6

Sequencing by Hybridization with Errors

Sequencing by hybridization (SBH) is a DNA sequencing technique, in which the sequence is reconstructed using its k-mer content. This content, which is called the *spectrum* of the sequence, is obtained by hybridization to a universal DNA array. Standard universal arrays contain all k-mers for some fixed k, typically 8 to 10. Currently, in spite of its promise and elegance, SBH is not competitive with standard gel-based sequencing methods. This is due to two main reasons: lack of tools to handle realistic levels of hybridization errors, and an inherent limitation on the length of uniquely reconstructible sequence by standard universal arrays.

In this chapter we deal with both problems. We introduce a simple polynomial reconstruction algorithm which can be applied to spectra from standard arrays and has provable performance in the presence of both false negative and false positive errors. We also propose a novel design of chips containing universal bases, that differs from the one proposed in [Preparata et al., 1999, Preparata and Upfal, 2000]. We give a simple algorithm that uses spectra from such chips to reconstruct with high probability random sequences of length lower only by a squared log factor compared to the information theoretic bound. Our algorithm is very robust to errors, and has a provable performance even if there are both false negative and false positive errors. Simulations indicate that its sensitivity to errors is also very small in practice.

The results in this chapter are published in [Halperin et al., 2003] (preliminary version in [Halperin et al., 2002]).

6.1 Introduction

Sequencing by Hybridization (SBH) was proposed in the late eighties [Bains and Smith, 1988, Drmanac et al., 1989] as an alternative approach to DNA sequencing. In this technique, a large set of short oligonucleotides (called *probes*) is arrayed on a solid surface, forming a DNA chip. A solution with copies of the target DNA sequence (the sequence we would like to read) is brought in contact with the chip. Oligonucleotides on the chip hybridize with reverse complement segments of the target DNA molecules.Using the hybridization signals, the subset of probes that hybridize with the target (the *spectrum* of the sequence) is detected, and a combinatorial algorithm reconstructs the DNA sequence from its spectrum. For an overview on SBH see [Pevzner, 2000, Pevzner and Lipshutz, 1994, Shamir, 2002].

Initially, SBH was designed for chips with probes consisting of all 4^k strings of length k over the DNA alphabet (A,C,G,T) (we call this design *classical SBH*). Assuming that the spectrum is error-free and that the multiplicity of each k-mer in the spectrum is known, the problem is reduced to finding an Eulerian path in a particular directed graph [Pevzner, 1989] and therefore is tractable. However, these two assumptions are not practical.

Real hybridizations are error prone. They contain both *false negative* errors (i.e., probes that match the sequence but are missing from the spectrum), as well as *false positives* (probes in the experimental spectrum that do not match any position in the sequence). For the case of false negatives only, several algorithms are proposed [Blazewicz et al., 1997, Pevzner, 1989]. When there are also false positive errors, there are three known algorithms. Lipshutz [1993] provides an iterative algorithm which assumes some knowledge of the error rates, and assumes an error model in which false positives can be obtained only by mismatches of a base at one end of the k-mer. The algorithm may not always converge, but simulations indicate that this happens with low probability. Blazewicz et al. [1999a,b] give two algorithms that minimize the number of errors and have worst case exponential running time. None of the algorithms that was previously proposed to address both types of errors in classical SBH has a theoretical proof that with high probability, the obtained sequence is the original one.

Another main obstacle is that often reconstruction is not unique. Theoretical studies [Arratia et al., 1996, Dyer et al., 1994, Shamir and Tsur, 2002] prove that the expected length of unambiguously reconstructible sequences by classical SBH with probes of length k is $O(2^k)$. This was also observed empirically [Pevzner and Lipshutz, 1994, Pevzner et al., 1991]. In contrast, a simple information-theoretic argument [Preparata et al., 1999] yields an upper bound of $O(4^k)$ if there are no restrictions on the set of probes, that is, the length of the target sequence may be no more than linearly proportional to the number of probes on the DNA chip.

Several approaches were proposed to decrease the probability of ambiguous reconstruction. Among them are alternative chip designs [Pevzner et al., 1991, Preparata et al., 1999, Preparata and Upfal, 2000], interactive protocols [Frieze and Halldorsson, 2002, Skiena and Sundaram, 1995, Phan and Skiena, 2001], using additional positional information [Ben-Dor et al., 2001, Hannenhalli et al., 1996, Shamir and Tsur, 2002], using a homologous reference sequence [Pe'er and Shamir, 2000] and resolving ambiguities by restriction enzymes [Snir et al., 2002]. An important breakthrough result due to [Preparata et al., 1999, Preparata and Upfal, 2000] uses *universal bases*, i.e., bases which hybridize with each of the four standard DNA bases (Probes containing universal bases are also called *gapped*, as the universal bases form gaps of unspecified positions between specified ones). Preparata et al. have given a new chip design based on probes which contain universal bases and is provably optimal (i.e., achieves the information theoretic bound) up to a constant factor (all results here and below hold with high probability for random sequences of independent, equiprobable bases).

In this chapter we deal with the problem of noise in SBH, as well as with the problem of ambiguous reconstruction. We first provide a polynomial algorithm which solves the classical SBH problem with both false negative and false positive errors. Our algorithm does not assume knowledge of the spectrum multiplicities. If the false positive rate is small and the probability of false negative is q, the probability that our algorithm fails to reconstruct a random sequence of length $O(2^{(1-3q)k})$ is bounded by an arbitrarily small constant. To the best of our knowledge, this is the first rigorous theoretical analysis of a polynomial algorithm for SBH with positive and negative errors. This answers the challenge posed in [Pevzner and Waterman, 1995, Problem 35].

We also provide an alternative chip design to the one of Preparata et al. [1999], Preparata and Upfal [2000] which uses universal bases. In contrast to the latter design, which has a deterministic periodic structure, we use *randomized probes*, in which the locations of the real (specified) bases among the universal ones are chosen randomly. We provide a second polynomial algorithm to reconstruct the target from such spectrum. Here too we do not assume that the multiplicities are known. The probability that our algorithm fails to reconstruct a random sequence of length $O(\frac{4^k}{k})$, using $\Theta(k4^k)$ probes, is bounded by an arbitrarily small constant, assuming there are no hybridization errors. Hence, our design is optimal up to a squared log factor.

The main advantage of our randomized probes design is that the same algorithm has provable performance in the presence of errors. If the false positive rate is small and the probability of false negative is q, the probability that our algorithm fails to reconstruct a random sequence of length $O((1-q)\frac{4^k}{k})$, using $\Theta(\frac{k4^k}{1-q})$ probes, is bounded by an arbitrarily small constant. This is the first rigorous theoretical analysis of any algorithm for SBH with gapped probes in the presence of errors: The analyses of [Preparata et al., 1999, Preparata and Upfal, 2000] assumes error-free data. Doi and Imai [2000] report on an empirical study on an extension of the algorithms of [Preparata et al., 1999, Preparata and Upfal, 2000] which handles errors. Furthermore, our simulations show that our algorithm is much more resistant to errors than both the algorithms of Preparata et al. and of Doi and Imai [2000].

6.1.1 Organization of the Chapter

In Section 6.2 we introduce the model, some definitions and useful lemmas. In Section 6.3 we describe and analyze the algorithm for classical SBH arrays. In Section 6.4 we introduce the new array design that uses universal bases and analyze its reconstruction algorithm. In Section 6.5 we describe our empirical study, and we conclude with some possible extensions in Section 6.6.

6.2 Preliminaries

In this section we give some basic definitions, describe our model, and state some known lemmas that we are going to use in the analysis.

Let $\Sigma = \{A, G, C, T\}$, where A, G, C and T represent (specified) nucleotide bases. Following [Preparata et al., 1999, Preparata and Upfal, 2000], we add a "don't care" symbol N, implemented using a universal base [Loakes and Brown, 1990] (biologically, a universal base can hybridize to any base). The DNA sequence that we wish to reconstruct, also called the *target*, consists of specified bases only. A *probe* is a short sequence of universal and specified bases. A sequence that contains universal bases will sometimes be referred to as the gapped subsequence of its specified locations. Hence, AANNTNC is the same as the subsequence AA**T*C.

Definition 5 (Matching Probe) *Probe a* matches *sequence S if it appears contiguously in S.*

For example, a =AANNTNC matches TAAGCTGCC. Note that the probe must be fully contained in S, so a does not match AACGTA.

A DNA array is a set of probes (typically of equal length). For a given array, the *theoretical spectrum* of a target is the set of all the array probes that match the target.

Note that we ignore multiplicities in the spectrum, so spectra are viewed as sets and not as multisets.

Definition 6 (Fooling Probe) *For a sequence S, which is not a subsequence of the target, a probe is called a* fooling probe *if it matches both S and the target.*

The (experimental) *spectrum* $\mathcal{ES}$ of a target is the set of probes in the array that were recorded by the experiment as present in the target. For errorless data, the theoretical and experimental spectra coincide. We model noise in the SBH experiment by adding some false positives and some false negatives to the theoretical spectrum $\mathcal{T}$: For every probe $a \in A$ independently, if $a \notin \mathcal{T}$ then $a \in \mathcal{ES}$ (i.e., a appears in the spectrum) with probability p. This is the false positive probability. If $a \in \mathcal{T}$ then a appears in the spectrum $\mathcal{ES}$ with probability $1 - q$, so q is the false negative probability. This probability is independent of the number of locations that a matches. Since the probability that a probe matches more than one location is very low, this model approximates quite well a model which assumes that false negative errors occur independently at each location. We assume that p is small enough so that the number of probes in the spectrum is dominated by the probes that do belong to $\mathcal{T}$.

Definition 7 (Supporting Probe) *A probe a* supports *a sequence S if it matches S and appears in the spectrum.*

Any probe that is a subsequence of the target is by definition a matching probe and, in case there are no false negative errors, also a supporting probe. For example, given errorless k-mer spectrum from a classical array, there will typically be k supporting probes for any subsequence of the target of length $2k - 1$ (if the subsequence is degenerate there may be fewer). When errors are introduced, not all matching probes support a true subsequence, due to false negatives. A sequence which is not a subsequence of the target may be supported by fooling probes (which are not false negatives) and by false positive probes (that do not appear in the theoretical spectrum). Note that the latter are not fooling probes.

The *SBH problem* is to reconstruct the target S from the spectrum $\mathcal{ES}$. We assume we are given a prefix of S, of length $i - 1$, where i is the length of the probes. Note that there are biochemical methods to fulfill the prefix requirement (see [Preparata et al., 1999]). We assume that S is chosen uniformly from all the DNA strings of a determined length m, i.e., each symbol is chosen uniformly and independently.

In our analysis, we will use the following Chernoff-like upper bounds for large deviations (their proof can be found, e.g., in [Alon and Spencer., 2000, Appendix A]):

Lemma 40 *Let Z be a random variable with a binomial distribution $Z \sim B(n,p)$. For every $\lambda > 1$, $Pr(Z > \lambda pn) < \left(e^{\lambda-1}\lambda^{-\lambda}\right)^{pn}$.*

Lemma 41 *Let Z be a random variable with the binomial distribution, $Z \sim B(n,p)$. For every $a > 0$, $Pr[Z - np < -a] < e^{-a^2/2pn}$.*

6.3 Classical SBH with Errors

In this section we assume that the array consists of all 4^k possible sequences of length $k \geq 6$, over the alphabet $\Sigma = \{A, G, C, T\}$. When there are neither false positives nor false negatives, it is shown [Dyer et al., 1994, Arratia et al., 1996, Shamir and Tsur, 2002] that with high probability, the longest possible sequences to be uniquely reconstructed, are of length $O(2^k)$. In this section, we address the errors-prone case. We describe a simple algorithm, and show that it reconstructs (with high probability) sequences of length $\Theta(2^{k(1-3q)})$, for $q < \frac{1}{3}$. Notice that in the error-free case, when $q = 0$, our algorithm is optimal up to a constant. As the number of probes is 4^k, and the length of the sequence is $O(2^k)$, we assume that $p < \frac{1}{2^k}$. Note that the length of the reconstructed sequences is not affected by p.

Let the input sequence be $s_1, \ldots, s_m$. Suppose that we already know the sequence $s_1, \ldots, s_i$, and we want to find s_{i+1}. The reconstruction algorithm is described in Figure 6.1.

1. Enumerate all 4^k sequences $a = a_1, \ldots, a_k$.
2. Pick a sequence a' such that the number of probes supporting $s_{i-k+2}, \ldots, s_i, a'_1, \ldots, a'_k$ is maximal (breaking ties arbitrarily).
3. Set $s_{i+1} = a'_1$.

Figure 6.1: Algorithm A for Classical SBH.

The running time of algorithm A is $O(mk4^k)$. Each of the $(2k-1)$-long sequences tried will be called a *path*.

6.3.1 Analysis of the Algorithm

Theorem 42 *The probability that algorithm A fails is bounded by an arbitrary small constant, if $m = O(2^{(1-3q)k})$.*

Proof: At each step, the algorithm tries to extend the current sequence by all possible sequences of length k. A sequence path $s_{i-k+2}, \ldots, s_i, a'_1, \ldots, a'_k$ in which the first new base (corresponding to a'_1 in algorithm A) is wrong, will be called a *bad path.* Let us fix a possible bad path. Denote by P_{bad} the probability that the number of supporting probes of this bad path is greater or equal to the number of probes which support the correct path (and hence in this case the algorithm fails). There are $\frac{3}{4}4^k$ possible bad paths, and the number of possible locations that might lead to an error is bounded by m. We have to show that the failure probability is bounded by a constant $\epsilon < 1$. Hence, what we need to prove is that

$$P_{bad} \cdot \frac{3}{4} 4^k \cdot m \leq \epsilon$$

Let X be a random variable that counts the number of supporting probes for the correct path, and let Y be the number of supporting probes for the bad path. Clearly, $Y = Y_1 + Y_2$, where Y_1 is the number of supporting probes for the bad path that are fooling (probes that appear also in the theoretical spectrum), and Y_2 is the number of supporting probes arising from false positives. The algorithm fails if at some point $X \leq Y$.

Suppose there were i fooling probes for the bad path. It is easy to see that $X \sim B(k, 1-q)$, $Y_1 \sim B(i, 1-q)$, and $Y_2 \sim B(k-i, p)$. Note that these random variables depend on i. However, the value of i will be clear from the context, so the random variables are not written as functions of i.

We first prove the following lemma:

Lemma 43 *The probability that a bad path has i fooling probes is at most* $2(k-i+1) \cdot m \cdot 4^{-k-i}$.

Proof: In the bad path, the first extension base is incorrect and therefore, all i fooling probes match other locations in the sequence. (Note that for a false path that starts with $1 \leq j \leq k$ correct bases - and hence is not called a bad path - the first j probes that support such path match also the correct path. Thus, they appear in the theoretical spectrum even if they do not match other locations in the sequence.)

Consider first the case where the i fooling probes of the bad path all match a single location (with appropriate shifts), which means that $k-1+i$ consecutive symbols of the bad path appear consecutively elsewhere in the sequence. This happens with probability at most $(k-i+1) \cdot \frac{m}{4^{k-1+i}}$ (there are $k-i$ possible sequences of $k-1+i$ consecutive symbols in the bad path).

The more general case is that the fooling probes match $x \geq 1$ different locations. In this case there are x different subsequences of the bad path of lengths $k-1+i_1,\ldots,k-1+i_x$, $\sum_{j=1}^{x} i_j = i$ that appear elsewhere in the sequence. There are $\binom{i-1}{x-1}$ possibilities of such decompositions of i into i_j's (the same as the number of possibilities of choosing a multiset of $i-x$ elements out of x elements). There are $kx+i$ restricted symbols, at most $\binom{m}{x}$ possible sets of locations in the sequence and at most $(k-i+x)^x$ possible sets of indices in the bad path. (Note that the fact that the occurrences of the probes in the sequence may overlap does not affect the analysis, since the number of restricted symbols remains the same.) Therefore the probability that there are i fooling probes is at most

$$\sum_{x=1}^{i} \binom{i-1}{x-1}\binom{m}{x}(k-i+x)^x\, 4^{-kx-i} \leq$$

$$\frac{4^{-i}}{i}\sum_{x=1}^{i}\left(i(k-i+x)\cdot m\cdot 4^{-k}\right)^x \leq 2(k-i+1)\cdot m\cdot 4^{-k-i}$$

The last inequality is obtained since for $k \geq 6$, the first summand (the expression for $x=1$) is bounded by $\frac{1}{2}$ (note that $x \leq i$), so the whole sum is bounded by twice the first summand. ■

Summing over the possible values of i, and using Lemma 43, we obtain

$$\begin{aligned} P_{bad} &= \sum_{i=0}^{k} Pr[X \leq Y]\cdot Pr[i\ fooling] \\ &\leq \sum_{i=0}^{k} Pr[X \leq Y]\cdot 2(k-i+1)m\cdot 4^{-k-i} \end{aligned}$$

By denoting $f(i) \stackrel{\text{def}}{=} Pr[X \leq Y]\cdot 2(k-i+1)m^2\cdot 4^{-i}$ we get

$$m\cdot 4^k\cdot P_{bad} = \sum_{i=0}^{(1-2q)k} f(i) + \sum_{i=(1-2q)k+1}^{k} f(i) \qquad (6.1)$$

which has to be bounded by $\frac{4}{3}\epsilon$.

We will bound separately the first and the second sum. Roughly speaking, the second sum is bounded using the fact that the number of fooling probes i is high, an event which happens with very low probability. The first sum is bounded by observing that when i is small, it is unlikely that the number of probes that support the correct path is close to i, since it is far from the expectation.

Let $\rho = m2^{-(1-3q)k}$. We will first bound the second sum in (6.1). In order to bound $Pr[X \leq Y]$ we only use the fact that $Pr[X \leq Y] \leq \frac{1}{2}$, and we get

$$\sum_{i=(1-2q)k+1}^{k} f(i) \leq 2(qk) \max_{i>(1-2q)k} f(i) \leq$$

$$4(qk)^2m^2 \cdot 4^{-k(1-2q)} \le \rho^2, \tag{6.2}$$

where the last inequality holds since for every integer $x \ge 0$, $2x^2 \le 4^x$, and therefore, $4(qk)^2 \le 4^{qk}$.

In order to bound the first sum in (6.1), we will first fix $i \le (1-2q)k$, and bound $f(i)$. For every $j > i$,

$$Pr[X \le Y] \le Pr[X \le j] + Pr[Y \ge j]$$

So our goal is to find a number j, such that $Pr[X \le j] + Pr[Y \ge j]$ is very small.

We will first bound $Pr[X \le j]$. For simplicity of notation, let $x = \frac{k-i+1}{kq}$, $y = \frac{k-j}{kq}$, and $\delta = 1 - \frac{y}{x}$, and thus, $i = k - kqx - 1, j = k - kqy$, and $y = x(1-\delta)$. By applying Lemma 40 for $Z = k - X$, and for $\lambda = y$, we have

$$\begin{aligned} Pr[X \le j] &= Pr[k - X \ge yqk] \\ &< \left(\frac{e^{y-1}}{y^y}\right)^{qk} = \left(\frac{1}{e}\left(\frac{e}{y}\right)^y\right)^{qk} \end{aligned} \tag{6.3}$$

In order to bound $Pr[Y \ge j]$, we first note that $Y_1 \le i$ is always true, and thus it is sufficient to bound the probability $Pr(Y_2 \ge j-i)$. We can now apply again Lemma 40 for Y_2 with $\lambda = \frac{j-i}{(k-i)p} = \frac{\delta}{p}$, and get that

$$\begin{aligned} Pr[Y \ge j] &\le Pr[Y_2 \ge j - i] \\ &< \left(\frac{e^{(\delta-p)/p}p^{\delta/p}}{\delta^{\delta/p}}\right)^{xqkp} < \left(\frac{ep}{\delta}\right)^{\delta xqk} \end{aligned} \tag{6.4}$$

Denote by $\gamma = qk$. Define

$$g(x,\gamma) \stackrel{\text{def}}{=} 2(k-i+1)m^2 4^{-i} Pr[X \le j]$$

and

$$h(x,\gamma) \stackrel{\text{def}}{=} 2(k-i+1)m^2 4^{-i} Pr[Y \ge j],$$

such that $f(i) \le g(x,\gamma) + h(x,\gamma)$.

- From equation (6.3), we get that

$$\begin{aligned} g(x,\gamma) &= 2(k-i+1)m^2 4^{-i} Pr[X \le j] \\ &= 2xqk\rho^2 4^{(x-3)qk} Pr[X \le j] \end{aligned}$$

$$\begin{aligned} &< 2\rho^2 x\gamma \left(\frac{4^{x-3}}{e}\left(\frac{e}{y}\right)^y\right)^\gamma \\ &\leq 2\rho^2 x \left(0.54\cdot 4^{x-3}\left(\frac{e}{y}\right)^y\right)^\gamma, \end{aligned}$$

where the last inequality holds since $\gamma^{1/\gamma} \leq 1.45$. Also, note that $g(x,\gamma)$ only depends on x and γ for given ϵ and ρ.

- From equation (6.4), we get

$$\begin{aligned} h(x,\gamma) &= 2(k-i+1)m^2 4^{-i} Pr[Y \geq j] \\ &= 2xqk\rho^2 4^{(x-3)qk} Pr[Y \geq j] \\ &\leq 2\rho^2 x\gamma \left(\frac{1}{64}\left(4\left(ep/\delta\right)^\delta\right)^x\right)^\gamma \\ &\leq 2\rho^2 x \left(\frac{1.45}{64}\left(4\left(ep/\delta\right)^\delta\right)^x\right)^\gamma \end{aligned}$$

When we set $\delta = 0.05$, we get

$$g(x,\gamma) \leq 2\rho^2 x \left(\frac{0.54}{64}\right)^\gamma \left(\frac{11}{x^{0.95}}\right)^{x\gamma},$$

$$h(x,\gamma) \leq 2\rho^2 x \left(\frac{1.45}{64}\right)^\gamma \left(5p^{0.05}\right)^{x\gamma}$$

We first assume that $\gamma \geq 1$. It is easy to verify that $g(x,\gamma) \leq 9\rho^2 0.95^{x\gamma}$, for $\gamma > 1, x \geq 2$, and that $h(x,\gamma) \leq \rho^2 0.95^{x\gamma}$ when $5p^{0.05} \leq 1$. As $p < \frac{1}{2^k}$, p is sufficiently small, for sufficiently large k (See Remark 44 regarding how to relax the restriction on p).

From the analysis above, we get that $f(i) < 10\rho^2 0.95^{k-i}$, and thus,

$$\sum_{i \leq (1-2q)k} f(i) \leq 10\rho^2 \sum_{i=2}^{\infty} 0.95^i = 200\rho^2 \tag{6.5}$$

Putting together equations (6.2) and (6.5), and letting $\rho = \sqrt{\frac{\epsilon}{151}}$, we get

$$Pr[failure] \leq P_{bad} \cdot \frac{3}{4} 4^k \cdot m \leq \frac{3}{4} 201\rho^2 < \epsilon$$

Hence, for every $0 < \epsilon < 1$, if $m \leq \sqrt{\frac{\epsilon}{151}} 2^{(1-3q)k}$ then the failure probability is bounded by ϵ.

The case where $\gamma < 1$ is trivial, as we can add artificial false negatives so that the false negative rate will be $\frac{1}{k}$, and then, we can recover a sequence of length $\rho 2^{k(1-3/k)} = \frac{\rho}{8}2^k$. ■

Remark 44 We note that the restriction that $5p^{0.05} \leq 1$ can be relaxed, by compromising on other parameters. For example, one can verify that by using $\delta = 0.5$, we get that all we need in order to show that $h(x, \gamma)$ is small is that $p < 0.01$, and to bound $g(x, \gamma)$, it is sufficient to use $m = \rho \cdot 2^{k(1-12q)}$, and to use $x \geq 11$. For $x < 11$, we can bound the sum similarly to equation (6.2). Thus, in this way we get a shorter reconstructed sequence in terms of q, but we can tolerate a constant frequency of false positives.

Remark 45 In case there are only false negative errors, a similar analysis shows that the length of the sequence is asymptotically the same. However, the hidden constant is significantly smaller. It is clear from the proof of Theorem 42 that in the case that there are only false positives, p could be quite large, with a small constant factor loss in the length of the reconstructed sequence.

6.4 A Randomized Chip Design Using Universal Bases

In this section we describe a new array design, over the alphabet $\{A, G, C, T, N\}$, that is, this time we allow gapped probes. Preparata et al. [1999], Preparata and Upfal [2000] describe a set of $\Theta(4^k)$ gapped probes, and showed how a randomly chosen sequence of size $m = \Theta(4^k)$ can be determined unambiguously with high probability, in the absence of errors. We shall prove that our new design is noise resistant, i.e., false positives or false negatives have little affect on the length of constructible sequences.

We build *randomized* probes, and show that a sequence of length $m \approx \frac{4^k}{k}$ can be determined unambiguously, with high probability, using a simple algorithm, based on a spectrum of a set of $\Theta(k4^k)$ probes, in the presence of errors. Actually, we take $m = \alpha\frac{4^k}{\beta k}$ for some universal constant α and some constant β that depends on the error rates.

We choose a set of probes in the following way. Let c be a sufficiently large integer. The length of each probe is $c \cdot k + 1$. The last symbol of the probe is a *specified* base, that is, A, G, C or T. First we form βk subsets of probes $A_1, \ldots, A_{\beta k}$. For each subset, we pick a random set of k positions out of $\{1, \ldots, ck\}$ and form all possible 4^{k+1} probes with specified bases in the chosen positions and in the last one, and the rest universal. Our overall set of probes is the union of the above subsets. Thus, the total number of probes we use is $n = \beta k 4^{k+1}$.

We assume that we are given a prefix of length ck of the sequence. Let the input sequence be $s_1, \ldots, s_m$. Suppose that we already know the sequence $s_1, \ldots, s_i$, and we want to find s_{i+1}. The reconstruction algorithm is described in Figure 6.2.

```
1. For each specified base $X$, count how many probes support $s_{i-ck+1}, \ldots, s_i, X$.

2. Set $s_{i+1}$ to be a base that has maximum support (breaking ties arbitrarily).
```

Figure 6.2: Algorithm B for SBH with universal bases.

As to the running time of algorithm B, at each step we have to check 4 probes from each of the βk subsets (one for each possible specified base in the probe's last position). Therefore, the total running time is $O(km)$.

6.4.1 Bounding Probability of Fooling Probes

The following definitions will be useful in the proofs of this section. We will extend definition 5: Probe *a partially matches* sequence S if its first ck bases (excluding the last specified base) appear contiguously in S. Probe *a matches* sequence S *in location* l if it appears contiguously in S, ending at location l.

Assume we are trying to extend the sequence at location l. In the absence of errors, a probe that supports a false base extension has to satisfy two conditions:

1. It partially matches S in location l.
2. It matches S in some other location l'.

Hence, it is a fooling probe. Notice that when errors are introduced, a supporting probe may be a false positive that does not satisfy the second condition.

Let S_x be the set of k indices of the specified bases of probe p_x, excluding the last specified base. We say that the two probes p_1 and p_2 *agree* if the values of the specified bases in indices $S_1 \cap S_2$ are the same in both probes. A probe $a = a_1, \ldots, a_{ck+1}$ Δ-*self-agrees* if for every position $1 \leq j \leq ck - \Delta$, $a_j = a_{j+\Delta}$ (using the convention that the universal base N equals to all bases). Note that the last specified base (at position $ck+1$) is not required to match the corresponding position. Note also that for $\Delta \geq ck$, every probe Δ-self-agrees.

Claim 46 *For each step of algorithm B, the probability that a false base will get one fooling probe is bounded by α.*

Proof: The fooling probe partially matches S at location l, and matches S in some other location l'. Let $\Delta = |l' - l|$. Obviously, a necessary condition is that it Δ-self-agrees.

We use the notation $S_x + y$ to denote the set S_x shifted by y. Let $i = |S_x \cap (S_x + \Delta)|$. In each one of the βk sets of probes, there are exactly 4^{k+1-i} probes which Δ-self-agree, since there are i constrained symbols. A probe is a fooling probe if it Δ-self-agrees, the $|S_x \cup (S_x + \Delta)| = 2k + 1 - i$ relevant specified bases match the sequence, and the base in location l' is different from the one in location l. The latter two events, happen with probability $\frac{3}{4}\frac{1}{4^{2k+1-i}}$.

We multiply this probability by the number of possible locations l' and by the total number of Δ-self-agreeing probes, and get that the probability that there is at least one fooling probe in the l^{th} location is

$$\frac{3}{4}m \cdot \frac{1}{4^{2k+1-i}} \cdot \beta k 4^{k+1-i} \leq \alpha$$

Notice that this probability is independent of i and β. ■

Claim 47 *The probability that a false base will get two fooling probes that arise from the same location l' is bounded by $\frac{exp(4k/(c-1))}{4^{4k}}$.*

Proof: Let the two fooling probes be p_1 and p_2. Assume first that $|l' - l| > ck$, and fix $i = |S_1 \cap S_2|$. Both probes partially match location l, and match location l'. Thus,

- p_1 and p_2 agree. This happens with probability $\frac{1}{4^i}$.
- The $|S_1 \cup S_2|$ specified bases of both probes p_1 and p_2 match the appropriate symbols in both locations l and l'. This happens with probability $\frac{1}{4^{2|S_1 \cup S_2|}} = \frac{1}{4^{4k-2i}}$.

Thus, the probability of this event, given $|S_1 \cap S_2| = i$, is $\frac{1}{4^{4k-i}}$.

For the case $|l' - l| \leq ck$, if p_1 and p_2 agree, we can consider a new "meta-probe" with specified bases in locations $S_1 \cup S_2$ with values induced naturally by the values of the specified bases of p_1 and p_2. By combining the above argument with the argument in the proof of claim 46, we get that in this case, the probability is $\frac{1}{4^{4k-2i}}$.

Now we will sum over all the values of i to get the desired probability. Notice first, that

$$Pr[|S_1 \cap S_2| = i] = \frac{\binom{k}{i}\binom{(c-1)k}{k-i}}{\binom{ck}{k}}. \tag{6.6}$$

Also notice that

$$\frac{\binom{k}{i}\binom{(c-1)k}{k-i}}{\binom{ck}{k}} \leq \frac{k^i}{i!}\frac{((c-1)k)^{k-i}}{(k-i)!}\frac{k!}{((c-1)k)^k} \leq \frac{k^i}{i!(c-1)^i}. \tag{6.7}$$

Now, by equations (6.6) and (6.7), we get

$$Pr[probes\ match] =$$

$$\sum_{i=0}^{k} Pr[probes\ match \mid |S_1 \cap S_2| = i] \cdot Pr[|S_1 \cap S_2| = i]$$

$$\leq \frac{1}{4^{4k}} \sum_{i=0}^{k} \frac{4^i k^i}{i!(c-1)^i}$$

$$\leq \frac{1}{4^{4k}} \sum_{i=0}^{\infty} \frac{1}{i!} (\frac{4k}{c-1})^i = \frac{exp(4k/(c-1))}{4^{4k}}$$

■

Claim 48 *For each step of algorithm B, the probability that a false base will get two fooling probes is bounded by α^2.*

Proof: Each fooling probe matches some location in the sequence. It could be the case that each probe matches a different location - that will happen, by Claim 46, with probability at most α^2, as these events are independent. Alternatively, they could arise from the same location l'. That is, both probes match the current location and also the same false location. There are at most m possible false locations and $\binom{n}{2}$ possible pairs of probes. Thus, using Claim 47, the probability of the above event is bounded by

$$mn^2 \frac{exp(4k/(c-1))}{4^{4k}} = \beta k \alpha \frac{exp(4k/(c-1))}{4^{k-2}} \leq \alpha^2,$$

for c large enough. ■

Claim 49 *The probability that throughout algorithm B, a false base will get three fooling probes from the same location in the sequence is exponentially small.*

Proof: Denote the three fooling probes by p_1, p_2, p_3. Assume $|S_1 \cap S_2| = i$ and $|(S_3 \cap (S_1 \cup S_2)| = j$. By the same arguments as in as in the proof of Claim 47, we get that the probability that the three probes agree is $\frac{1}{4^{i+j}}$, and the probability that they match both locations (given that they agree) is $\frac{1}{4^{2|S_1 \cup S_2 \cup S_3|}} = \frac{1}{4^{6k-2i-2j}}$.

Summing over all possible values of i and j, and using equations (6.6) and (6.7), we get

$$Pr[the\ 3\ probes\ match] =$$

$$\sum_{i=0}^{k} Pr[|S_1 \cap S_2| = i] \cdot$$

$$\sum_{j=0}^{k} Pr[|(S_3 \cap (S_1 \cup S_2)| = j||S_1 \cap S_2| = i] \cdot \frac{1}{4^{6k-i-j}}$$

$$\leq \frac{1}{4^{6k}} \sum_{i=0}^{k} \frac{1}{i!}(\frac{k}{c-1})^i \sum_{j=0}^{k} \frac{1}{j!}(\frac{2k-i}{c-2})^j \cdot 4^{i+j}$$

$$\leq \frac{1}{4^{6k}} \sum_{i=0}^{k} \frac{1}{i!}(\frac{4k}{c-1})^i \sum_{j=0}^{k} \frac{1}{j!}(\frac{8k}{c-2})^j$$

$$\leq \frac{1}{4^{6k}} \sum_{i=0}^{k} \frac{1}{i!}(\frac{4k}{c-1})^i exp(\frac{8k}{c-2})$$

$$\leq \frac{1}{4^{6k}} e^{4k/(c-1)} e^{8k/(c-2)} \approx \frac{e^{12k/c}}{4^{6k}}$$

There are at most m^2 pairs of locations and at most n^3 triplets of probes. Thus, the probability is bounded by

$$m^2 n^3 \frac{e^{12k/c}}{4^{6k}} \leq \frac{\alpha^2 \beta k e^{12k/c}}{4^k} << 1$$

for c large enough.

Hence, the probability that throughout the algorithm, there are three fooling probes arising from the same location, is negligible. ■

6.4.2 Correctness of Algorithm B

Now we have the tools to prove the correctness of algorithm B in the various cases:

Theorem 50 *The probability that algorithm B fails is bounded by an arbitrary small constant, if the length of the sequence is $m = O(\frac{4^k}{k})$ and the number of probes is $n = \Theta(k4^k)$, assuming there are no hybridization errors.*

Proof: Since there are no errors, the correct base has always at least k supporting probes. Fix an incorrect base, and let Y be the number of supporting probes of this base. The algorithm fails if at some point $Y \geq k$. Therefore, we would like to show that for some $\epsilon < 1$,

$$Pr[Y \geq k] < \frac{\epsilon}{m}$$

For $k > 3$, if there are k fooling probes they can be partitioned into subsets of sizes 1 or 2 of probes which come from the same location (this is true since by Claim 49, the probability of 3 fooling probes that arise from the same location is negligible). By Claim 46, the probability of a single fooling probe is bounded by α and by Claim 48, the probability of two fooling probes that arise from the same location is bounded by α^2. The number of possibilities for partitions into subsets is the k^{th} Fibonacci number. Hence,

$$Pr[Y \geq k] \leq \alpha^k \cdot \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^k - \left(\frac{1-\sqrt{5}}{2} \right)^k \right) \leq \frac{\epsilon}{m}$$

for α small enough.

Hence, for any ϵ, one can get $m = O(\frac{4^k}{k})$ and $n = \Theta(k4^k)$ with the desired failure bound. ■

Note that by increasing k, one can get any derived failure probability even if $m = \alpha\frac{4^k}{k}$ and $n = k4^k$.

Theorem 51 *Suppose there are both false negatives and false positives, the length of the sequence is $m = O((1-q)\frac{4^k}{k})$ and the number of probes is $n = \Theta(\frac{k4^k}{1-q})$, where q is the rate of false negatives and the rate of false positives is $p \leq \frac{1-q}{430}$. Then, the probability that algorithm B fails is bounded by an arbitrary small constant.*

Proof: Let X be a random variable that counts the number of supporting probes for the correct base. Fix an incorrect base, and let Y be the number of supporting probes of this base. Clearly, $Y = Y_1 + Y_2$, where Y_1 is the number of supporting probes of the false extension that are fooling probes, and Y_2 is the number of supporting probes arising from false positives. The algorithm fails if at some point $Y \geq X$. Therefore, we would like to show that for some $\epsilon < 1$,

$$Pr[X \leq Y_1 + Y_2] < \frac{\epsilon}{m}$$

We first bound X. In this case, X is binomial, as in our model, the errors are independent. Thus, $X \sim B(\frac{5k}{1-q}, 1-q)$ and therefore, by Lemma 41,

$$Pr[X < k] = Pr[X - 5k < -4k] < e^{-1.6k} < \frac{\epsilon}{3m}$$

Y_1 is the same random variable as Y in the proof of Theorem 50 and therefore, $Pr[Y_1 \geq \frac{k}{2}] < \frac{\epsilon}{3m}$, for α small enough. Y_2 is clearly bounded by $\frac{5k}{1-q}$. As we need to bound the probability that Y_2 is large, we can assume that $Y_2 \sim B(\frac{5k}{1-q}, p)$. The expectation of Y_2 is $\frac{5kp}{1-q} \leq \frac{k}{86}$. By Lemma 40,

$$Pr[Y_2 > \frac{k}{2}] = Pr[Y_2 > 43 \cdot \frac{k}{86}] < \left((e/43)^{\frac{1}{2}} e^{-\frac{1}{86}}\right)^k < \frac{\epsilon}{3m}$$

Hence,

$$Pr[X \leq Y] \leq Pr[X < k] + Pr[Y_1 > k/2] + Pr[Y_2 > k/2] < \frac{\epsilon}{m}$$

■

Remark 52 The assumption that the false positive rate is very low has the following *biochemical* justification. By increasing the hybridization stringency, the number of false positives can be decreased at the expense of increasing the false negative rate, which has a small effect on the success probability (the length of the reconstructed sequence depends linearly on the rate of false negatives).

6.5 Experimental Results

We implemented algorithm B and tested it in simulations with noisy data. The implementation is fast: on a regular workstation, reconstruction requires less than a second even with $n = 4^{10}$ probes and targets of length 35,000.

We tested the algorithm on various types of sequences:

- Randomly generated sequences.
- DNA sequences, which originate from bases 20,000,000 - 40,000,000 of human chromosome 10 [National Center for Biotechnology Information, 2002a].
- DNA coding sequences, which were formed by a concatenation of the unigene file Hs.seq.uniq
  [National Center for Biotechnology Information, 2002b].

For some of the sequences we tested also the success of almost perfect reconstruction. A reconstruction is considered almost perfect if up to 0.5% of the bases of the sequence are incorrect.

Figure 6.3 summarizes the performance as a function of the noise parameters. Behavior as a function false positive rate is surprisingly good and extends far beyond the range of the theoretical analysis: In the absence of false negative errors, even if $p = 0.2$, we can reconstruct random sequences of length over $22,000$. Tolerance to false negatives is a bit lower. The simulations indicate a nearly linear correlation between the length of reconstructible sequence and p. As expected, the performance on randomly generated sequences is the best, and coding sequences behave better than arbitrary DNA sequences, since they are much less repetitive. An interesting feature of the algorithm is that it tends to make few and local errors (if any): if we tolerate a very small number of mismatches (almost perfect reconstruction), then, for example, the average length of reconstruction with 0.2 error rates goes up from 379 to 2380.

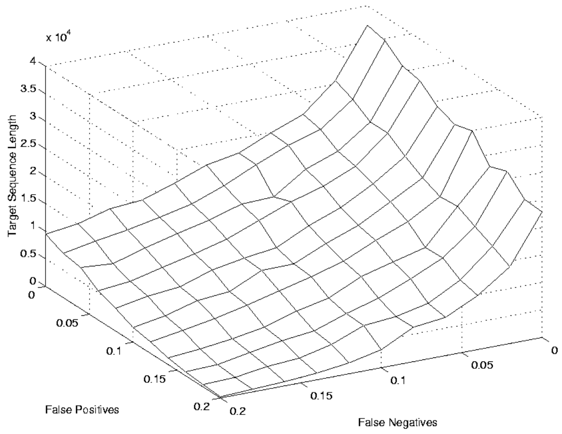
Figure 6.4 measures the effect of the parameter c that reflects the length of the probes (and the sparsity of specified positions in them), for various sequences. The increase in c improves the results for all types of sequences. We do not have a complete explanation to the jump observed for random sequences. Our theoretical analysis (section 6.4.2) requires that c is sufficiently large, and perhaps $c < 3$ suffices, at least for almost perfect reconstruction.

Figure 6.5 compares the performance of our algorithm to the only other experimental report on performance on error-prone data using universal probes. Doi and Imai [2000] develop an extension to the Preparata et al. scheme which handles noisy spectra. For the comparison, we used the same number of probes reported by Doi and Imai [2000] and plotted their and our experimental success rate as a function of the target length. For the error-less case the algorithm of Preparata et al. is superior: For example, using 4^{10} probes, that algorithm reconstructs sequences of length 70000 bases, while ours can only reconstruct sequences of length 37000. However, the results in Figure 6.5 show that our algorithm is much more robust to errors. For example, the length reconstructed with 97% success rate improves the results of Doi and Imai [2000] by a factor of 4. The Doi-Imai algorithm is superior for success rate bellow 50%, but obviously, this is not the range of interest.

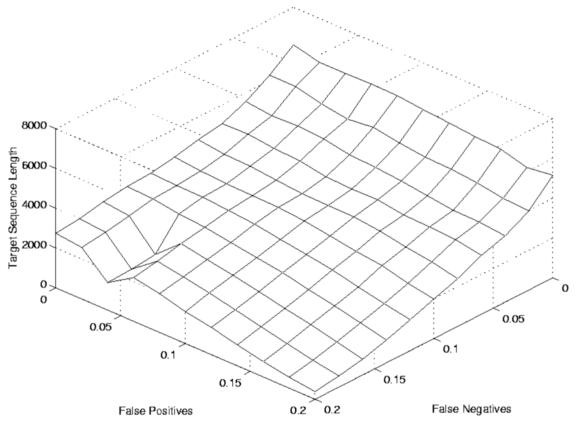
6.6 Discussion and Subsequent Work

In this chapter we introduced a new approach to design DNA arrays coping with false positive and negative errors. To the best of our knowledge this is the first time a theoretical analysis was done for such a model. We show that both theoretically, and in practice (using simulations), the sensitivity of our algorithm to errors is smaller, in comparison with previous algorithms.

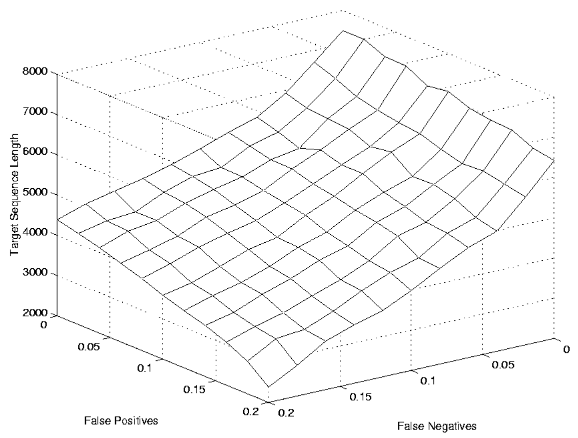
A natural extension of algorithm B is an algorithm similar to algorithm A, that is, instead of counting the number of supporting probes for only four possible extensions, we



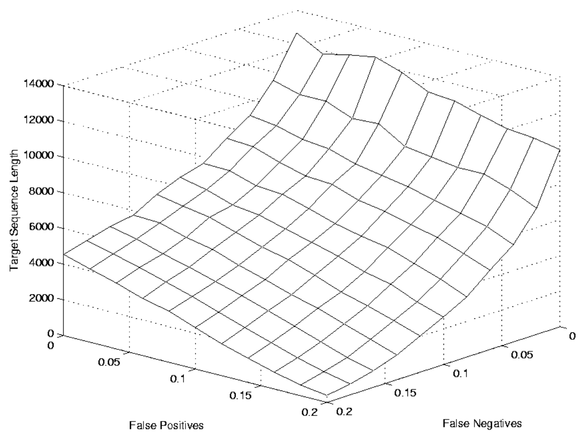
(a) Randomly generated sequences



(b) DNA sequences



(c) DNA sequences (almost perfect reconstruction)



(d) DNA coding sequences

Figure 6.3: The length of the reconstructed target sequences as a function of the error rates, for various sequences. The z axis measures L, the length of the longest targets correctly reconstructed in at least 90% of the cases. In all cases we used $k = 8$,$n = 4^{10}$ probes and $c = 10$. In (a), we used a sample of 200 random sequences to obtain statistics on each data point. In (b),(c) and (d), we used non-overlapping substrings of length L, covering together the DNA sequence source, to obtain statistics on each data point.

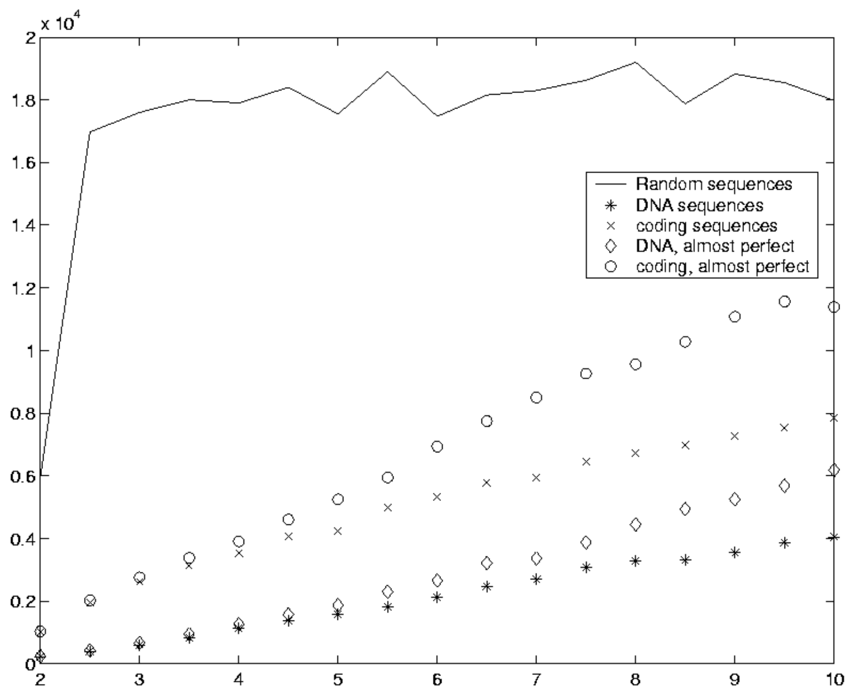


Figure 6.4: The length of the reconstructed target sequences as a function of the sparsity of specified symbols in the probes (the parameter c), for various sequences. The y axis measures L, the length of the longest targets correctly reconstructed in at least 90% of the cases. A sample of 200 sequences was used to obtain statistics on each data point. In all cases we used $k = 8$,$n = 4^{10}$ probes and $p = q = 0.05$.

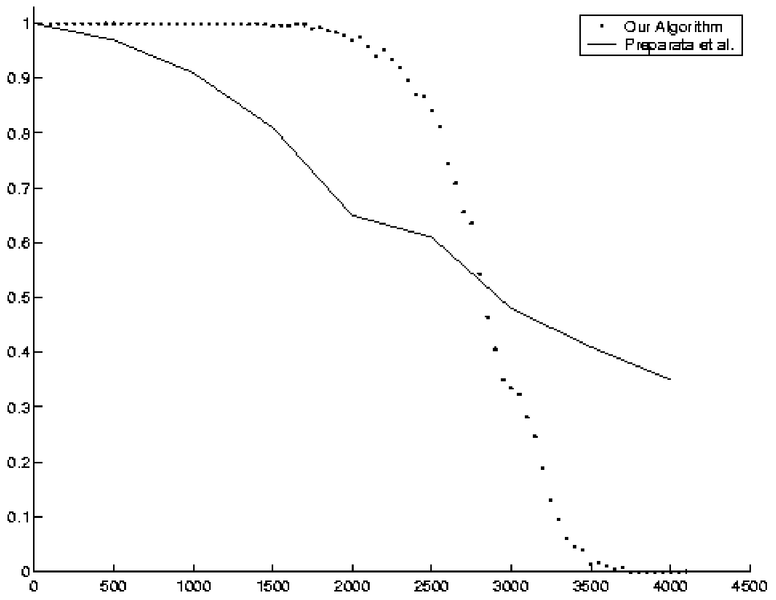


Figure 6.5: Reconstruction success rate as a function of target length by our algorithm (solid line) and the extension of Doi and Imai [2000] to the algorithm of Preparata et al. (dotted line). The success rate is the fraction of correctly reconstructed sequence targets. A sample of 200 random sequences was used to obtain statistics on each data point. In both simulations there were 4^8 probes, the error level was $p = q = 0.001$. The probe sparsity was $c = 4$.

could count the number of supporting probes of a set of possible paths that extend the current reconstructed sequence. It is likely that this algorithm will give better results in practice than algorithm B.

Although universal bases have been generated successfully in the laboratory [Loakes and Brown, 1990], it is unclear yet if long probes that contain many universal bases hybridize reliably. The fact that low sparsity suffices ($c = 4$ when $k = 6$) is encouraging.

Our ideas are extended by Tsur to deal with resequencing by hybridization [Tsur, 2003a] and SBH with rounds [Tsur, 2003b]. Baptista and Guimares [2003] implemented our algorithms and suggested an alternative approach.

Chapter 7

Multiplexing Applications of Universal DNA Arrays

In this chapter we study a design and optimization problem that occurs, for example, when single nucleotide polymorphisms (SNPs) are to be genotyped using a universal DNA tag array. The problem of optimizing the universal array to avoid disruptive cross-hybridization between universal components of the system was addressed in previous work [Ben-Dor et al., 2000]. Cross-hybridization can, however, also occur assay-specifically, due to unwanted complementarity involving assay-specific components.

Here we examine the problem of identifying the most economic experimental configuration of the assay-specific components that avoids cross-hybridization. Our formalization translates this problem into the problem of covering the vertices of one side of a bipartite graph by a minimum number of balanced subgraphs of maximum degree 1. We show that the general problem is NP-complete. However, in the real biological setting the vertices that need to be covered have degrees bounded by d. We exploit this restriction and develop an $O(d)$-approximation algorithm for the problem. We also give an $O(d)$-approximation for a variant of the problem in which the covering subgraphs are required to be vertex-disjoint.

The results in this chapter are published in [Ben-Dor et al., 2004] (preliminary version in [Ben-Dor et al., 2003]).

7.1 Introduction

SNPs (single nucleotide polymorphisms) are differences, across the population, in a single base, within an otherwise conserved genomic sequence [Wang et al., 1998]. The sequence variation represented by SNPs is often directly related to phenotypic traits. Such is the case when the variation occurs in coding or other functional (e.g., regulatory) regions [Cargill et al., 1999]. Somatic or native SNPs in oncogenes or in related regions can determine cancer susceptibility and are often related to pathogenesis (see, e.g., [Venitt, 1994, 1996, Kristensen et al., 2000, Watanabe et al., 2002]). *Genotyping* is a process that determines the variants present in a given sample, over a set of SNPs. SNPs also serve as genetic markers that can be used in linkage and association studies (see, e.g., [Risch, 2000]). In the latter case a population of samples is jointly measured and the frequencies of the different variants are inferred. Efficient SNP detection, genotyping and measurement techniques have, therefore, great clinical, scientific and commercial value.

Methods for high throughput SNP genotyping are under fast development and evolution. This task enlists various molecular biology techniques, separation technologies and detection methods. Methods based on mass spectrometry or length separation are described, e.g, in [Grant and Phillips, 2001, Schouten et al., 2002]. Other methods are based on hybridization array technology [Syvanen, 1999, Hacia, 1999, Drmanac et al., 1991]. In an array-based hybridization assay a target-specific set of oligonucleotides is synthesized or deposited on a solid support surface (e.g., silicon or glass). A fluorescently labeled target sample, a mixture of DNA or RNA fragments, is then brought in contact with the treated surface, and allowed to hybridize with the surface oligonucleotides. Scanning the resulting fluorescence pattern reveals information about the content of the sample mixture. Theoretically, the assay conditions are such that hybridization only occurs in sites on the surface that are Watson-Crick complements to some substring in the target. In practice, cross-hybridization is a main source of signal contamination in any array-based hybridization assay.

S. Brenner and others [Brenner, Morris et al., Gerry et al., 1999] suggest an alternative approach based on *universal arrays* containing oligonucleotides called *antitags*. The Watson-Crick complement of each antitag is called a *tag*. The tag–antitag pairs are designed so that each tag hybridizes strongly to its complementary antitag, but not to any other antitag. We shall call the entire system a *DNA Tag/AntiTag system* and in short a *DNA TAT system*. To exemplify the approach we describe in detail the application of universal arrays to SNP genotyping. The method is illustrated in Figure 7.1 and consists of the following steps:

1. A set of reporter molecules (one for each SNP) is synthesized. Each reporter molecule consists of two parts that are ligated together. The *primer* part is the Watson–Crick complement of the upstream sequence that immediately precedes the polymorphic site of the SNP. The other part is a unique tag – an element of the universal set of

tags.

2. When an individual is to be genotyped, a sample is prepared that contains the sequences flanking each of the SNP sites. Typically these are PCR amplicons. The sample is mixed with the reporter molecules and solution-phase hybridization takes place. Assuming that specificity is perfect, the flanking sequences of the SNPs only hybridize with the appropriate reporter molecule.

3. Single dideoxynucleotides, `ddA,ddC,ddT,ddG`, fluorescently labeled with four distinct chemical dyes, are added to the mixture. In a polymerase-driven reaction each hybridized reporter molecule is extended by exactly one labeled dideoxynucleotide.

4. The extended reporter molecules are separated from the sample fragments, and brought into contact with the universal array. Assuming that specificity is perfect, the tag part of each reporter molecule will only hybridize to its complementary antitag on the array.

5. For each site of the array, the fluorescent dyes present at that site are detected. The colors indicate which bases participated in the extension reaction, at the corresponding SNP site and, thus, reveal the SNP variations possessed by the tested individual.

This method, with appropriate modifications, is also applicable in a pooled genotyping strategy, where PCR is applied to pooled DNA from several individuals and the purpose is to determine allele frequencies. In addition, the general idea of a universal array is also applicable for other measurement purposes. For example, if the reporter molecules are designed to be specific, each for some target mRNA, then the same protocol can be used for expression profiling.

Designing DNA TAT systems presents a tradeoff. Clearly, it is desirable to have as many tags as possible, in order to maximize the number of SNPs that can be genotyped in parallel. On the other hand, if too many tags are used, similar tags will necessarily entail cross-hybridization events (where tags hybridize to foreign antitags), reducing accuracy. The design of DNA TAT systems is independent of any particular application scenario and is, thus, optimized to avoid cross-hybridization between tags and foreign antitags. This issue is addressed in [Ben-Dor et al., 2000].

In performing an actual genotyping assay there are also assay-specific sources of potential cross-hybridization. One major source involves the primer parts of the reporter molecules hybridizing to array bound antitags, producing a confusing signal unless the corresponding site on the array is designated to the primed SNP site. As this problem is

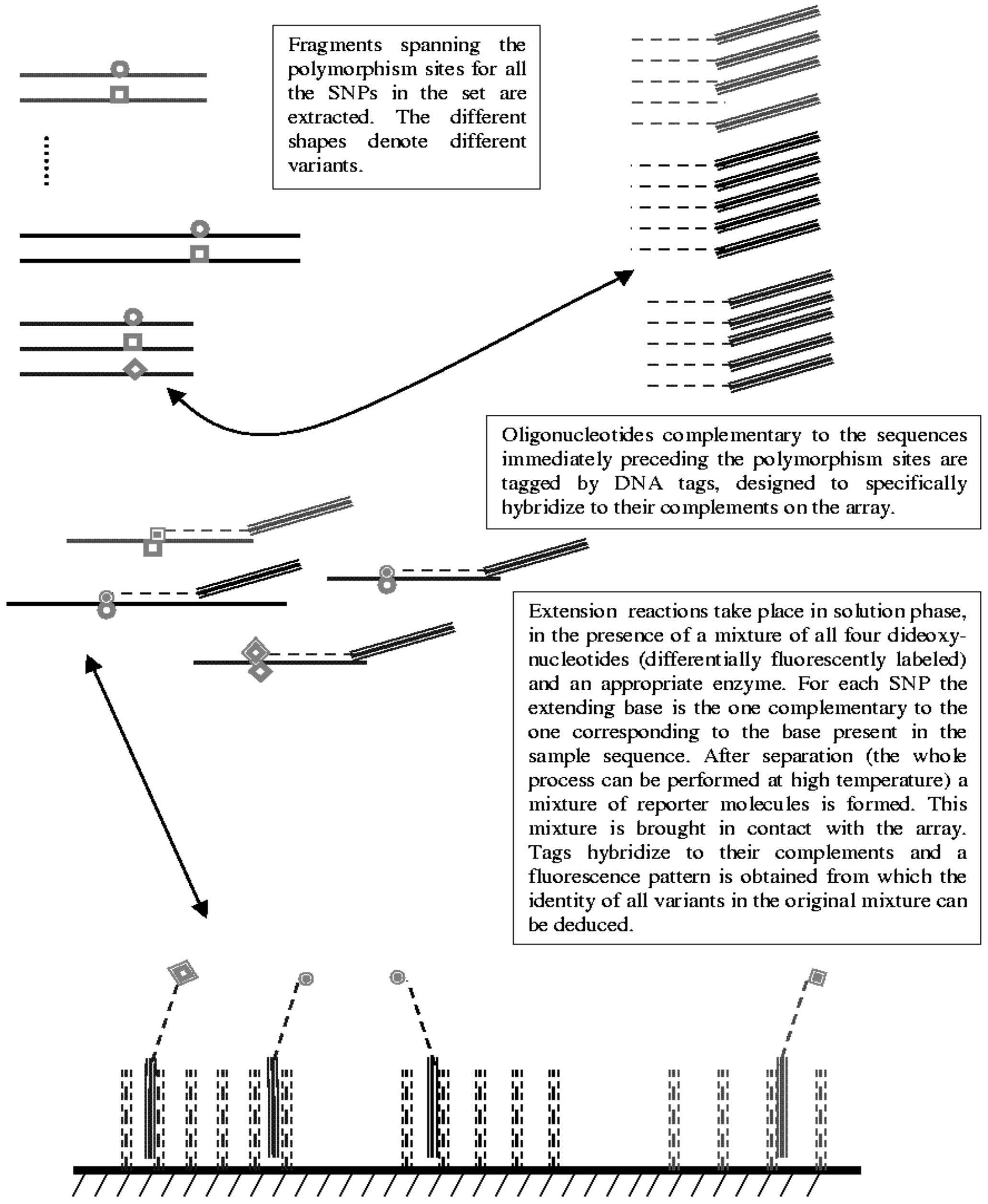

Fragments spanning the polymorphism sites for all the SNPs in the set are extracted. The different shapes denote different variants.
Oligonucleotides complementary to the sequences immediately preceding the polymorphism sites are tagged by DNA tags, designed to specifically hybridize to their complements on the array.
Extension reactions take place in solution phase, in the presence of a mixture of all four dideoxy-nucleotides (differentially fluorescently labeled) and an appropriate enzyme. For each SNP the extending base is the one complementary to the one corresponding to the base present in the sample sequence. After separation (the whole process can be performed at high temperature) a mixture of reporter molecules is formed. This mixture is brought in contact with the array. Tags hybridize to their complements and a fluorescence pattern is obtained from which the identity of all variants in the original mixture can be deduced.

Figure 7.1: A scheme for SNP genotyping using a DNA TAT system.

specific to the actual set of SNPs to be studied, it is impossible to address it in the TAT system design stage.

In this chapter we assume that the set of primers for the reaction was designed to achieve the desired level of specificity in the target genome, i.e., reporter molecules will not extend on unintended genomic sequences. Remaining assay-specific sources of potential confusing signal are as follows: Primer to antitag cross-hybridization as described above. Sandwich cross-hybridization: A duplex of two reporter molecules hybridizes to a single site in the array. The duplex is formed due to high complementarity of the sequences and hybridizes to the array site through one of the tags. Sandwich cross-hybridization involves a complicated configuration and is rare. Primer to primer mis-extension: The primer parts of reporter molecules can hybridize to other primers in the extension step in a configuration that allows for polymerase extension. Primer to tag mis-extension: Similar, but with primer parts of reporter molecules appropriately hybridizing to tags in the extension step. The latter two are similar to primer-dimer formations in PCR. Note that the cross-hybridization needs to be perfect at the 3' end of the reporter for these problems to occur.

Primer to antitag cross-hybridization is, therefore, by far the most probable source of confusing signal. This is the only problem we explicitly address in this chapter, although the methods we develop can be extended to handle primer to tag mis-extension. In addition, any multiplexing scheme for a universal array based assay can be screened for any undesired properties prior to performing the measurement. Avoiding less common configurations should be deferred to such a screening stage rather than taken into account in the design stage.

Maximizing the multiplexing rate for a given set of SNPs (alternatively, minimizing the number of arrays to be used for a single genotyping measurement of this set), under given primer to antitag cross-hybridization constraints, is the main subject of this chapter. Every time we say cross-hybridization we mean primer to antitag cross-hybridization. To control the multiplexing rate we use our freedom to choose how to partition the set of SNPs into *assignable* subsets (subsets that can be measured using one array without cross-hybridization) and to assign tags to SNP sites. The assignment of a primer to a tag means that they will form a reporter molecule. A proper assignment $(p_1, t_1), \ldots, (p_k, t_k)$ of primers to tags should avoid cross-hybridization between every primer p_i and antitag $\overline{t_j}$, unless $i = j$.

To approach the multiplexing problem we model the input data using a bipartite graph, in which the primers are on one side and the tags are on the other side. Each edge in the graph indicates potential cross-hybridization between a primer and the corresponding antitag. The multiplexing problem then translates to the problem of covering the primer vertices of the graph using a minimum number of balanced induced subgraphs of maximum degree one. We prove that the general problem is NP-complete. However, in actual applications the primer vertices have degrees bounded by some constant d. We exploit this restriction and develop an $O(d)$-approximation algorithm for the problem. Specifically, we give an algorithm that produces a cover of cardinality $(\lceil \frac{m}{n} d \rceil + 2)$ for m primers and n tags, whenever $m \leq n$. We derive an algorithm for the case $m > n$, which guarantees a cover of size $\lceil \frac{m}{n} \rceil (d+2)$. We also give an $O(d)$-approximation for a variant of the problem in which the covering subgraphs are required to be vertex-disjoint.

7.1.1 Organization of this Chapter

In Section 7.2 we mathematically model and formalize the optimization problem. In Sections 7.3, 7.4 and 7.5 we present hardness results and algorithmic approaches to the covering problem and its variants. Approximation results are given in Section 7.6.

7.2 Formal Problem Definition

Denote the set of DNA tag sequences associated with the universal array by T, and the corresponding set of antitags by $\overline{T}$. By P we denote the set of primers (the sequences complementary to the upstream regions of the SNPs). Let $m = |P|$ and $n = |T|$. A *reporter molecule* is a primer-tag pair (p, t), where $p \in P$ and $t \in T$. For a graph G and a subset of its vertices R, we denote by G_R the subgraph of G induced by R. We denote by $V(G)$ and $E(G)$ the sets of vertices and edges of G, respectively. For a vertex v we denote by $N(v)$ its set of neighbors in G.

Potential cross-hybridization between primers and antitags can be determined experimentally or predicted computationally, e.g., on the basis of the sequence [Ben-Dor et al., 2000]. The methods presented here are not specific to any such determination mechanism. We only assume that potential cross-hybridizations are given in the form of a binary $m \times n$ matrix A, such that:

$$\mathcal{A}pt = \begin{cases} 1 & \text{if } p \in P \text{ potentially hybridizes with } \bar{t} \in \overline{T}, \\ 0 & \text{otherwise.} \end{cases}$$

Solutions to our multiplexing problem correspond to proper partitions of P into subsets, where each subset corresponds to one array experiment. A set of SNPs can be measured in a single array operation, without cross-hybridization, if all corresponding members can be assigned to tags such that cross-hybridization is avoided. Formally:

Definition 8 *Let $R = \{(p_1, t_1), (p_2, t_2), \ldots, (p_k, t_k)\}$ be a set of reporter molecules with distinct $p_i \in P$ and distinct $t_j \in T$. R is said to be* non-cross-hybridizing *if $\mathcal{A}p_it_j = 0$ for all $i \neq j$.*

Definition 9 *A set of k distinct primers $P' = \{p_1, p_2, \ldots, p_k\} \subseteq P$ is called* assignable *if there exists a non-cross-hybridizing set of reporter molecules $\{(p_1, t_1), (p_2, t_2), \ldots, (p_k, t_k)\}$ for k distinct tags $t_1, \ldots, t_k \in T$. An assignable set of tags is similarly defined.*

We now give a characterization of assignable primer sets in terms of the cross-hybridization matrix A.

Definition 10 *A* subpermutation matrix *is a 0–1-matrix whose rows and columns can be permuted such that all entries outside the main diagonal are 0.*

Lemma 53 *A set of primers $P' \subseteq P$ is assignable if and only if P' corresponds to the row set of a square submatrix of A that is a subpermutation matrix.*

Proof: Let $P' = \{p_1, \ldots, p_k\}$. If P' is assignable then there exists a set of non-cross-hybridizing reporter molecules $\{(p_1, t_1), (p_2, t_2), \ldots, (p_k, t_k)\}$ with $\mathcal{A}p_it_j = 0$ for all $i \neq j$. Hence, the square submatrix of A induced by the rows in P' and the columns in $T' = \{t_1, t_2, \ldots, t_k\}$ is a subpermutation matrix.

Conversely, suppose there exists a square subpermutation submatrix of A with row set P'. Denote by $R = \{(p_1, t_1), (p_2, t_2), \ldots, (p_k, t_k)\}$ those elements of the submatrix that end up on the diagonal if its rows and columns are permuted such that all other entries are 0. Clearly, R corresponds to a non-cross-hybridizing set of reporter molecules, and its row set P' is assignable. ■

An alternative point of view models the input matrix A as a bipartite graph $G = (P, T, A)$, whose vertices are primers (P) and tags (T), and whose edges represent potential cross-hybridizations between primers and the corresponding antitags. Throughout the chapter we use graph and matrix language interchangeably. For convenience, we use A to denote both the input cross-hybridization matrix and the edge set of G, which is the set of primer-tag pairs $\{(p, t) : p \in P, t \in T, A_{p,t} = 1\}$. A subgraph $H = (P', T', E')$ of G is called *balanced* if $|P'| = |T'|$. H is called an *assignable subgraph* if H is a balanced induced subgraph of maximum degree 1.

Observation 54 *A matrix A with a set of rows P and a set of columns T is a subpermutation matrix if and only if the bipartite graph $G = (P, T, A)$ is an assignable graph.*

The following proposition formalizes a necessary and sufficient condition for a set of primers to be assignable:

Proposition 55 *Let $G = (P, T, A)$ be a bipartite graph, with $T = \{t_1, \dots, t_n\}$ and $P = \{p_1, \dots, p_m\}$. For $j = 1, \dots, n$, let $Y(j) = 1$ if t_j has degree zero, and 0 otherwise. For $i = 1, \dots, m$, let $X(i) = 1$ if G contains a tag of degree 1, which is adjacent to p_i, and 0 otherwise. Then P is assignable if and only if*

$$\sum_{j=1}^{n} Y(j) + \sum_{i=1}^{m} X(i) \geq m.$$

Proof: According to Observation 54 P is assignable if and only if G has an assignable subgraph H that covers the m vertices of P. On the one hand, if such a subgraph exists, each of its m tags contributes a value of 1 to either $\sum_{j=1}^{n} Y(j)$ or $\sum_{i=1}^{m} X(i)$. On the other hand, if one attributes one tag to each 1 generated in the above sum, the subgraph induced by m of these tags and P is an assignable graph. $\blacksquare$

Observation 54 leads to a short statement of our main optimization problem:

Definition 11 *A partition $\mathcal{E}$ of the primer set P is called a* primer cover *if each $P' \in \mathcal{E}$ is assignable.*

Problem 1 (Minimum Primer Cover (MPC)) *Given a bipartite graph $G = (P, T, A)$, find a minimum primer cover of P.*

Equivalently, MPC calls for finding a minimum set of assignable subgraphs that cover P, or a minimum set of square subpermutations submatrices of A that cover all rows of A.

7.3 Minimum Primer Cover

In this section we show that MPC is NP-complete and give an algorithm with guaranteed worst case performance for the problem when the degrees of the primer vertices in the input graph are bounded.

Theorem 56 *MPC is NP-complete.*

Proof: By reduction from SET COVER, where all input subsets are required to have cardinality at least 2. This problem is known to be NP-complete [Garey and Johnson, 1979, Problem SP5]. Given an instance $(P, \mathcal{S}, l)$ of SET COVER, where $\mathcal{S}$ is a collection of subsets of a finite set P, and l is an integer, we construct an instance $(G = (P, T, A), l)$ of MPC as follows: For every subset $S_i = \{s_{i,1}, \ldots, s_{i,k}\} \in \mathcal{S}$, we add vertices $T_i = \{t_{i,1}, \ldots, t_{i,k}\}$ to T, such that every $t_{i,j}$ $(1 \leq j \leq k)$ is adjacent to all the vertices in $P \setminus S_i$ (and to no other vertex).

A set-cover $S_{i_1}, \ldots, S_{i_l}$ induces a primer cover $\mathcal{E} = (\mathcal{S}_{\rangle_\infty}, \ldots, \mathcal{S}_{\rangle_\updownarrow})$ with the same cardinality, since each S_{i_j} is assignable. Conversely, suppose there exists a primer cover $\mathcal{E}$ of size l. A set $S \in \mathcal{E}$ is called *homogeneous* if all its primers belong to the same subset S_i. S is called *crossing* if $S = \{p, p'\}$ and no $S' \in \mathcal{S}$ contains both p and p'.

Observe that every assignable primer set is either homogeneous or crossing. If all the primer sets in the cover are homogeneous, taking the corresponding subsets yields a set cover of size l. Otherwise, we can apply a series of modifications to the cover that eliminate the crossing sets in the cover and preserve its cardinality. In each step we consider a crossing primer set $S = \{s_i, s_j\} \in \mathcal{E}$, where $s_i \in S_i, s_j \in S_j$ and $i \neq j$. If some $s'_i \in S_i$ is covered by a homogeneous set S', we can move s_i from S to S', eliminating one crossing set. Otherwise, there exists a crossing set $S'' = \{s'_i, s_k\}$, which contains some $s'_i \in S_i$. By moving s'_i to S and s_j to S'', we eliminate one crossing set. Applying these modifications to the cover we necessarily end with a homogeneous cover of size l. ■

Note that the proof of Theorem 56 implies that MPC is NP-complete even if the number of tags is required to be greater or equal to the number of primers.

In the context of DNA TAT systems, as constructed in [Ben-Dor et al., 2000], the choice of tags implies that the degree of every $p \in P$ is bounded by some constant d. This is true since, by construction, strings that are long enough to potentiate cross hybridization are not common to any two tags. Each primer (of bounded length) can have at most d such substrings (where d depends on the primer length and the thermodynamical model) and can, therefore, form edges with at most d tags. Practical values of d range between 5–15. We call an instance $G = (P, T, A)$ of MPC a *d-bounded instance* if the degree of every $p \in P$ is bounded by d. We shall exploit this restriction and develop an $O(d)$ approximation algorithm for MPC on d-bounded instances.

By Proposition 55, instances that can be covered by one subgraph are easily determined. Henceforth we assume that the input MPC instance has an optimum solution of cardinality at least 2. The case $d = 1$ is polynomial for $m \leq n$:

Lemma 57 *Let $G = (P, T, A)$ be a 1-bounded instance of MPC, with $m \leq n$. Then a minimum primer cover can be found in polynomial time.*

Proof: Assume $m = n$ (if $n > m$, remove arbitrarily $n - m$ vertices from T). Let G_1 be the subgraph induced by the vertices of a maximal matching M in G. Let G_2 be the subgraph induced by all other vertices. Clearly, G_1 and G_2 are balanced and together they span the entire set of primers. Since the degree of every primer is at most 1, G_1 has maximum degree 1. By maximality of M, G_2 contains no edges. The claim follows. ■

Our main result in this section is a polynomial algorithm which guarantees finding a solution of cardinality at most $(\lceil \frac{m}{n} d \rceil + 2)$ for a d-bounded input instance with $m \leq n$.

Theorem 58 *Let $G = (P, T, A)$ be a d-bounded instance of MPC with $m \leq n$. Assume that $n > d(d+1)$. Then we can find, in $O(\frac{m^2}{n} d(d + \log m))$ time, a solution to MPC on G of cardinality at most $\lceil \frac{m}{n} d \rceil + 2$.*

Proof: **Algorithm:** Let $x = \lceil \frac{m}{n} d \rceil$. First, assume for simplicity that $\frac{m}{x+1}$ is an integer. Consider the following iterative algorithm: The algorithm has $x + 1$ steps. At each step it identifies an assignable subgraph of size $\frac{m}{x+1}$, deletes the primer vertices of this subgraph (as they are covered), and continues the next iteration on the remaining graph. The assignable subgraph at each iteration is constructed on the basis of the $\frac{m}{x+1}$ tags of lowest degree. Altogether we obtain a cover with $x + 1$ assignable subgraphs. We now prove the correctness of the algorithm.

In the i-th iteration ($0 \leq i \leq x$), the current graph G_i consists of $m\left(\frac{x+1-i}{x+1}\right)$ primers and n tags. Since the degree of each primer is bounded by d, the number of edges in G_i is at most $md\left(\frac{x+1-i}{x+1}\right)$. Let T' be the set of $\frac{m}{x+1}$ tags with lowest degrees in G_i.

Claim 59 *The degree of each tag in T' is bounded by $x - i$.*

Proof: Suppose there exists a tag in T' with degree at least $x - i + 1 \geq 1$. Hence, all tags in $T \setminus T'$ have degree at least $x - i + 1$, so the number of edges incident on $T \setminus T'$ is at least $r = \left(n - \frac{m}{x+1}\right)(x - i + 1) = (n(x+1) - m)\left(\frac{x+1-i}{x+1}\right)$. Since $n \geq m$ and $nx \geq md$, we have $r \geq |E(G_i)|$, a contradiction. ■

Claim 59 implies that the total number of edges adjacent to tags in T' is bounded by $m\left(\frac{x-i}{x+1}\right)$. Therefore, there are at least $m\left(\frac{x+1-i}{x+1}\right) - m\left(\frac{x-i}{x+1}\right) = \frac{m}{x+1}$ primers with degree 0 in G_i. Taking these $\frac{m}{x+1}$ primers along with the tags in T' yields an assignable subgraph. This completes the correctness proof.

If $\frac{m}{x+1}$ is not an integer, we can use the above algorithm to find $x + 1$ assignable subgraphs of size $\lfloor \frac{m}{x+1} \rfloor$ each. The number of primers left to cover is $m - (x+1)\lfloor \frac{m}{x+1} \rfloor \leq x \leq d$. Those primers are adjacent to at most d^2 tags, so there are at least $x + 1$ other tags that together with the uncovered primers form an assignable subgraph.

Complexity: The algorithm performs $x+1$ iterations. For every i, iteration i requires sorting the tags by their degrees, which costs $O(m \log m)$ time, and finding $\frac{m}{x+1}$ primers of degree 0 in G_i, which costs $O\left(md\left(\frac{x+1-i}{x+1}\right)\right)$ time. Summing over all iterations we get the stated bound. ■

Note that our algorithm actually gives a solution in which every subgraph is an independent set or, equivalently, the submatrices covering A are all 0.

Corollary 60 *Let $G = (P, T, A)$ be a bipartite graph in which the degree of each $p \in P$ is bounded by d, and $m > n > d(d+1)$. Then we can find, in polynomial time, a solution to MPC on G of cardinality at most $\lceil \frac{m}{n} \rceil (d+2)$.*

Proof: This follows by arbitrarily partitioning P into sets of size at most n and applying Theorem 58. ■

7.4 Maximum Assignable Primer Set

In this section we study a greedy approach to MPC that mimics approximation algorithms for SET COVER (cf. [Cormen et al., 1990]). The scheme is recursive: The largest assignable subset in P is identified and removed, and the algorithm proceeds recursively on the remaining graph. If possible, this approach could guarantee an $O(\log m)$ approximation, and would typically perform better. However, each of the stages is NP-complete:

Problem 2 (Maximum Assignable Primer-set (MAP)) *Given a bipartite graph G, find a maximum assignable subgraph H of G.*

Theorem 61 *MAP is NP-complete.*

Proof: By reduction from the complete balanced bipartite subgraph problem, where the input is a bipartite graph and an integer k, and the objective is to find a complete balanced subgraph with k vertices on each side. This problem is known to be NP-complete [Garey and Johnson, 1979, Problem GT24], and can be trivially reduced to the empty balanced bipartite subgraph problem, where the input is a bipartite graph and an integer k, and the objective is to find a balanced subgraph with k vertices on each side and no induced edges. We reduce the latter problem to MAP.

Given an instance $(G = (U, V, E), k)$ of the empty balanced bipartite subgraph problem, where $|U|, |V| < l$, we build an instance $(G' = (U', V', E'), lk)$ of MAP. Each vertex

v in G is duplicated l times $v^1, \ldots, v^l$ in G'. For every edge $(u, v) \in E$ we add the edges (u^i, v^j) to E' for all $1 \leq i, j \leq l$.

Clearly, an empty balanced induced subgraph of size k induces a solution to MAP of size at least lk. Conversely, suppose that $H = (X, Y, F)$ is an assignable subgraph of G', and $|X| \geq lk$. We first claim that $|F| < l$. If $|F| \geq l$, then F contains, w.l.o.g., two edges $(u^1, a), (u^2, b)$ for some $u \in U$. But then either $a = b$, implying that a has degree at least 2 in H, or both u^1 and u^2 have degree at least 2 in H, a contradiction.

Removing all vertices incident to edges in F we obtain a solution to MAP with size (strictly) greater than $(k-1)l$, since F is a matching. This implies an empty balanced induced subgraph of size k in G. ■

Note that the related problem of finding a maximum induced matching in a bipartite graph is also NP-complete [Cameron, 1989].

7.4.1 Algorithmic approaches

In this subsection we give a polynomial algorithm for MAP in the case that every primer has at most one adjacent tag, and an integer programming formulation for the general case.

Lemma 62 *Let $G = (P, T, A)$ be a bipartite graph in which the degree of each $p \in P$ is at most 1. Then a maximum assignable subgraph of G can be found in polynomial time.*

Proof: Let $T^* = \{t : |N(t)| > 1\}$ and let $n^* = |T^*|$.

Observation 63 *Let $H = (P', T', E')$ be a solution to MAP. For every $t \in T^*$, either $t \notin T'$ or $|N(t) \cap P'| = 1$. Moreover, for $p' = P' \cap N(t)$ and for every other $p \in N(t)$, $(H \setminus \{p'\}) \cup \{p\}$ is also a solution.*

The observation motivates the following algorithmic scheme for creating a solution: For every $t \in T^*$, either (1) remove t from G, or (2) arbitrarily choose $p \in N(t)$ and remove $N(t) \setminus \{p\}$ from G. It remains to show how choose for each $t \in T^*$ between (1) and (2). We can enumerate the number of tags from T^* in an optimum solution. Denote this number by k. By choosing for removal the $n^* - k$ lowest degree tags in T^*, we end with a graph of maximum degree 1, which contains the required number of tags and a maximum number of primers. ■

We now give an integer programming formulation for MAP. In this formulation there are binary variables $e_{p,t}$ for every primer-tag pair. These variables are constrained so that

$e_{p,t} = 1$ if and only if in a maximum assignable set of primers $p \in P$ is matched with $t \in T$.

$$\begin{aligned}
\max \quad & \sum_{p,t} e_{p,t} && (7.1)\\
s.t. \quad & e_{p,t} \in \{0,1\} && (7.2)\\
& e_{p_1,t} + e_{p_2,t} \leq 1 \quad p_1 \neq p_2 && (7.3)\\
& e_{p,t_1} + e_{p,t_2} \leq 1 \quad t_1 \neq t_2 && (7.4)\\
& e_{\pi,t} + e_{p,\tau} \leq 1 \quad A_{p,t} = 1, \pi \neq p, \tau \neq t && (7.5)
\end{aligned}$$

Lemma 64 *The above integer programming solves MAP.*

Proof: Suppose that $\{e_{p,t}\}$ is a vector satisfying the constraints. We shall prove that it is an appropriate primer-tag assignment. Constraints 3,4 ensure that there will be a 1-1 correspondence between primers and tags. Let this correspondence be $\{(p_1,t_1),\ldots,(p_k,t_k)\}$ (i.e., $e_{p_i,t_i} = 1$), then it suffices to prove that $A_{p_i,t_j} = 0$ for $i \neq j$. Suppose to the contrary that $A_{p_i,t_j} = 1$ for $i \neq j$. Then constraint 5 ensures that $e_{p_i,t_i} + e_{p_j,t_j} \leq 1$, a contradiction.

Conversely, suppose that $\{(p_1,t_1),\ldots,(p_k,t_k)\}$ is a non-cross-hybridizing set. Define $e_{p_i,t_j} = 1$ if and only if $i = j \leq k$. It is easy to check that this assignment satisfies the constraints of the programming. ■

7.5 Minimum Partition into Disjoint Assignable Subgraphs

In our discussion so far we did not require the covering subsets in a solution of MPC to be tag disjoint. From the assay point of view there is no need for such requirement. In this section we study a mathematically related question of optimally partitioning a bipartite graph into a set of vertex-disjoint assignable subgraphs that cover the set of primers. Note that it is meaningful only when the number of primers is at most the number of tags. We henceforth assume this is the case. We give an algorithm which produces a cover of size at most $2d$ for a graph with d-degree bounded primer vertices. The problem is formally stated as follows:

Problem 3 (Minimum Partition into Disjoint Assignable Subgraphs (MPDAS)) *Given a bipartite graph $G = (P,T,A)$, find a minimum set of* vertex-disjoint *assignable subgraphs that cover P.*

MPDAS is NP-complete by essentially the same reduction as in the proof of Theorem 56. Our covering algorithm is based on graph coloring and is given below.

Theorem 65 *Let $G = (P, T, A)$ be a d-bounded instance of MPDAS with $m \leq n$. Then we can find, in time $O(n + m^2 d)$, a solution to MPDAS on G of cardinality at most $2d$.*

Proof: Assume $n = m$ (if $n > m$, remove arbitrarily $n - m$ vertices from T). We shall find a collection of at most $2d$ assignable subgraphs that span the vertices of P. Let M be a maximal matching in G. Let H be the subgraph induced by the set of vertices that are not incident to edges of M. Clearly, H contains no induced edges and is assignable.

We now construct a directed graph $G' = (V', E')$ as follows: Every vertex $v \in V'$ corresponds to a pair of vertices $p \in P, t \in T$ that were matched by M. An edge $e \in E'$ is directed from $v_1 = (p_1, t_1)$ to $v_2 = (p_2, t_2)$ if and only if $(p_1, t_2) \in A$. By construction every vertex in G' has out-degree at most $d - 1$. Hence, G' can be colored using at most $2d - 1$ colors using SLO (smallest-last ordering) coloring [Matula and Beck, 1983]. Each coloring class corresponds to the vertices of an assignable subgraph, and together with H these subgraphs cover P.

The running time is dominated by finding a maximal matching in G, which costs $O(m^2 d)$ time (cf. [Cormen et al., 1990]). ■

In fact, we can produce smaller covers if the number of tags is strictly greater than the number of primers as the following theorem shows.

Theorem 66 *Let $G = (P, T, A)$ be a d-bounded instance of MPDAS with $n \geq (k + 1)m$, for some $k \geq 1$. Then we can find, in polynomial time, a solution to MPDAS on G with cardinality at most $2\lfloor \frac{d}{k} \rfloor$.*

Proof: We first remove from G all $n - m \geq mk$ tags with highest degrees. Clearly, the degree of each remaining tag is bounded by $\lfloor \frac{d}{k} \rfloor$. By changing the roles of tags and primers in the proof of Theorem 65, we obtain a solution of cardinality $2\lfloor \frac{d}{k} \rfloor$. ■

We end this section by commenting on the applicability of MPDAS: There is a protocol solution to avoiding primer to antitag cross-hybridization. The idea is to introduce blocking oligonucleotides, perfect Watson-Crick complements of the primers used in the assay, right after the extension reaction and prior to the array hybridization step. As these occupy the primer parts of the reporter molecule they block any potential hybridization of these primers. The main source of confusing signal now becomes primer to tag mis-extensions. By solving MPDAS for multiplexing the solution-phase experiments, it is possible to perform the genotyping using a single array, at the cost of performing slightly more solution-phase experiments (since, typically, a solution for MPC would have smaller cardinality than a solution for MPDAS on the same instance). This protocol has not been experimentally tested, to our knowledge. The principal motivation for MPDAS, therefore, remains purely mathematical.

7.6 Approximating MPC and MPDAS

In this section we study the approximation ratios of our algorithms for MPC and MPDAS. The first set of results uses Proposition 55, which implies that for both problems we can decide, in polynomial time, whether a solution of cardinality 1 exists.

Claim 67 *There exists a $(\lceil \frac{m}{n} d \rceil /2 + 1)$-approximation algorithm for MPC on d-bounded instances with $m \leq n$ and $n > d(d+1)$.*

Proof: Follows directly from Theorem 58 and Proposition 55. ■

Claim 68 *There exists a $(d+2)$-approximation algorithm for MPC on d-bounded instances with $m > n > d(d+1)$.*

Proof: Follows from Corollary 60, and the fact that at least $\lceil \frac{m}{n} \rceil$ subgraphs are needed in order to cover the primer set of an instance with $m > n$. ■

Claim 69 *There exists a d-approximation algorithm for MPDAS on d-bounded instances.*

Proof: Follows directly from Theorem 65 and Proposition 55. ■

Claim 70 *There exists a $\lfloor \frac{d}{k} \rfloor$-approximation algorithm for MPDAS on d-bounded instances with $n \geq (k+1)m$.*

Proof: Follows directly from Theorem 66 and Proposition 55. ■

Next, we show how to improve the approximation ratios of our algorithms for MPC and MPDAS in the case that $d > c \log^2 n$ for an appropriate constant c. This result is based on an improved algorithm for the case that the optimum solution has cardinality 2. This algorithm uses an approximation algorithm for the minimum bisection problem by Feige and Krauthgamer [2000]. A *minimum bisection* of a graph G with $2l$ vertices is a partition of $V(G)$ into two sets of size l, such that the number r of edges that cross the partition is minimum. The algorithm of Feige and Krauthgamer [2000] produces a bisection of size at most $kr \log^2 l$ for an appropriate constant k.

Theorem 71 *Let $G = (P, T, A)$ be a d-bounded instance of MPDAS with an optimum solution of cardinality 2. Then we can find, in polynomial time, a solution to MPDAS on G with cardinality $O(\log m\sqrt{d})$.*

Proof: We assume that $n = m$ (otherwise remove $n - m$ tags arbitrarily). Since P can be covered using two vertex-disjoint assignable subgraphs, these subgraphs induce a balanced cut (bisection) in G of size at most m. This bisection is formed by taking one side of the cut to be the union of the set of primers of one subgraph and the set of tags of the other subgraph. Our algorithm for MPDAS in this case starts by applying to G the minimum bisection approximation algorithm of Feige and Krauthgamer [2000], producing a bisection of size at most $km\log^2 m$. This bisection induces a partition into two balanced subgraphs (by joining the primers on one side of the cut with the tags on the other side). Clearly, in each subgraph there are at most $km\log^2 m$ edges. Next, these two subgraphs are handled using the following lemma.

Lemma 72 *Let $H = (P', T', E')$ be a d-bounded instance of MPDAS with $|P'| = |T'| = r$ and $|E| \le kr\log^2 r$. Then we can find, in polynomial time, a solution to MPDAS on H with cardinality at most $4\log r\sqrt{kd}$.*

Proof: Let $c = \log r\sqrt{kd}$. We partition the set of primers into two subsets, and cover each one separately:
(1) $S_1 = \{p \in P : deg(p) > c\}$. Clearly, $|S_1| \le \frac{kr\log^2 r}{c}$ and, thus, by Theorem 66 S_1 can be covered by $\frac{2kd\log^2 r}{c} = 2\log r\sqrt{kd}$ subgraphs.
(2) $S_2 = \{p \in P : deg(p) \le c\}$. By Theorem 65, S_2 can be covered by $2c = 2\log r\sqrt{kd}$ subgraphs. ■

Using Lemma 72 we can cover each of the two subgraphs produced by the bisection approximation algorithm using $4\log m\sqrt{kd}$ vertex-disjoint assignable subgraphs. Overall, we obtain a solution to MPDAS on G with cardinality at most $8\log m\sqrt{kd}$. ■

Corollary 73 *For $d > 36k\log^2 m$ we can approximate MPDAS on d-bounded instances in polynomial time to within a factor of $\frac{2d}{3}$ of optimum.*

Corollary 74 *For $d > 36k\log^2 m$ we can approximate MPC on d-bounded instances with $m \le n$ and $n > d(d+1)$, in polynomial time, to within a factor of $\frac{2d}{3}$ of optimum.*

Proof: Suppose the input instance has an optimum solution of cardinality 2. Observe that every tag which participates in both subgraphs in the optimum solution must have degree at most 2. Let T' denote the set of those tags. As a first stage we handle the subgraph induced by the tags in T' and up to $|T'|$ of their adjacent primers (arbitrarily selected). Due to the degree bound, this subgraph can be covered using at most 4 assignable subgraphs. Now, the remaining subgraph H can be covered by two vertex-disjoint assignable subgraphs. For convenience we shall assume that $H = (P, T, E)$ and $|P| = |T| = r \leq n$. ■

7.7 Discussion and Open Problems

In this chapter we formulated three combinatorial problems arising in multiplexing universal array experiments. MPC, MAP and MPDAS were shown to be NP-complete. Approximation algorithms were given for MPC and MPDAS in the case that the primer degrees in the input graph are bounded.

This chapter is based on some of the results that appeared in [Ben-Dor et al., 2004] (preliminary version in [Ben-Dor et al., 2003]). The paper contains also a stochastic model for the input data that is used to prove a lower bound on the cover size. In addition, the theoretical analysis is complemented by an implementation of two heuristic approaches and tests of their performance on simulated and real SNP data. Since I did not take part in these results, there are not included in this thesis.

Several theoretical problems remain open: The complexity of the MPC, MAP and MPDAS, when the degree of each primer is bounded, is still unknown. The approximation ratios achieved for MPC and MPDAS are not known to be tight. A variant of MPC arises if we wish to solve simultaneously also the primer to primer mis-extension problem. The corresponding graph is no longer bipartite; it contains edges between primers that may potentially cross-hybridize. The goal here is to find a minimum partition into assignable subgraphs, such that the set of primers in each subgraph induces no edges.

Bibliography

B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J.D. Watson. *Molecular Biology of the Cell*. Garlend Publishing Inc., New York and London, 1994.

N. Alon and J. H. Spencer. *The Probabilistic Method*. John Wiley and Sons, Inc., 2000.

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, 1996.

D.A. Bader, B. M.E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8(5):483–491, 2001.

V. Bafna and P. A. Pevzner. Genome rearragements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.

V. Bafna and P. A. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, May 1998. ISSN 0895-4801 (print), 1095-7146 (electronic).

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

E. S. Baptista and K. S. Guimares. Uma abordagem alternativa para seqenciamento por hibridizao. In *Proceedings of Seminrio Integrado de Software e Hardware (SEMISH'03)*, 2003.

A. Ben-Dor, T. Hartman, R. M. Karp, B. Schwikowski, R. Sharan, and Z. Yakhini. Towards optimally multiplexed applications of universal arrays. *Journal of Computational Biology*, 11(2-3):477–493, 2004.

A. Ben-Dor, T. Hartman, B. Schwikowski, R. Sharan, and Z. Yakhini. Towards optimally multiplexed applications of universal DNA tag systems. In *Proceedings of the Seventh Annual International Conference on Computational Molecular Biology (RECOMB 2003)*, pages 48–56, 2003.

A. Ben-Dor, R. M. Karp, B. Schwikowski, and Z. Yakhini. Universal DNA tag systems: A combinatorial design scheme. *Journal of Computational Biology*, 7(3):503–519, 2000.

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, 2001.

A. Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. In *Proc. 12th Annual Symposium on Combinaotrial Pattern Matching (CPM '01)*, 2001.

A. Bergeron, C. Chauve, T. Hartman, and K. Saint-Onge. On the properties of sequences of reversals that sort a signed permutation. In *Proceedings of JOBIM2002*, pages 99–108, St. Malo, France, June 2002.

A. Bergeron, J. Mixtacki, and J. Stoye. Reversal distance without hurdles and fortresses. In *Proc. 15th Annual Symposium on Combinaotrial Pattern Matching (CPM '04)*, pages 388–399. Springer, 2004.

P. Berman, S. Hannanhalli, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Proc. of 10th Eurpean Symposium on Algorith,s (ESA'02)*, pages 200–210. Springer, 2002. LNCS 2461.

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.

P. Berman and M. Karpinski. On some tighter inapproximability results. In *Proceedings of the 26th ICALP*. Springer, 1999.

M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172:11–17, 1996.

J. Blazewicz, P. Formanowicz, F. Glover, M. Kasprzak, and J. Weglarz. An improved tabu search algorithm for DNA sequencing with errors. In *Proceedings of the III Metaheuristics International Conference MIC'99, Angra dos Reis*, pages 69–75, 1999a.

J. Blazewicz, P. Formanowicz, K. Kasprzak, W. T. Markeiwicz, and J. Weglarz. DNA sequencing with positive and negative errors. *Journal of Computational Biology*, 6(1): 113–123, 1999b.

J. Blazewicz, J. Kaczmarek, K. Kasprzak, W. T. Markeiwicz, and J. Weglarz. Sequential and parallel algorithms for DNA sequencing. *CABIOS*, 13:151–158, 1997.

S. Brenner. *Methods for sorting polynucleotides using oligonucleotide tags*, US Patent 5,604,097.

K. Cameron. Induced matchings. *Discrete Applied Mathematics*, 24:97–102, 1989.

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 1997. ACM Press. ISBN 0-89791-882-7.

A. Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, February 1999. ISSN 0895-4801 (print), 1095-7146 (electronic). URL `http://epubs.siam.org/sam-bin/dbq/article/31994`.

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.

D. A. Christie. A 3/2-approximation algorithm for sorting by reversals. In *Proc. ninth annual ACM-SIAM Symp. on Discrete Algorithms (SODA 98)*, pages 244–252. ACM Press, 1998.

D. A. Christie. *Genome Rearrangement Problems*. PhD thesis, University of Glasgow, 1999.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

T. Dobzhansky and A. H. Sturtevant. Inversions in the chromosomes of *drosophila pseudoobscura*. *Genetics*, 23:28–64, 1938.

K. Doi and H. Imai. Sequencing by hybridization in the presence of hybridization errors. In *Proceedings of the Workshop on Genome Informatics (GIW '00)*, volume 11, pages 53–62, 2000.

R. Drmanac, I. Labat, L. Brukner, and R. Crkvenjakov. Sequencing of megabase plus DNA by hybridization: Theory and method. *Genomics*, 4:114–128, 1989.

R. Drmanac, G. Lennon, S. Drmanac, I. Labat, R. Crkvenjakov, and H. Lehrach. Partial sequencing by oligohybridization: Concept and applications in genome analysis. In *Proceedings of the first international conference on electrophoresis supercomputing and the human genome Edited by C. Cantor and H. Lim*, pages 60–75, Singapore, 1991. World Scientific.

M. Dyer, A. Frieze, and S. Suen. The probability of unique solution of sequencing by hybridization. *Journal of Computational Biology*, 1:105–110, 1994.

N. Eriksen. $(1+\epsilon)$-approximation of sorting by reversals and transpositions. *Theoretical Computer Science*, 289:517–529, 2002.

N. Eriksen, D. Dalevi, S. G. E. Andersson, and K. Eriksson. Gene order rearrangements with derange: weights and reliability, 2001. Manuscript.

H. Eriksson, K. Eriksson, J. Karlander, L. Svensson, and J. Wastlund. Sorting a bridge hand. *Discrete Mathematics*, 241(1-3):289–300, 2001. ISSN 0012-365X.

U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. In *Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS)*, pages 105–115, 2000.

A. Frieze and B. Halldorsson. Optimal sequencing by hybridization in rounds. *Journal of Computational Biology*, 9(2):355–369, 2002.

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

N.P. Gerry, N.E. Witowski, J. Day, R.P. Hammer, G. Barany, et al. Universal DNA microarray method for multiplex detection of low abundance point mutations. *J. Mol. Biol.*, 292(2):251–62, 1999.

D.M. Grant and M.S. Phillips. *Technologies for the Analysis of Single-Nucleotide Polymorphisms: An Overview*. Marcel Dekker, Inc., New York, 2001.

Q. P. Gu, S. Peng, and H. Sudborough. A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science*, 210(2): 327–339, 1999.

J.G. Hacia. Resequencing and mutational analysis using oligonucleotide microarrays. *Nature Genetics*, 21(1):42–47, Jan 1999.

E. Halperin, S. Halperin, T. Hartman, and R. Shamir. Handling long targets and errors in sequencing by hybridization. In *Proceedings of the Sixth Annual International Conference on Computational Moleculaer Biology (RECOMB 2002)*, pages 176–185, 2002.

E. Halperin, S. Halperin, T. Hartman, and R. Shamir. Handling long targets and errors in sequencing by hybridization. *Journal of Computational Biology*, 10(3-4):483–497, 2003.

S. Hannenhalli, W. Feldman, H. F. Lewis, S. S. Skiena, and P. A. Pevzner. Positional sequencing by hybridization. *CABIOS*, 12(1):19–24, 1996.

S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46:1–27, 1999. (Preliminary version in Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing 1995 (STOC 95), pages 178–189).

T. Hartman. A simpler 1.5-approximation algorithm for sorting by transpositions. In *Proc. 14th Annual Symposium on Combinaotrial Pattern Matching (CPM '03)*, pages 156–169. Springer, 2003.

T. Hartman and R. Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions, 2004. submitted to the *SIAM journal on Computing*.

T. Hartman and R. Sharan. A 1.5-approximation algorithm for sorting by transpositions and transreversals, 2004a. submitted to the *Journal of Computer and System Sciences*.

T. Hartman and R. Sharan. A 1.5-approximation algorithm for sorting by transpositions and transreversals. In *Proc. 4th Workshop on Algorithms in Bioinformatics (WABI'04)*, 2004b. To appear.

M. I. Honda. Implementation of the algorithm of Hartman for the problem of sorting by transpositions, 2004. Master Thesis.

S. B. Hoot and J. D. Palmer. Structural rearrangements, including parallel inversions, within the chloroplast genome of Anemone and related genera. *J. Molecular Evooution*, 38:274–281, 1994.

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, 2000. (Preliminary version in Proceedings of the eighth annual ACM-SIAM Symposium on Discrete Algorithms 1997 (SODA 97), ACM Press, pages 344–351).

H. Kaplan and E. Verbin. Effficient data structures and a new randomized approach for sorting signed permutations by reversals. In *Proc. 14th Annual Symposium on Combinaotrial Pattern Matching (CPM '03)*, pages 170–185. Springer, 2003.

J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1/2):180–210, January 1995. A preliminary version appeared in *Proc. CPM93*, Springer, Berlin, 1993, pages 87–105.

V. N. Kristensen, N. Harada, N. Yoshimura, E. Haraldsen, P. E. Lonning, B. Erikstein, T. Kristensen R. Kresen, and A. L. Brresen-Dale. Genetic variants of cyp19 (aromatase) and breast cancer risk. *Oncogene*, 19(10):1329–33, March 2000.

ES. Lander et al. Initial sequencing and analysis of the human genome. international human genome sequencing consortium. *Nature*, 409:860–921, 2001.

W. H. Li and D. Graur. *Fundamentals of Molecular Evolution*. Sinauer, Sunderland, MA, 1991.

G. H. Lin and G. Xue. Signed genome rearrangements by reversals and transpositions: Models and approximations. *Theoretical Computer Science*, 259:513–531, 2001.

R. J. Lipshutz. Likelihood DNA sequencing by hybridization. *J Biomolecular Str. Dyn.*, 11:637–653, 1993.

D. Loakes and D. M. Brown. 5-Nitroindole as a universal base analogue. *Nucleic Acids Research*, 18:2653–2660, 1990.

D. Matula and L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30:417–427, 1983.

J. Meidanis, M. E. Walter, and Z. Dias. Reversal distance of signed circular chromosomes. manuscript, 2000.

O. J. Miller and E. Therman. *Human Chromosomes*. Springer-Verlag, 4th edition, 2001.

M. S. Morris, D. D. Shoemaker, R. W. Davis, and M. P. Mittmann. *Methods and compositions for selecting tag nucleic acids and probe arrays*, European Patent Application 97,302,313.

National Center for Biotechnology Information. Human genome sequencing, 2002a. `http://www.ncbi.nlm.nih.gov/genome/seq/`.

National Center for Biotechnology Information. The unigene database, 2002b. `http://www.ncbi.nlm.nih.gov/UniGene/`.

J. D. Palmer and L. A. Herbon. Tricircular mitochondrial genomes of Brassica and Raphanus: reversal of repeat configurations by inversion. *Nucleic Acids Research*, 14: 9755–9764, 1986.

J. D. Palmer and L. A. Herbon. Unicircular structure of the Brassica hirta mitochondrial genome. *Current Genetics*, 11:565–570, 1987.

J. D. Palmer and L. A. Herbon. Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *J. Molecular Evolution*, 28:87–97, 1988.

J. D. Palmer, B. Osorio, and W.R. Thompson. Evolutionalry significance of inversions in legume chorloplast DNAs. *Current Genetics*, 14:65–74, 1988.

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 2000. AAAI Press.

P. A. Pevzner. l-tuple DNA sequencing: computer analysis. *J. Biomol. Struct. Dyn.*, 7: 63–73, 1989.

P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach.* MIT Press, 2000.

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.

P. A. Pevzner and M. S. Waterman. Open combinatorial problems in computational molecular biology. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems (ISTCS)*, pages 158–173, 1995.

V. T. Phan and S. S. Skiena. Dealing with errors in interactive sequencing by hybridization. *Bioinformatics*, 17(10):862–870, 2001.

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.

F. Preparata and E. Upfal. Sequencing by hybridization at the information theory bound: An optimal algorithm. *Journal of Computational Biology*, 7(3-4):621–630, August 2000.

N. J. Risch. Searching for genetic determinants in the new millennium. *Nature*, 405(6788): 847–56, 2000.

D. Sankoff and N. El-Mabrouk. Genome rearrangement. In T.Jiang, T. Smith, Y. Xu, and M. Q. Zhang, editors, *Current Topics in Computational Molecular Biology*. MIT Press, 2002.

J. P. Schouten, C. J. McElgunn, R. Waaijer, D. Zwijnenburg, F. Diepvens, and G. Pals. Relative quantification of 40 nucleic acid sequences by multiplex ligation-dependent probe amplification. *Nucleic Acids Research*, 30(12), June 2002.

J. Setubal and J. Meidanis. *Introduction to Computational Biology*. PWS Publishing Co., 1997.

R. Shamir. Algorithms in molecular biology: Lecture notes, 2002. Available at http://www.math.tau.ac.il/∼rshamir/algmb/01/algmb01.html.

R. Shamir and D. Tsur. Large scale sequencing by hybridization. *Journal of Computational Biology*, 9(2):413–428, 2002.

S. S. Skiena and G. Sundaram. Reconstructing strings from substrings. *J. Comput. Biol.*, 2:333–353, 1995.

D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. Assoc. Comput. Mach.*, 32:652–686, 1985.

S. Snir, E. Yeger-Lotem, B. Chor, and Z. Yakhini. Using restriction enzymes to improve sequencing by hybridization. Technical Report CS-2002-14, Technion, Haifa, Israel, 2002.

A. Solomon, P. Sutcliffe, and R. Lister. Sorting circular permutations by reversal. In *Proc. Workshop on Algorithms and Data Structures (WADS '03)*. Springer, 2003.

A. C. Syvanen. From gels to chips: "minisequencing" primer extension for analysis of point mutations and single nucleotide polymorphisms. *Hum. Mutat.*, 13(1):1–10, 1999.

E. Tannier and M. Sagot. Sorting by reversals in subquadratic time. In *Proc. 15th Annual Symposium on Combinaotrial Pattern Matching (CPM '04)*, pages 1–13. Springer, 2004.

D. Tsur. Bounds for resequencing by hybridization. In *Proceedings of the Third Workshop on Algorithms in Bioinformatics (WABI' 03)*, pages 498–511, 2003a.

D. Tsur. Sequencing by hybridization in few rounds. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA '03)*, pages 506–516, 2003b.

S. Venitt. Mechanisms of carcinogenesis and individual susceptibility to cancer. *Clin. Chem.*, 40(7.2):1421–5, July 1994.

S. Venitt. Mechanisms of spontaneous human cancers. *Environ. Health Perspect.*, 104(3): 633–7, May 1996.

JC. Venter et al. The sequence of the human genome. *Science*, 291:1304–1351, 2001.

M. E. Walter, Z. Dias, and J. Meidanis. Reversal and transposition distance of linear chromosomes. In *String Processing and Information Retrieval: A South American Symposium (SPIRE 98)*, 1998.

M. E. Walter, Z. Dias, and J. Meidanis. A new approach for approximating the transposition distance. In *String Processing and Information Retrieval: A South American Symposium (SPIRE 00)*, 2000.

M. E. Walter, M. I. Honda, and L. S. Soares. Working on the algorithm of Hartman for the problem of sorting by transpositions. submitted.

M. E. Walter, L. Reginaldo, A. F. Curado, and A. G. Oliveira. Working on the problem of sorting by transpositions on genome rearrangements. In *Proc. 14th Annual Symposium on Combinaotrial Pattern Matching (CPM '03)*, pages 372–383. Springer, 2003.

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. Watanabe, A. Fujiyama, Y. Ichiba, M. Hattori, T. Yada, Y. Sakaki, and T. Ikemura. Chromosome-wide assessment of replication timing for human chromosomes 11q and 21q: disease-related genes in timing-switch regions. *Human Molecular Genetics*, 11(1): 13–21, January 2002.