

An Optimal Method for DNA Sequencing by Hybridization

by

Samuel Aaron Heath

B.Sc., Computer Science, University of British Columbia, 1998

Sc.M., Computer Science, Brown University, 2000

A dissertation submitted in partial fulfillment of the  
requirements for the Degree of Doctor of Philosophy  
in the Department of Computer Science at Brown University

Providence, Rhode Island

May 2003

UMI Number: 3087272

UMI<sup>®</sup>

---

UMI Microform 3087272

Copyright 2003 by ProQuest Information and Learning Company.

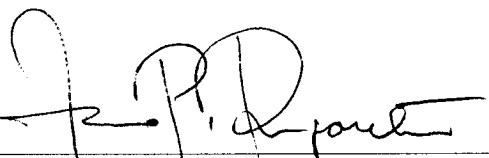
All rights reserved. This microform edition is protected against  
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company  
300 North Zeeb Road  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

© Copyright 2003 by Samuel Aaron Heath

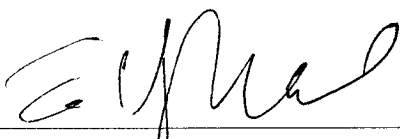
This dissertation by Samuel Aaron Heath is accepted in its present form by  
the Department of Computer Science as satisfying the dissertation requirement  
for the degree of Doctor of Philosophy.

Date Feb 13, 2003

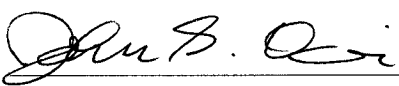
  
\_\_\_\_\_  
Franco P. Preparata, Director

Recommended to the Graduate Council

Date 2/13/03


  
\_\_\_\_\_  
Eli Upfal, Reader

Date 2/13/03

  
\_\_\_\_\_  
John M. Oliver, Reader

Approved by the Graduate Council

Date 2/23/03

  
\_\_\_\_\_  
Karen Newman  
Dean of the Graduate School



# Abstract

Sequencing by Hybridization (SBH) is a potentially powerful method for sequencing unknown DNA. An information-theoretic argument indicates that SBH can be used to sequence DNA fragments orders of magnitude longer than existing methods. Although originally proposed in 1988, SBH was unable to fulfill its promise until recently, with the invention of *gapped probes* containing biochemical wildcard nucleotides called *universal bases*. Using gapped probes and novel algorithms exploiting their structure, SBH performance has been able to formally approach the theoretic bound. This research is concerned with the development, analysis and refinement of algorithms which push performance closer to the bound, for randomly generated and natural DNA sequences.

We present a novel analysis of the branching behaviour of the gapped SBH algorithm, and of the two failures modes of the sequencing algorithm. We also present several enhancements to the basic gapped SBH algorithm. Two of these enhancements improve performance on random DNA, and a third is designed to identify and recover from failures which occur only in natural sequences. The aggregate result of the improvements presented is to push the overall effectiveness of the algorithm on random data to almost  $2/3$  of the proven theoretic bound, and to significantly reduce the gap between random and natural sequence reconstruction.

# Acknowledgements

The person most of all responsible for alternately prodding and guiding me through grad school is my advisor and mentor, Franco Preparata. Without a lot of good advice from him, I never would have finished. And if I had heeded more of his advice, I would have finished much sooner. I also owe thanks to Eli Upfal and John Oliver for their time and patience over the last year or two. Joel Young contributed valuable ideas and was one of my only collegial peers during the development of this thesis. Without my former research advisors, Leslie Kaelbling and Craig Boutilier, I never would have even started this project; they started me down the research road. Kevin Ingersoll micro-edited the first two chapters, and kept me alive with many home-cooked dinners. A thousand thanks to Brian Perkins—one for every trip to Thayer Street for hot dogs or ice cream when I needed a break.

Finally, thanks to my family—Mom, Dad, Brander, Simon, Matthew and Cameron—for being there not only while I was researching and writing this thesis, but whenever I needed them for anything at all.

The Road goes ever on and on  
Down from the door where it began.  
Now far ahead the Road has gone,  
And I must follow, if I can,  
Pursuing it with eager feet,  
Until it joins some larger way  
Where many paths and errands meet.  
And whither then? I cannot say.  
—J. R. R. Tolkien

# Contents

<b>List of Figures</b>	<b>x</b>
<b>1 Background and Introduction</b>	<b>1</b>
1.1 Chapter List & Overview . . . . .	2
1.2 What is DNA? . . . . .	4
1.2.1 Amplification and Fragmentation . . . . .	6
1.3 DNA Sequencing . . . . .	7
1.4 The Invention of Sequencing by Hybridization . . . . .	9
1.4.1 DNA Microarrays . . . . .	10
1.4.2 The SBH Algorithm . . . . .	12
1.5 Universal Bases and Gapped Probes . . . . .	20
1.6 SBH with Gapped Probes . . . . .	25
<b>2 Implementation of the Gapped-SBH Algorithm</b>	<b>33</b>
2.1 C++ Pseudo-code Usage . . . . .	33
2.2 Overview of the Sequencing Algorithm . . . . .	36
2.2.1 Probe Structure . . . . .	36
2.2.2 Spectrum Structure . . . . .	40
2.3 The EXTEND Algorithm . . . . .	44
2.4 The SEQUENCE Algorithm . . . . .	55
2.5 Changing Direction: Reverse Sequencing . . . . .	56
2.6 Sequencing Failure Modes . . . . .	59
2.7 Logging Algorithmic Behaviour . . . . .	64
2.8 Initial Performance Observations . . . . .	69

<b>3</b>	<b>Analysis of the Gapped-SBH Algorithm</b>	<b>72</b>
3.1	Finding a Fooling Probe . . . . .	73
3.2	Overlapping Probes . . . . .	74
3.3	Tree Size and Work . . . . .	78
3.3.1	Branching Events . . . . .	81
3.3.2	Reverse Probes . . . . .	84
3.3.3	Direct Probes . . . . .	94
3.3.4	Segment Length, Tree Size and Work . . . . .	109
<b>4</b>	<b>Failure Modes of the Gapped-SBH Algorithm</b>	<b>118</b>
4.1	Sequencing Failures . . . . .	119
4.2	Mode 1 Failures . . . . .	120
4.3	Introduction to Mode 2 Failures . . . . .	125
4.4	Probability of Mode 2 Failures . . . . .	127
4.4.1	Reverse Probes . . . . .	131
4.4.2	Direct Probes . . . . .	135
4.5	Predicted Probability of Success . . . . .	140
4.6	Measuring Performance on Natural DNA . . . . .	142
4.6.1	Selection of Natural DNA Sources . . . . .	142
4.6.2	Observed SBH Results for Natural DNA . . . . .	145
4.6.3	Some Comments on SBH with Natural DNA . . . . .	147
<b>5</b>	<b>Resolving Ambiguous Extensions by Polling</b>	<b>151</b>
5.1	Definition of Used Probes . . . . .	152
5.2	Polling and Mode 1 Failures . . . . .	153
5.3	Polling and Mode 2 Failures . . . . .	156
5.4	Overall Performance Improvement . . . . .	157
5.5	Using PCR Primers . . . . .	160
5.6	Effectiveness of Polling on Natural DNA . . . . .	164
<b>6</b>	<b>Identifying and Escaping from Repeating Segments</b>	<b>167</b>
6.1	Recognizing Repeating Segments . . . . .	167
6.2	Causes of Repeat Failures . . . . .	170
6.3	Variations on Repeat Failure . . . . .	174

6.4	The REPEATRECOVERY Algorithm . . . . .	177
6.5	Results and Observations . . . . .	181
<b>7</b>	<b>Pooling Information from Multiple Spectra</b>	<b>184</b>
7.1	Choice of Probing Patterns . . . . .	185
7.2	The Multi-spectrum SBH Algorithm . . . . .	185
7.3	Probability of Failure with Tandem Spectra . . . . .	187
7.3.1	Mode 1 Failures . . . . .	188
7.3.2	Mode 2 Failures . . . . .	189
7.4	Performance Improvement . . . . .	191
7.5	Integration with Polling and Repeat-Recovery . . . . .	193
7.5.1	Performance on Natural DNA . . . . .	195
<b>8</b>	<b>Constructing a Seed</b>	<b>198</b>
8.1	Constructing a Seed . . . . .	199
8.1.1	Detailed Example of seed Construction . . . . .	200
8.2	Complexity of the Basic seed Algorithm . . . . .	204
8.2.1	Different Shift Strategies . . . . .	210
8.2.2	Spectrum Scans . . . . .	213
8.2.3	Searching for a Better Fill Strategy . . . . .	218
8.3	Probe Structure . . . . .	221
8.3.1	Autocorrelation and Seed Strategy . . . . .	223
<b>9</b>	<b>Conclusion &amp; Future Work</b>	<b>226</b>
9.1	Contributions . . . . .	227
9.2	Future Work . . . . .	229
<b>A</b>	<b>GenBank Accession Numbers</b>	<b>232</b>
<b>B</b>	<b>Obtaining the Simulator Source Code</b>	<b>233</b>
B.1	Obtaining the Source Code . . . . .	233
<b>C</b>	<b>Model of Duplicated Subsequences in DNA</b>	<b>235</b>
C.1	Natural DNA and Models . . . . .	235
C.2	Suffix Trees . . . . .	236

C.3	Markov Models and Entropy . . . . .	238
C.4	Building a Model . . . . .	241
C.4.1	Length of duplicate strings: $\tau$ . . . . .	243
C.4.2	Gap between duplicated strings (intra-gap spacing): $t$ . . . . .	245
C.4.3	Gap between events (inter-gap spacing): $T$ . . . . .	246
C.5	Evaluating the Model . . . . .	246
<b>Bibliography</b>		<b>248</b>

# List of Figures

1.1	Stylized rendering of the DNA double-helix. . . . .	4
1.2	Diagram of electrophoresis sequencing gel. . . . .	9
1.3	Small microarray with $4^2 = 16$ features. . . . .	10
1.4	Illustration of SBH preparation steps. . . . .	12
1.5	Spectrum of unknown sequence using 4-mer probes. . . . .	13
1.6	Hamiltonian path SBH example. . . . .	14
1.7	Eulerian path SBH example. . . . .	15
1.8	Graph with an unambiguous Eulerian path. . . . .	18
1.9	Graph with two valid Eulerian paths. . . . .	19
1.10	Comparing performance ungapped and gapped-SBH. . . . .	32
2.1	Diagram of the branching EXTEND algorithm. . . . .	46
2.2	Performance of gapped SBH on natural DNA using (4,4)-probes. . . .	70
3.1	Plots of $\alpha_{(\kappa,m)}$ for various $\kappa$ . . . . .	75
3.2	Autocorrelation function for (4,4)-probing pattern. . . . .	76
3.3	Hypothetical path tree for (4,3)-direct probes. . . . .	81
3.4	Path tree for (4,3)-direct probes with $\beta$ multipliers. . . . .	87
3.5	Diagram of primary and secondary path trees. . . . .	89
3.6	Primary and secondary trees for (3,3)-reverse probes. . . . .	90
3.7	Tree width for (4,4)-reverse probes. . . . .	95
3.8	Primary and secondary trees for (3,3)-direct probes. . . . .	96
3.9	A path tree consisting of two independent groups. . . . .	102
3.10	Query regions for forward probing patterns. . . . .	103
3.11	Example forward-probe query in region 2. . . . .	104
3.12	Tree width for (4,4)-direct probes ( $m = 12000$ ). . . . .	108

3.13	Tree width for (4,4)-direct probes ( $m = 15000$ ) . . . . .	109
3.14	Tree depth for (4,4)-reverse probes. . . . .	114
3.15	Tree depth for (4,4)-direct probes. . . . .	116
4.1	Incidence of Mode 1 failures with (4,4) probes. . . . .	124
4.2	Incidence of Mode 2 failures for $(s, r)$ -reverse probes. . . . .	134
4.3	Incidence of Mode 2 failures for $(s, r)$ -forward probes. . . . .	139
4.4	Predicted probability of success for all $(s, r)$ -probes. . . . .	141
4.5	Predicted probability of success for (4,4)-probes. . . . .	141
4.6	Relative proportion of Mode 1 and Mode 2 failures (4,4). . . . .	142
4.7	Relative proportion of Mode 1 failures for $(s, r)$ -probes. . . . .	143
4.8	Performance of gapped-SBH on natural DNA 1. . . . .	147
4.9	Performance of gapped-SBH on natural DNA 2. . . . .	148
4.10	Performance of gapped-SBH on natural DNA 3. . . . .	149
5.1	Sequencing performance with polling provision. . . . .	159
5.2	Probability of incorrect polling choice in $m$ -character sequence. . . . .	160
5.3	Polling failure as a function of $m$ and failure point. . . . .	161
5.4	Probability of polling failure with bi-directional sequencing. . . . .	162
5.5	Observed performance of polling provision 2. . . . .	165
5.6	Observed performance of polling provision 2. . . . .	166
6.1	Observed performance of repeat recovery 1. . . . .	182
6.2	Observed performance of repeat recovery 2. . . . .	183
7.1	Cross-correlation of a (5,3)-direct and (5,3)-reverse probe. . . . .	186
7.2	Performance of tandem-spectrum sequencing with (4,4)-probes. . . . .	191
7.3	Tandem-spectrum sequencing with polling using (4,4)-probes. . . . .	195
7.4	Performance of tandem-spectrum sequencing with (4,4)-probes 1. . . . .	196
7.5	Performance of tandem-spectrum sequencing with (4,4)-probes 2. . . . .	197
8.1	Autocorrelation function for (4,4)-probing pattern. . . . .	222
8.2	Autocorrelation function for (5,3)-probing pattern. . . . .	223
8.3	Autocorrelation function for modified (7,1)-probing pattern. . . . .	224



9.1	Single and tandem sequencing performance with and without polling	228
C.1	The suffix tree for a 200Kb random DNA sequence shows a full tree up to length 7, followed by a probabilistic exponential falloff for longer strings. . . . .	237
C.2	Suffix trees for natural DNA demonstrate an increased incidence of long duplicated strings. . . . .	238
C.3	Duplicate strings in <i>e. coli</i> . . . . .	240
C.4	The suffix tree for human coding sequences shows several extremely long repeated strings. . . . .	243
C.5	The histograms of the lengths of duplicate strings observed in <i>e. coli</i> (left) and predicted by a blended model (right). . . . .	244
C.6	Comparing natural and modeled <i>e. coli</i> and human coding sequences.	247

# Chapter 1

## Background and Introduction

DNA carries all of the information needed by every organism to guide it from conception through its entire life. From minor variations which distinguish individuals of a species, to the differences between bacteria, houseplants, insects and humans, everything comes down to changes in the sequence of DNA. Responses to diseases (and the causes of many diseases) are encoded in DNA. The first step in understanding the processes of life, disease and death is to read the sequence of DNA ourselves: this is DNA sequencing.

Traditional methods of DNA sequencing are labour-intensive, allowing DNA fragments of only about 600 base pairs to be processed in a single experiment. Each experiment requires the time and expertise of a trained lab technician, and the results of the experiments can contain errors. DNA *Sequencing by Hybridization* or *SBH*, is a relatively new method for performing *de novo* DNA sequencing. It was originally proposed over ten years ago by four different research groups [BS91, L+88, D89, P89, P91] to solve some of the problems inherent in traditional sequencing methods. It attempts to reconstruct a fragment of DNA from the complete family (or set) of the fragment's subsequences. SBH consists of two steps: the first is biochemical and allows the acquisition of the complete family of a fragment's subsequences. Following this initial step, the computational process of reconstructing the sequence itself begins.

This dissertation describes an advanced method for DNA Sequencing by Hybridization which is orders of magnitude faster than current techniques. In 2000, Preparata, Frieze & Upfal [PU00] developed the basic SBH method, which achieved

asymptotically optimal performance. However, the potential applications of this technique are important enough that incremental increases in effectiveness can be very valuable. The current work describes several improvements to the basic SBH algorithm which improve overall performance by nearly 50%.

SBH is a process which relies first on the physical resource of a *DNA microarray* and the expertise of a wet-lab technician to conduct an initial biochemical experiment. After the initial (and only) laboratory experiment, the SBH process is entirely computational. The primary aim of the research described in this dissertation is to maximize the length of sequences which may be unambiguously reconstructed using fixed physical resources, while minimizing the computational resources required. When a complete and unambiguous reconstruction proves to be impossible, we attempt to maximize the proportion of the sequence which *can* be produced, while minimizing potential errors.

## 1.1 Chapter List & Overview

This document contains the following chapters:

- **Chapter 1 - Background and Introduction.** The current chapter contains a brief introduction to DNA, and three processes for manipulating it: *hybridization*, *amplification* and *fragmentation*. After discussing electrophoresis-based sequencing, Sequencing by Hybridization is introduced with a discussion of the original ungapped-SBH method. The chapter concludes with a non-technical discussion of gapped-SBH and its advantages over the ungapped method.
- **Chapter 2 - Implementation of the Gapped SBH Algorithm.** This chapter describes the specific implementation of the gapped SBH algorithm which was used to conduct simulations of the method. The data structures used to represent individual probes and the sequence as it is reconstructed, and the container used to hold the spectrum are discussed. The different failure modes of the EXTEND algorithm are described along with the method by which they are detected. The chapter concludes with a short survey of sequencing results from simulations.

- **Chapter 3 - Analysis of the Gapped SBH Algorithm.** This chapter analyzes the complexity and performance of the SBH algorithm using gapped probes. It contains a detailed discussion of the effects of various parameters on sequencing, specifically the probing pattern used and the breadth ( $B$ ) and depth ( $H$ ) limits in the branching EXTEND algorithm.
- **Chapter 4 - Failure Modes of the Gapped SBH Algorithm.** The two primary types of failure (Mode 1 and Mode 2) encountered while sequencing uniform random sequences are explored, and compared with the failures typically encountered while sequencing natural DNA.
- **Chapter 5 - Recovering from Failures by Polling the Spectrum.** This is the first of three chapters dealing with failure-recovery: methods of continuing sequencing when the branching EXTEND algorithm fails. Chapter 5 describes a POLLING algorithm which exploits the information available in the spectrum during sequence reconstruction to recover Mode 1 and Mode 2 failures, which represent the vast majority of failures which occur in artificial maximum-entropy sequences. A numerical analysis is presented, along with experimental simulation results.
- **Chapter 6 - Identifying and Escaping from Repeating Segments.** Natural DNA contains many instances of short repeating segments (tandem repeats and microsatellites are two examples) which cause SBH to fail. This chapter describes an algorithm for identifying such repeating segments as they are encountered, and continuing sequencing beyond them. Experimental results of its effectiveness are also included.
- **Chapter 7 - Pooling Information from Multiple Spectra.** The performance of SBH can be improved by using information from two or more independent spectra. This chapter contains a description and analysis of an algorithm which performs SBH using multiple spectra.
- **Chapter 8 - Constructing a Seed.** This chapter describes a method for constructing a *seed* from the information contained in the spectrum of a sequence, allowing SBH to be used even in the absence of manually attached

primer sequences.

- **Chapter 9 - Conclusion and Future Work.** This chapter contains a summary of the performance increases afforded by the various algorithmic improvements presented herein. We also summarize the effectiveness of SBH on natural DNA, and present some possible solutions to the unique problems encountered when sequencing non-random sequences.

The rest of this chapter contains an introduction to DNA, traditional methods of sequencing, and SBH. We begin with a discussion of DNA.

## 1.2 What is DNA?

The genetic information of every cellular living organism is stored in its chromosomes. Each chromosome is a large double-stranded molecule of deoxyribonucleic acid—DNA—compacted by coiling and supercoiling into a protein matrix<sup>1</sup>. Chromosomal DNA consists of two long, paired strands of nucleotide bases. These strands form the famous double-helix configuration (see Figure 1.1) of the DNA molecule. There are only four of these bases, and their sequence encodes the entirety of an organism's genetic information: adenosine (A), cytosine (C), thymine (T) and guanine (G).

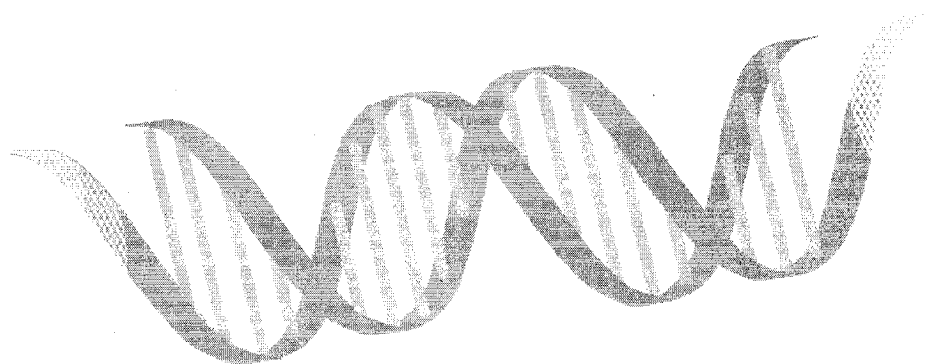


Figure 1.1: Stylized rendering of the DNA double-helix.

---

<sup>1</sup>This is true for eukaryotes. Prokaryotic chromosomes lack the protein matrix, but still contain the huge DNA molecules.

Although chromosomal DNA is always double stranded, single strands of nucleotides also exist in nature. During the process of cell division, the two strands of the helix are separated so that the chromosomes may be duplicated. In the laboratory, double-stranded DNA may be *denatured*: a process which separates the two strands of the helix by heating it to nearly 100 degrees Celsius; this is an important step in many wet-lab processes. Also, there is another form of nucleic acid called RNA (ribonucleic acid) which accomplishes various functions in the body, and is naturally single stranded<sup>2</sup>. The nucleotides which comprise any single-stranded nucleic acid may occur in any order, and the exact sequence of nucleotides in such a sequence can be neatly represented using the 4-character alphabet {A, C, T, G}.

While individual strands of nucleic acids can be created from arbitrary configurations of the four bases, pairwise bonds between bases in two different strands are essentially restricted to only two combinations: **A** forms cross-strand bonds only with **T** and **C** with **G**. These two pairs are *Watson-Crick complementary*. Mismatches are very rare, and are one of the primary causes of genetic mutations. We define the complement of a nucleotide sequence in terms of Watson-Crick base pairing.

**Definition 1.1.** *Two single-stranded nucleotide sequences are complementary if and only if every base in the first strand is the Watson-Crick complement of the corresponding base in the second strand.*

**Example 1.1.** The nucleotide sequence A-C-T-T-G is complementary to T-G-A-A-C because the bases at each position are complementary. From left to right, A pairs with T, C pairs with G, T pairs with A, and so on, as shown in the following table:

A	C	T	T	G
T	G	A	A	C

■

Complementary single-stranded sequences naturally form double-stranded molecules when they are brought together in solution. This phenomenon is called

---

<sup>2</sup>In RNA molecules, the nucleotide uracil (U) replaces thymine (T), but for the purposes of this thesis, the two nucleotides are functionally identical; all of the rules which govern thymine apply also to uracil.

*hybridization*, and is (unsurprisingly) the basis for SBH. We say that a sequence  $s$  *hybridizes* another sequence  $r$  if  $s$  is complementary to  $r$ .

### 1.2.1 Amplification and Fragmentation

Most biochemical procedures which measure or manipulate DNA, such as sequencing and genetic fingerprinting, require a large sample of identical DNA molecules in order to function. The process of taking a single sequence of DNA and producing a sufficiently large sample is called *amplification*. The primary method of DNA amplification used today is *polymerase chain reaction*, or PCR.

PCR is a process which is conducted entirely *in vitro*. Using PCR, a single target DNA sequence may be amplified by the repeated application of a few simple steps. Amplification occurs exponentially by repeatedly heating a DNA sample to denature the two strands, then stimulating the polymerization of the second strand complementary to each of the denatured originals from a solution of free nucleotides. Each time these steps are repeated, the size of the DNA sample doubles. In this way, only twenty steps suffice to amplify a single sequence more than a million ( $2^{20}$ ) times. PCR depends on foreknowledge of the exact sequence of short *primers*—sequences of about 20 bases—at both ends of the target sequence.

Once a sample has been sufficiently amplified, it is often useful to be able to break it into smaller pieces, or fragments, in a process called *fragmentation*. The process of DNA fingerprinting, for instance, uniquely identifies individuals by the length of fragments resulting from such a process. Fragmentation also allows large molecules of DNA to be broken down into more manageable pieces for the purposes of sequencing.

Amplified DNA may be fragmented by introducing chemicals or *restriction enzymes* to the sample. Chemical fragmentation produces probabilistic breaks at specific bases (a chemical might have a 1% chance to disrupt all bonds with thymine), whereas restriction enzymes deterministically cleave a molecule, but only at the location of a specific subsequence. A particular restriction enzyme might cleave a DNA sequence wherever it finds the string ATTAG, for instance. The means of fragmentation are secondary to the effects of fragmentation, however. The lengths of the fragments which result from enzymatic cleaving are used to determine a genetic fingerprint.

## 1.3 DNA Sequencing

DNA sequencing is the fundamental tool of molecular biology (and, of course, of bioinformatics). It allows the determination of the exact order of nucleotides along a segment of DNA, letting us ‘read’ a previously unknown sequence. Once read, a DNA sequence can be studied, measured, compared with other sequences, searched for potential genes, and stored in databases for later retrieval. Without the knowledge of the order of nucleotides along a sequence, bioinformaticians would essentially be blind.

The standard method of DNA sequencing currently used is *electrophoresis*, using either a gel medium or capillaries. In either case, the process results in a separation of molecules based on their size. DNA and RNA molecules have a natural charge, which allows them to be manipulated by an electric field. When molecules of different sizes are placed in a medium which impedes their movement—either the gel or capillaries—and exposed to an electric field, the smaller molecules are drawn more quickly through the medium than the larger ones. When the electric field is removed, the molecules are arranged in sequence according to size, with the smallest molecules having moved the farthest from their starting point.

It is possible to construct a set of all possible prefixes of a DNA sequence  $S$  using radioactively or fluorescently-labeled dideoxy *terminator* molecules. There are four such terminators: one for each of the natural bases. When inserted into a DNA sequence in place of a standard nucleotide, a terminator nucleotide prevents further biochemical extension of the sequence. The prefix-construction process is similar to a single PCR amplification step. A DNA sequence must first be amplified and then divided into four separate, but identical, samples. A small proportion (about  $\frac{1}{100}$  of the natural bases present) of a single dideoxy-terminator nucleotide is then introduced into each of the four samples. One sample contains dideoxy-thymine, one contains dideoxy-adenosine, etc... Each sample is denatured by heating, and then polymerization of the complementary strand is initiated.

In the sample containing dideoxy-thymine, there is a small chance (approximately 1%) that the terminator molecule replaces any particular occurrence of the standard thymine, and ends the polymerization process of a sequence. Since hundreds of thousands of copies of the target sequence are being simultaneously extended, the final



sample will contain sequences which—with very high probability—have been terminated at every possible location (after each thymine). Analogous processes occur in the other three samples.

**Example 1.2.** The sequence AGATAAGGCTAGTG is amplified and divided into four separate samples. The set of sequence fragments produced from the sample containing dideoxy-thymine is {AGAT, AGATAAGGCT, AGATAAGGCTAGT}. There is only a single sequence fragment produced from the sample containing dideoxy-cytosine: AGATAAGGC. ■

The four samples are then subjected to electrophoresis. The smaller fragments move farther through the gel or capillaries, so the increasing size of the fragments along the medium corresponds to longer and longer prefixes of the original sequence  $S$ . The radioactive or fluorescent labeling of the terminating molecules allows the last base in each sequence to be identified; fragments of the same size cluster together in a visible *spot* in the medium. Since all of the fragments are prefixes of the target sequence, the identity of each of the terminal bases, ordered from shortest to longest along the gel, gives the sequence of the string ( $S$ ).

**Example 1.3.** The sequence complete set of prefixes of the fragment AGGCATAGA consists of the following sequences in order of increasing size: { A, AG, AGG, AGGC, AGGCA, AGGCAT, AGGCATA, AGGCATAG, AGGCATAGA }. This is the same order in which they are arranged once electrophoresis has been completed (see Figure 1.2). Only the final base—the base which is radioactively or chemically dyed—of each prefix can be identified in the electrophoresis medium. The final bases of each of the prefix fragments, from smallest to largest, is: { A, G, A, T, A, C, G, G, A }. This sequence of nucleotides is identical to the order of bases in the original sequence. ■

The nature of the electrophoresis process limits the maximum length of DNA fragments to about 600bp<sup>3</sup>. The identity of the labeled terminator bases can only be read if the spots in the medium are sufficiently separated, and long sequences

---

<sup>3</sup>This document adopts *bp* as the standard abbreviation for base pairs. One thousand (kilo) base pairs is written *Kbp*, and one million (mega) base pairs *Mbp*.

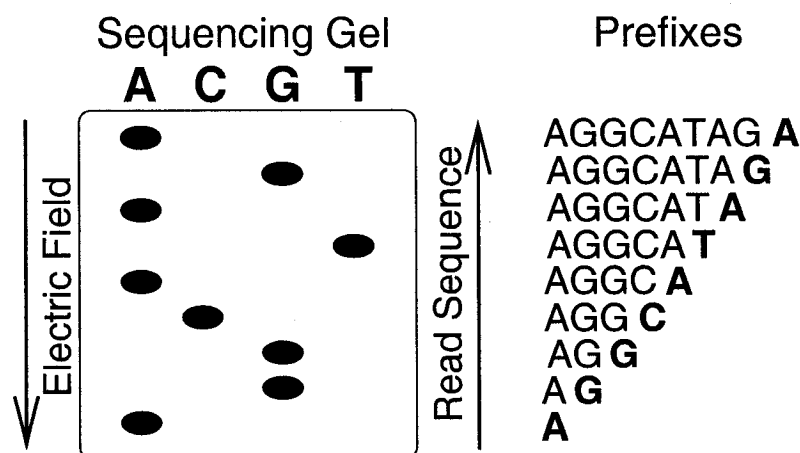


Figure 1.2: Simplified example of electrophoresis sequencing gel for the sequence AGGCATAGA, showing all spots on the gel and the prefix corresponding to each spot. The electric field pulls molecules down the gel, so the sequence is read from bottom (shortest prefix) to top (longest prefix).

produce spots which tend to blur together. For the human genome, the sequenceable fragments which have been used are only 543bp long.

Electrophoresis sequencing has become highly automated, with large machines able to perform all of the required steps without human intervention. These machines can sequence tens of thousands of bases per day with very few errors. However, the high cost of automated sequencing machines keeps these high-throughput processes out of reach for smaller research groups, which are still limited to single labour-intensive experiments.

## 1.4 The Invention of Sequencing by Hybridization

Around 1988, four different research groups [BS91, L+88, D89, P89] independently and simultaneously proposed a new method of DNA sequencing, hoping to solve some of the problems inherent in electrophoresis-based methods. The new method, called *DNA Sequencing by Hybridization*, or *SBH*, involves two steps: the first biochemical, the second computational. The biochemical step relies on small optical chips called *DNA microarrays*, which are also called *DNA chips*. SBH makes use of a single microarray to reveal the complete set of fixed-length subsequences present in a target

DNA sequence. The promise of SBH was that it would allow the sequencing of much longer DNA fragments in a single experiment.

The algorithmic phase of SBH—the subject of this dissertation—is a method for reconstructing a complete target sequence from its subsequences. This process is the central focus of this dissertation. It will be described in detail in Chapter 2, while Chapters 3 and 4 contain an analysis of its performance. The rest of this chapter merely provides some history of the development of SBH and the tools it requires, as well as a high-level overview of its function.

### 1.4.1 DNA Microarrays

Microarrays are small glass plates which contain thousands of short artificially synthesized DNA sequences. These synthetic DNA sequences, called probes, all have the same length  $\kappa$ , and so are also referred to as  $\kappa$ -mers<sup>4</sup>. The probes are arranged into a regular grid of features on the chip, where each feature contains many copies of the same  $\kappa$ -mer sequence. Using such a chip, a single simple experiment suffices to reveal whether *each* of the  $\kappa$ -mer probes on the microarray are present in a DNA sequence.

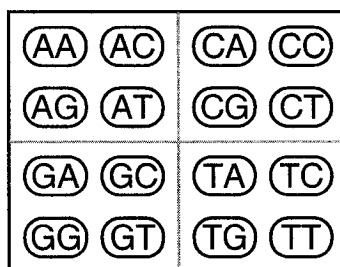


Figure 1.3: A microarray containing all  $4^2 = 16$  possible sequences of length 2. The four features in the top-left quadrant begin with A, in the top right-quadrant with C, in the bottom left with G and in the bottom right with T. Note that in this and subsequent figures, we adopt the convention of labeling features with the sequence which they will hybridize.

Although they were initially invented for the purpose of DNA sequencing, microarrays have many applications today. They are used to look for single-nucleotide polymorphisms (SNPs), to search for potential cancer-causing genes, and to trace

<sup>4</sup>An unbroken fragment of DNA is called an *oligomer*, from which  $\kappa$ -mer is derived.

the expression levels of different genes throughout the course of a disease. For these applications, the set of probes contained on a single DNA chip is carefully designed to yield results relevant to the problem being studied. Microarrays used for sequencing must contain *all* possible  $\kappa$ -mers ( $4^\kappa$  probes).

When they were first introduced, microarrays were technologically limited to a maximum of less than  $10^6$  features, limiting probes to a maximum of about 7 or 8 bases ( $4^8 \approx 6.5 \times 10^5$ ). Today, microarrays containing  $10^6$  features are common, permitting complete families of 10-base ( $4^{10} \approx 10^6$ ) oligonucleotides to be arrayed on a single chip. In fact, by using the appropriate masks, a chip containing  $4^\kappa$  features can be produced in only  $4 \cdot \kappa$  steps. Several companies produce microarrays commercially, and there are bench-top devices similar to inkjet printers which allow the automated creation of custom chips.

Once a microarray has been produced, the experimental procedure is simple. A fluorescently labeled DNA fragment (the target sequence) is amplified, fragmented and prepared in a solution. When the solution is brought into contact with a microarray, probes which are complementary to any subsequences of length  $\kappa$  in the target sequence will hybridize the fragments. An illustration of this process is shown in Figure 1.4.

After the hybridization reaction has completed, the microarray is chemically treated to highlight features where hybridization has occurred. The chip is then optically scanned to reveal the locations of features where hybridization has taken place. Since the identity and location of each of the synthetic probes on the chip is known, the locations on the chip where hybridization has occurred yields the identity of the probes which hybridized a subsequence of the target.

The microarrays used for sequencing simply contain all possible probes of length  $\kappa$ , requiring  $4^\kappa$  features: the target sequence is then stitched together from the complete set of its  $\kappa$ -base subsequences. The set of all probes of a given type is called a *complete family* of probes, and formally defined here:

**Definition 1.2.** *A complete family of probes is the set of all possible  $4^\kappa$  probes of a specific length  $\kappa$ .*

**Example 1.4.** The complete family of 2-nucleotide probes contains  $4^2 = 16$  probes: { AA, AC, AG, AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, TT }. ■

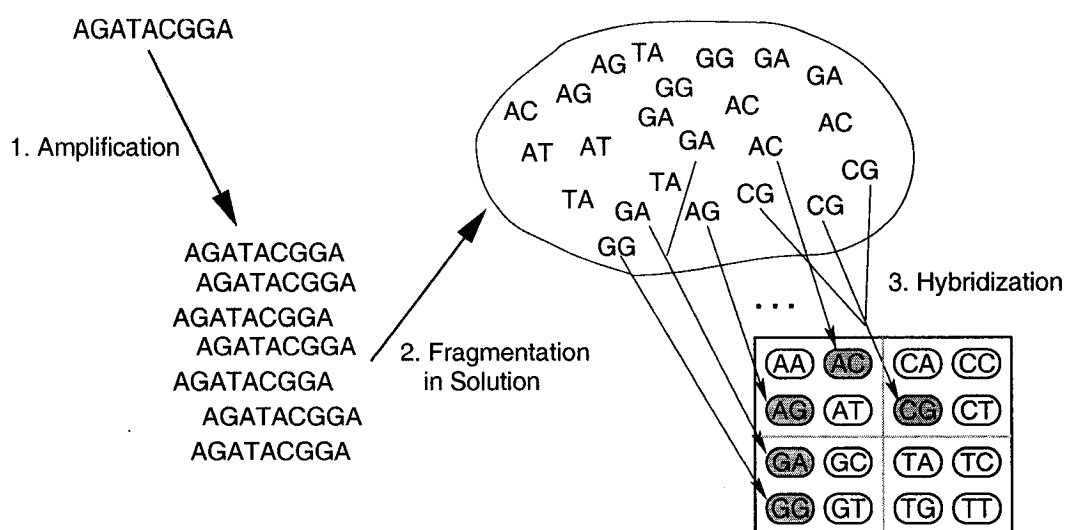


Figure 1.4: Illustration of the amplification, fragmentation and hybridization probes of a short DNA sequence. A single-stranded DNA string is amplified (Step 1) thousands of times, fragmented in solution (Step 2), and then placed in contact with a microarray, where the fragments hybridize (Step 3) at the appropriate spots.

The results of a single experiment using a microarray containing a complete family of  $\kappa$ -nucleotide probes reveals the complete set of  $\kappa$ -nucleotide subsequences of a target sequence. This is called the *spectrum* of a target sequence.

**Example 1.5.** The sequence  $s = \text{AAGCTGCTA}$  is amplified, fragmented, and brought into contact with a microarray containing a complete family of 4-nucleotide probes. The *spectrum* of the sequence  $s$  is the set of probes  $\{\text{AAGC}, \text{AGCT}, \text{GCTG}, \text{CTGC}, \text{TGCT}, \text{GCTA}\}$ . ■

Another microarray containing the spectrum of a longer, unknown sequence using a family of 256 probes (for  $\kappa = 4$ ) is shown in Figure 1.5. Once the spectrum of a target sequence has been revealed by hybridization with an appropriate microarray, the computational work of SBH begins.

### 1.4.2 The SBH Algorithm

Sequencing by Hybridization is the process of completely reconstructing a target sequence from its spectrum. Microarrays are the hardware which permit the spectrum

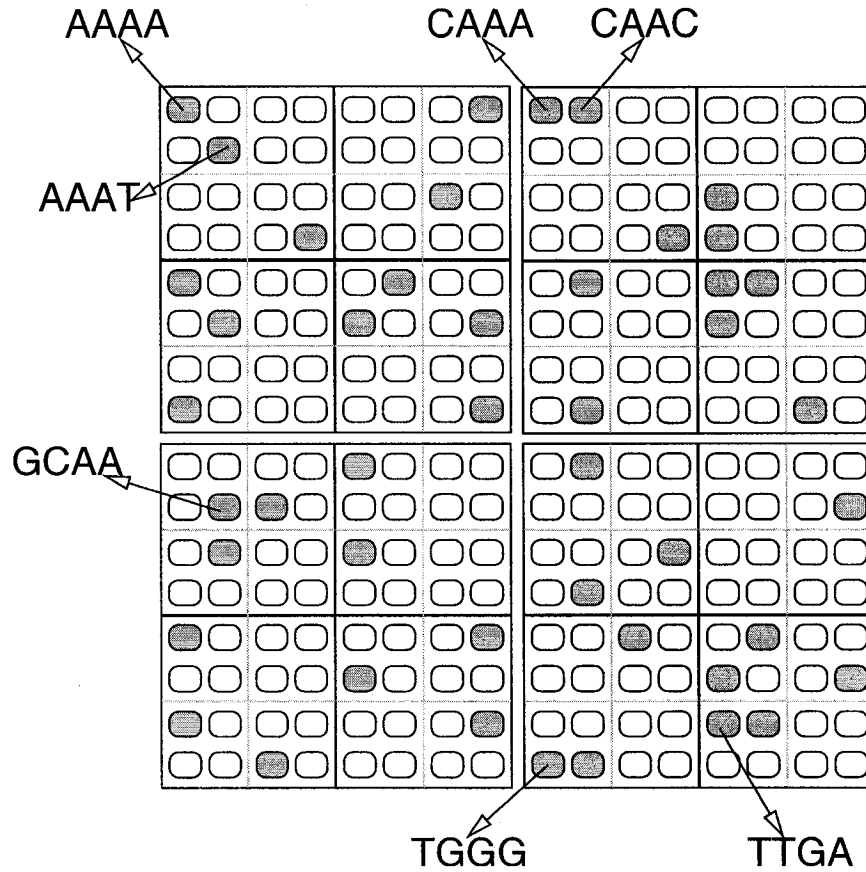


Figure 1.5: The spectrum of an unknown sequence in a family of 4-nucleotide probes. The features on the microarray where hybridization has occurred are shaded, and some of these features are identified by the sequences which they hybridized.

of a target sequence to be determined, but the reconstruction itself is performed by software. The promise of the SBH method was that it would allow DNA sequences of length  $O(4^\kappa)$  to be sequenced by means of a single microarray experiment. Electrophoresis sequencing methods are limited to fragments of only a few hundred bases in length. Using a microarray containing all possible 8-mers ( $\kappa = 8$ ), SBH has the theoretical potential for an increase of two orders of magnitude in fragment length over electrophoresis.

In 1989, Pevzner [P89] presented a graph-based algorithm which reconstructs a DNA sequence from its spectrum in polynomial time. Pevzner's method reduces the reconstruction process to finding an Eulerian path through a graph, which is a well-known and well-studied problem. The reconstruction algorithm does not actually

construct and explicitly traverse a graph, but the sequencing process implicitly finds an Eulerian path as it stitches a sequence together from its  $\kappa$ -mers. We will first describe the theoretical graph solution to the problem, and then describe an algorithm which actually reconstructs a sequence from its spectrum.

Initially, the SBH reconstruction process appeared to be a Hamiltonian path problem, to which there are no known efficient solutions. The simplest graph-based approach to SBH would begin by constructing a graph with  $\kappa$ -mers as nodes. Directed edges would be added between nodes where the  $(\kappa - 1)$ -suffix of one node corresponds to the  $(\kappa - 1)$ -prefix of another. For instance, from a spectrum of 5-mers, the two probes  $p_1 = \text{GAGGT}$  and  $p_2 = \text{AGGTC}$  would generate two nodes  $n_1$  and  $n_2$  with an edge from  $n_1$  to  $n_2$ , since the 4-suffix of node  $n_1 = \text{AGGT}$  matches the 4-prefix of node  $n_2 = \text{AGGT}$ . A target sequence may be reconstructed from such a graph by finding a Hamiltonian path which visits each node exactly once. Figure 1.6 shows how a simple Hamiltonian path reconstruction would work.

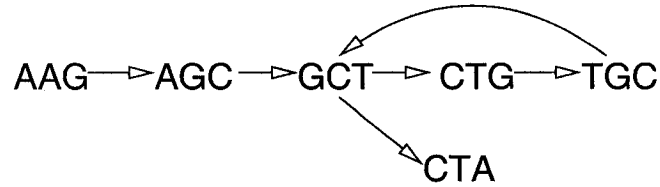


Figure 1.6: The graph of the sequence AAGCTGCTA from a spectrum of 3-mers, using 3-mers to label the nodes. There is only a single Hamiltonian path through all nodes.

Pevzner recognized that the problem could be reformulated as an Eulerian path problem by changing the structure of the underlying graph. Pevzner's reconstruction algorithm also begins with a spectrum of  $\kappa$ -mers. However, the graph  $G$  is constructed with nodes corresponding to  $(\kappa - 1)$ -mers; each probe in the spectrum yields two nodes in the graph. The edges of the graph are oriented to point from the  $(\kappa - 1)$  prefix of a  $\kappa$ -mer probe to the  $(\kappa - 1)$ -suffix of the same probe. For instance, if a spectrum of 6-mers contains the probe AGGTAG, the graph  $G$  must contain two nodes  $n_1 = \text{AGGTA}$  and  $n_2 = \text{GGTAG}$ , with a directed edge  $e_1$  from  $n_1$  to  $n_2$ . Using this graph, the target sequence can be reconstructed by finding an Eulerian path which traverses each edge in the graph exactly once, and there are well-known efficient algorithms for

finding Eulerian paths. Figure 1.7 shows the Eulerian path version of the Hamiltonian path reconstruction in Figure 1.6.

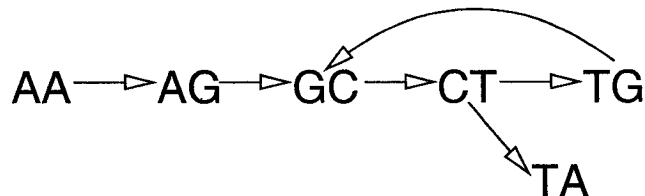


Figure 1.7: The Eulerian-path graph of the sequence AAGCTGCTA from a spectrum of 3-mers. Although this graph is constructed from the same spectrum as the Hamiltonian graph in Figure 1.6, the nodes in this graph are identified only by 2-mers.

In practice, it is infeasible to first construct a complete graph from the spectrum of a target sequence and then perform a complete traversal. More to the point, it is unnecessary to do so. The process of finding an Eulerian path through the appropriate spectrum can be performed implicitly, beginning with an arbitrary  $\kappa$ -mer, and *extending* the initial  $\kappa$ -mer one character at a time. The basic process is quite simple, although we assume (without loss of generality) that we know which  $\kappa$ -mer is located at the beginning (the first position) of the target sequence.

The following is a very conservative algorithm which performs SBH with standard  $\kappa$ -mer probes. There are straightforward extensions that can be made to this algorithm which allow longer sequences to be reconstructed, but these improvements increase the maximum length of target sequences by only a factor of 2 or 3. We refer to this SBH method as ‘ungapped SBH’, to differentiate it from the gapped-probe algorithm which is the subject of this thesis. In the literature, it is commonly called ‘uniform,’ or ‘standard’ SBH.

### Ungapped Sequencing by Hybridization

1. Select the first  $\kappa$ -mer ( $p_0$ ) of the target sequence from the spectrum, and let the putative sequence be equal to  $p_0$ .
2. Search the spectrum for a probe (or probes) with a  $(\kappa - 1)$ -prefix matching the  $(\kappa - 1)$ -suffix of the putative sequence.



3. If there is exactly one such matching probe (denoted  $p_i$ ), *extend* the putative sequence by appending the  $\kappa^{\text{th}}$  character of  $p_i$ , and goto Step 2.
4. If there are more than one matching probes, the reconstruction fails.
5. If there are no matching probes, the reconstruction is complete.

■

If it is impossible to uniquely identify  $p_0$  in the spectrum, then *any* probe may be used as a starting point for sequencing. The only difference is that once the sequence has been extended to the *right* until the reconstruction either fails or completes, it must then be extended to the *left*. The process is virtually identical to rightward extension, except that in Step 2, the spectrum must be searched for probes with a  $(\kappa - 1)$ -*suffix* matching the  $(\kappa - 1)$ -*prefix* of the putative sequence and, in Step 3, the first character of the matching probe must be prepended to the putative sequence. A simple example of SBH using 5-mers (i.e.  $\kappa = 5$ ) to reconstruct a 23-character sequence follows:

**Example 1.6.** Let the target sequence  $s = \text{GGAGGCTATTATCGAATATCCCC}$ . The spectrum  $\mathcal{S}$  of the string  $s$  is  $\{ \text{AATAT}, \text{AGGCT}, \text{ATATC}, \text{ATATT}, \text{ATCCC}, \text{ATCGA}, \text{ATTAT}, \text{GAATA}, \text{CGAAT}, \text{CTATT}, \text{GAGGC}, \text{GCTAT}, \text{GGAGG}, \text{GGCTA}, \text{TATCC}, \text{TATCG}, \text{TATTA}, \text{TCCCC}, \text{TCGAA}, \text{TTATC} \}$ . We want to reconstruct the target sequence  $s$  from its spectrum.

First, we select probe  $p_0 = \text{GGAGG}$  as the starting point for sequencing (Step 0). Then we search the spectrum for a probe which has a 4-prefix matching the 4-suffix of the putative sequence, GAGG. There is a single matching probe, so  $p_1 = \text{GAGG}\text{C}$ . The character C is appended to the putative sequence, giving us the 6-character sequence GGAGGC.

Now Step 1 is repeated. Searching the spectrum for a probe which has a 4-prefix matching the 4-suffix AGGC yields the single response AGGC**T**, so the character T is appended to the putative sequence, resulting in GGAGGCT.

The next (third) extension step yields the probe GGCT**A** matching the 4-suffix GGCT, so A is appended to the sequence. Continuing in the same manner, the fourth extension step yields the probe GCTA**T** as the sole match to the 4-suffix GCTA, and so a T is appended to the sequence.

Each subsequent iteration is identical in process to the first: the spectrum is searched for a probe which matches the 4-suffix of the putative sequence, and (since this is an example of a successful reconstruction) there is a single matching probe. The remaining 4-suffixes, the probes with 4-prefixes matching the 4-suffix, and the character added to the putative sequence as a result of the successful search are shown in the rows of the following table. Recall that the putative sequence thus far is GGAGGCTAT.

4-suffix	Matching Probe	Appended Character
CTAT	CTATT	<b>T</b>
TATT	TATTA	<b>A</b>
ATTA	ATTAT	<b>T</b>
TTAT	TTATC	<b>C</b>
TATC	TATCG	<b>G</b>
ATCG	ATCGA	<b>A</b>
TCGA	TCGAA	<b>A</b>
CGAA	CGAAT	<b>T</b>
GAAT	GAATA	<b>A</b>
AATA	AATAT	<b>T</b>
ATAT	ATATC	<b>C</b>
TATC	TATCC	<b>C</b>
ATCC	ATCCC	<b>C</b>
TCCC	TCCCC	<b>C</b>
CCCC	None	-

Each character in the third column is appended, in turn, to the initial putative sequence (GGAGGCTAT). Thus, we arrive at the sequence GGAGGCTATT-**ATCGAATATCCCC**. The final search of the spectrum attempts to find a probe with a 4-prefix matching CCCC. There is no such probe in the spectrum, so the search returns 0 matching probes, and the reconstruction process terminates successfully. ■

The above example illustrated a successful sequencing attempt. It is possible to construct a sequence which cannot be as easily reconstructed by making only a single-character change to the target sequence  $s$ . If the 19<sup>th</sup> character of  $s$  is changed to T, the new target sequence  $s' = \text{GGAGGCTATTATCGAATAT}\text{T}\text{CCC}$  is produced. The spectrum of  $s'$  (with the changes highlighted in grey) is: { AATAT, AGGCT,

ATATC, ATCGA, ATTAT, ATTCC, GAATA, CGAAT, CTATT, GAGGC, GC-TAT, GGAGG, GGCTA, TATCG, TATTC, TATTA, TCGAA, TTATC, TTCCC }.

This spectrum will be denoted  $\mathcal{S}'$ .

Now, if sequencing again begins with probe  $p_0 = \text{GGAGG}$ , the reconstruction process proceeds as in the above example until the sequence has been extended to GGAGGCTATT. The next extension step searches  $\mathcal{S}'$  for probes with a 4-prefix matching TATT. There are two matching probes in  $\mathcal{S}'$ : TATTA and TATTC. There are two potential extensions to the putative sequence, and thus the reconstruction process fails.

We generally refer to the case where there is more than a single probe matching an extension query as an *ambiguous extension*. When an ambiguous extension occurs, it is often possible to determine which extension (or which branch) is the correct one by continuing to extend *both* possible sequences.

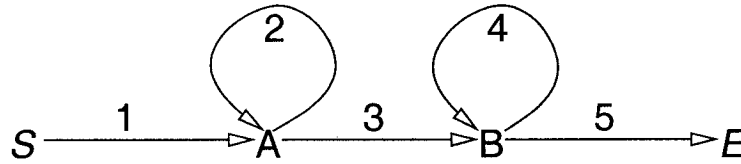


Figure 1.8: A graph with only one Eulerian path. ‘A’ and ‘B’ are points where an ambiguous extension occurs, but there is only one possible traversal of all 5 edges: 1-2-3-4-5. Thus, any target sequence which can be represented by a graph of this type can be unambiguously reconstructed.

Let’s return to the case above, where the target sequence  $s' = \text{GGAGGC TATT-ATCGAA TATT CCC}$ . The first ambiguous extension occurs at position 7 (the first grey box). If both sequences are extended, the *spurious* sequence (the branch beginning with the character C) will be extended by only three more characters before the end of the sequence is reached. Conversely, the *correct* sequence (the branch beginning with the character A) will be extended until position 17 (the lighter grey box), at which point there is another identical ambiguous extension. We can construct a graph  $G_s$  representing the sequences which are consistent with this process. There are vertices for the beginning of the sequence  $S$ , the end of sequencing  $E$ , and all ambiguous extensions. The edges are labeled with the sequence fragments produced

between vertices. The graph for  $s'$  is of the form shown in Figure 1.8; it is trivial to see that there is only one path from  $S$  to  $E$  which incorporates all of the edges (and thus sequence fragments) in the graph.

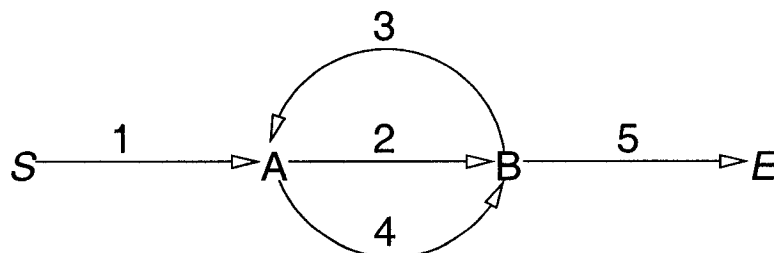


Figure 1.9: A graph with two Eulerian paths. ‘A’ and ‘B’ are points where an ambiguous extension occurs, and there are two valid traversals of all 5 edges: 1-2-3-4-5 and 1-4-3-2-5. Any target sequence which contains such a pair of interleaved repeats cannot be unambiguously reconstructed.

In general, if there is a single Eulerian path from  $S$  to  $E$  in  $G_s$  for a sequence, then we can unambiguously reconstruct the target sequence from its spectrum. However, there exist quite simple graphs for which there is no unambiguous Eulerian path. Let  $A$  and  $B$  denote two  $(\kappa - 1)$ -character strings. If  $A$  and  $B$  are each repeated within the target sequence, in the relative order  $ABAB$ , there is no unambiguous path through  $G_s$ , as shown in Figure 1.9. Pevzner showed in [P91] that there is no way of resolving these interleaved duplicate strings, and that they are common enough to restrict SBH to sequences of length  $O(2^\kappa)$ , which compares very poorly with the promised  $O(4^\kappa)$  information-theoretic limit.

While SBH initially seemed to be a promising technique, Pevzner’s work showed that it was limited by the probing schemes available at the time: the length of sequences which can be unambiguously reconstructed using 8-mers is only a few hundred bases—offering virtually no improvement over electrophoresis. Furthermore, increasing  $\kappa$  by 1 only doubles the length of feasible target sequences, at a cost of 4 times as much chip area. At the time, the benefits of SBH seemed to be outweighed by the problems it introduced, and it lost much of its original lustre.

## 1.5 Universal Bases and Gapped Probes

When the initial burst of interest in SBH waned, research in the area nearly stopped for several years. Then in 1999, Preparata, Frieze and Upfal [PFU99] presented a novel approach to SBH which increased the power of the method nearly a hundred-fold. The new technique makes use of wild-cards in the probing pattern, which can possibly be realized with artificial nucleotides known as *universal bases*. When initially discovered, they were thought to be nothing more than an annoyance to researchers, but they allow a powerful improvement to be made to the SBH method.

**Definition 1.3.** *A universal base is a nucleotide which forms a non-selective pairwise bond with any of the four natural bases.*

Two universal bases commonly in use today are 3-nitropyrrole 2'-deoxynucleoside and 5-nitroindole 2'-deoxynucleoside (5-nitroindole) [KB01]. These universal bases are not perfect, as they still exhibit some selectivity in the bonds formed with natural bases. An ideal universal base could be inserted into a single-stranded sequence of natural bases, and have the same hybridization strength with adenine, cytosine, guanine, or thymine.

Universal bases modify the rules of hybridization significantly. When considering whether or not two sequences will hybridize, universal bases 'match' any natural base. Usually, the character '\*' denotes a universal base. To improve the legibility of this document, probing patterns will be shown using 'N's and '.'s to represent natural and universal bases, respectively. Instances of probes will be displayed using the four natural bases, A, C, G and T in place of the N character, while '.' will still be used to represent the universal bases.

For example, the string ATG . TC represents a sequence consisting of three natural bases (ATG), followed by a universal base (.), followed by two more natural bases (TC). This segment will hybridize any of the four sequences TACAAG, TACCAG, TACGAG and TACTAG.

The model we adopt in this thesis assumes error-free hybridization between single-stranded DNA fragments. Reality, of course, is more subtle, allowing for a range of errors. Both *false positive* and *false negative* hybridizations may occur. False positives take place when two sequences which are not complementary hybridize anyway. This

results in an entry in the spectrum for a probe which is not present in the target sequence. False negatives occur when a probe fails to hybridize its complementary sequence. Some research has been done to perform SBH in the presence of errors [DH00, H+02], but we deal only with the noise-free case.

**Definition 1.4.** *Two single-stranded nucleotide sequences are complementary if and only if one of the following three conditions holds, for every position in the two aligned sequences:*

1. *The first strand contains a universal base.*
2. *The second strand contains a universal base.*
3. *Both strands contain natural bases, and the nucleotide in the first strand is the Watson-Crick complement of the nucleotide in the second strand.*

**Example 1.7.** The sequence ‘AT.TTAG’ is complementary to the sequence ‘TAG.A.C’. The two sequences align as follows:

Position	1	2	3	4	5	6	7
Sequence 1	A	T	.	T	T	A	G
Sequence 2	T	A	G	.	A	.	C

At each position which contains a natural base in both sequences, the nucleotide in the first sequence is Watson-Crick complementary to the nucleotide in the second. At position 3, the first sequence contains a universal base; the second sequence contains a universal base at positions 4 and 6. ■

When only the four natural bases (A, C, T and G) are used to create probes, the number of different probes on a microarray is determined solely by the length of the probes used; a chip which contains a complete family of  $\kappa$ -nucleotide probes must contain  $4^\kappa$  features. The addition of universal bases changes the calculation of chip size, since universal bases allow longer probes to be used without increasing the number of features on the DNA chip.

For example, consider a microarray containing the complete family of 4-nucleotide probes (consisting of  $4^4 = 256$  probes),  $\mathcal{C}_n = \{ \text{AAAA, AAAC, } \dots, \text{TTTT} \}$ . Two

universal bases may be inserted into the middle of each probe, creating a family of probes  $\mathcal{C}_u = \{ AA..AA, AA..AC, AA..AT, \dots, TT..TT \}$ . Note that this is still a complete family of probes; the addition of the two universal bases does not require that any additional probes be added to the microarray. In fact, both chips resolve four-character subsequences of a target sequence. In general, universal bases (and groups of adjacent universal bases) simply act as *gaps* between natural bases. Accordingly, Preparata & Upfal [PU00] make the following definition:

**Definition 1.5.** *Probes which contain universal bases are called gapped probes.*

In theory, universal bases allow microarrays to be constructed with arbitrarily long probes, since an unlimited number may be inserted into ungapped probes without increasing the chip size. Another ten universal bases may be inserted into the 6-base gapped probes above to create probes of the form  $NN.....NN$ . Although this probe has an overall length of 16 nucleotides, a total of 12 are universal bases, and so a microarray containing a complete family of probes of this form contains only  $4^4 = 256$  features.

Universal bases may be inserted into an ungapped probe at more than one location (creating gapped probes that contain more than one gap). For instance, probes of the form  $NN...N..N$  are perfectly valid. Note that probes always begin and terminate with a natural base, since adding universal bases at the end of an ungapped probe serves no useful purpose. In general, any configuration of natural and universal bases defines a *probing pattern*, as follows:

**Definition 1.6.** *A probing pattern (or probe pattern) beginning and ending with N refers to a particular configuration of natural and universal bases. It can be represented as a binary string, with 1's denoting natural bases and 0's universal bases. Probing patterns may be roughly described by two parameters: their length ( $\lambda$ ) and the number of natural bases they contain ( $\kappa$ ).*

**Example 1.8.** The binary probing pattern 1111 represents standard ungapped 4-nucleotide probes. ■

**Example 1.9.** The first gapped probes discussed above, created by inserting two universal bases into 4-nucleotide ungapped probes, are represented by the pattern 110011. ■

The positions in a probing pattern which contain natural bases are typically called the *specified positions*. This is in contrast with the *unspecified positions* occupied by the universal bases. In explanation, consider the gapped probe A . . T . . GAA, chosen at random from some spectrum. We know that the probe corresponds to a specific subsequence of the target sequence, but out of the 9 bases contained within the span of the probe, the specific identity of only  $\kappa = 5$  bases is known. These 5 positions are said to be *specified*; the remaining 4 are left *unspecified*—their identities are unknown.

Using ungapped  $\kappa$ -nucleotide probing patterns, each probe hybridizes a  $\kappa$ -base substring of the target. The location of each probe in the target string can be denoted by the left-most character in the subsequence to which it hybridizes. For example, the ungapped probe AGGC occurs at position 4 in the sequence TTAAGCGA. Note that two ungapped  $\kappa$ -probes which occur at adjacent positions in the sequence always have a  $\kappa - 1$  base overlap; the last three bases in A **GGC** are the same as the first three bases in the probe **GGC**A, which occurs at position 5 of the sequence.

The location of a  $\lambda$ -nucleotide gapped probe can be defined analogously to the location of an ungapped probe, except that gapped probes do not correspond to specific  $\lambda$ -substrings of the target, but rather  $\kappa$ -subsequences, where the  $\kappa$  natural bases need not necessarily be adjacent:

**Definition 1.7.** *A probe is said to occur at the  $i$ th position of a sequence if, when the left-most position of the probe is aligned with the  $i$ th position of the sequence, all of the natural bases in the probe match the corresponding bases in the aligned sequence.*

**Example 1.10.** Consider the DNA sequence ACCCAGTAGCGTAGA and the probing pattern 111001001. The probe defined by aligning the left-most position of the pattern with the first character of the sequence is ACC . . G . . G; we say that this probe occurs at position 1 of the sequence. The probe which occurs at position 6 of the sequence is defined by aligning the left-most position of the probing pattern with the sixth character of the sequence, GTA . . G . . G. ■

The binary string representing a probing pattern can be thought of as a sampling filter which is applied to a DNA sequence. With the left-most position of the pattern aligned to a particular position of the sequence, each position of the sequence which aligns with a 1 in the pattern is ‘sampled’ by the pattern. Positions in the sequence



which align with a 0 in the pattern are not sampled; they are effectively ignored. For example, consider the string  $S = \text{ATGTAAATA}$ , and the probing pattern  $P = 1101001$ . One possible alignment of the  $P$  with  $s$  is:

Position	1	2	3	4	5	6	7	8	9
Sequence	A	T	G	T	A	A	A	T	A
Probe Pattern		1	1	0	1	0	0	1	

In this alignment, the pattern ( $P$ ) samples 4 positions of the sequence: 2 (T), 3 (G), 5 (A) and 8 (T). The three other positions which are contained within the span of the probe are not sampled, since they align with universal bases in the probing pattern. Put another way, the characters sampled by a probing pattern aligned with the  $i$ th position of a sequence define a probe; the probe defined by the above alignment is  $\text{TG} \cdot \text{A} \cdot \cdot \text{T}$ . Formally, we define sampling as follows:

**Definition 1.8.** *A probe is said to sample a character ( $c$ ) in a sequence if one of the specified positions in the probe aligns with  $c$ .*

A probing pattern with  $\kappa$  natural bases samples  $\kappa$  of a sequence. Conversely, in a the spectrum  $\mathcal{S}$  of a target sequence, there are  $\kappa$  probes which sample every position (excluding the first and last  $\lambda - 1$  characters, which are sampled by fewer than  $\kappa$  probes).

**Example 1.11.** For example, consider the probing pattern 100100100111. If the left-most position of the probe is aligned with the  $i$ -th character of the sequence, the probe samples the six positions  $\{i, i + 3, i + 6, i + 9, i + 10, i + 11\}$ . When aligned with the third character in the following sequence, the pattern samples the italicized characters:  $\text{TTAAGCGGTGGAAG}$ . ■

Alignments of a probing pattern with a sequence are only valid if the probing sequence is completely contained within the span of the sequence. Consider again the same probing pattern and sequence. There are only three valid ways of aligning the probing pattern with the sequence: the left-most position of the pattern may be aligned with the first, second, or third character of the sequence, as follows:

Position	1	2	3	4	5	6	7	8	9
Sequence	A	T	G	T	A	A	A	T	A
	1	1	0	1	0	0	1		
		1	1	0	1	0	0	1	
			1	1	0	1	0	0	1

There are no further valid alignments of  $P$  with  $s$ , since all other potential alignments result in the pattern  $P$  extending one or more positions beyond the end (or beginning ) of the sequence  $s$ . There are always  $m - \lambda + 1$  valid alignments of a sequence of length  $m$  and a probe of length  $\lambda$ .

When a complete family of gapped probes are arrayed on a DNA chip, the spectrum resulting from hybridization with a target sequence does not reveal a complete set of  $\kappa$ -mers, since the  $\kappa$  natural bases in a gapped probe are not adjacent to one another. The spectrum resulting from such an experiment identifies instead the set of all *probes* ( $\kappa$ -subsequences) which hybridize with the target sequence. If there are no duplicates, the spectrum of a sequence of length  $m$  contains  $m - \lambda + 1$  unique probes.

**Definition 1.9.** *The spectrum of a target sequence ( $s$ ) of length  $m$  consists of every probe which can be generated by all possible alignments of the probing pattern with the sequence  $s$ . Given a probing pattern of length  $\lambda$ , there are  $m - \lambda + 1$  probes in the spectrum, assuming no duplication of probes.*

**Example 1.12.** The spectrum of the target sequence ATGTAAATA using the probing pattern 1101001 (or N.N..N) consists of the three probes which are generated by the three valid alignments of  $p$  with  $s$  shown above: AT.T..A, TG.A..T, and GT.A..A. ■

Now that we have described the nature of a gapped-probe spectrum, the algorithm which reconstructs a sequence from such a spectrum must be discussed.

## 1.6 SBH with Gapped Probes

The gapped-SBH algorithm developed by Preparata, Frieze & Upfal begins with a  $(\lambda - 1)$ -nucleotide segment of known DNA (called a *seed*), and then extends that

segment a single base at a time until the entire sequence has been reconstructed. There are two modes of extension, called SIMPLE and SUPER. The SUPER mode of extension is alternately referred to as the *branching* mode of extension, for reasons which will shortly become clear.

The increased power of gapped-probe SBH is derived from one simple property. Two adjacent ungapped  $\kappa$ -mers always share a  $(\kappa - 1)$ -length subsequence; the  $(\kappa - 1)$ -suffix of the probe at position  $i$  is identical to the  $(\kappa - 1)$ -prefix of the probe at  $i + 1$ . Adjacent gapped probes, on the other hand, may share fewer than  $\kappa - 1$  nucleotides. In fact, two adjacent gapped probes need not share *any* common subsequences. Two adjacent probes (located at positions  $i$  and  $i + 1$ ) with the pattern  $P = \text{NN} \cdot \text{N} \cdot \cdot \text{N}$  have only a single base in common. For instance, consider the probes of this form occurring at positions 1 and 2 in the sequence  $S = \text{ATGTAATA}$ :  $p_1 = \text{AT} \cdot \text{T} \cdot \cdot \text{A}$  and  $p_2 = \text{TG} \cdot \text{A} \cdot \cdot \text{T}$ . The only position in the sequence which is sampled by both probes is the second character (T) in the sequence  $S$ ; it occurs at both the second position in  $p_1$  and the first position in  $p_2$ . Furthermore, two adjacent probes of the pattern  $\text{N} \cdot \cdot \text{N} \cdot \cdot \text{N} \cdot \cdot \text{N}$  have absolutely no characters in common.

**Definition 1.10.** *An  $(s, r)$ -probing pattern is a particular family of probing patterns. They can be either direct or reverse patterns. A direct  $(s, r)$ -pattern has the form  $1^s(0^{s-1}1)^r$ , while a reverse pattern has the form  $(10^{s-1})^r1^s$ . They have length  $\lambda = s(r + 1)$  and  $\kappa = s + r$  natural bases.*

It is interesting to note that the traditional ungapped probe is just a special case of an  $(s, r)$ -probing pattern. Both  $(\kappa, 0)$  and  $(1, \kappa - 1)$  probes resolve to a solid probe of  $\kappa$  natural bases, with no fooling bases interjected. Here are two more examples of gapped probing patterns, both of which are somewhat more useful.

**Example 1.13.** A direct  $(4, 4)$ -probing pattern has the form 11110001000100010001, or  $\text{NNNN} \cdot \cdot \cdot \text{N} \cdot \cdot \cdot \text{N} \cdot \cdot \cdot \text{N} \cdot \cdot \cdot \text{N}$ . The corresponding reverse pattern has the form 10001000100010001111, or  $\text{N} \cdot \cdot \cdot \text{N} \cdot \cdot \cdot \text{N} \cdot \cdot \cdot \text{N} \cdot \cdot \cdot \text{NNNN}$ . ■

**Example 1.14.** The spectrum  $\mathcal{S}$  of the sequence  $\text{ATAGCGATAGCGA}$  using a reverse  $(3, 2)$ -probe  $(\text{N} \cdot \cdot \text{N} \cdot \cdot \text{NNN})$  pattern contains the following probes, listed in the same order in which they are found in the sequence, from left to right:  $\{ \text{A} \cdot \cdot \text{G} \cdot \cdot \text{ATA}, \text{T} \cdot \cdot \text{C} \cdot \cdot \text{TAG}, \text{A} \cdot \cdot \text{G} \cdot \cdot \text{AGC}, \text{G} \cdot \cdot \text{A} \cdot \cdot \text{GCG}, \text{C} \cdot \cdot \text{T} \cdot \cdot \text{CGA} \}$ . The

sequence has length 13, the probing pattern is 9 characters long, and there are no duplicate probes, so the spectrum contains  $m - \lambda + 1 = 13 - 9 + 1 = 5$  distinct probes.

■

As mentioned before, we assume that the sequencing process begins with a short known sequence of DNA—a seed, or primer string. This seed sequence needs merely to have length at least  $(\lambda - 1)$  and be an actual substring of the target sequence. The seed can occur at any position in the target DNA sequence. The gapped-probe SBH algorithm reconstructs the complete target sequence from the seed and the spectrum, by extending the putative sequence character by character, beginning with the seed. Note that when using ungapped probes, *any* probe meets the requirements for a seed, since  $\lambda = \kappa$ , and each probe *is* a fully specified substring of the target sequence.

The simple mode of extension using gapped probes is nearly identical to the extension mode for ungapped probes. The primary difference between the gapped and ungapped modes of operation for the sequencing algorithm occurs when there are two or more probes which provide valid extensions to the current putative sequence. When using ungapped probes, such an ambiguous extension can only be resolved by an Eulerian path solving algorithm.

However, gapped probes allow an initially ambiguous extension to be resolved quite simple. The first component of the disambiguation process requires the definition of the *spectrum query* operation. Recall that Definition 1.4 describes complementary sequences containing universal bases.

**Definition 1.11.** *The spectrum query is the elementary operation of SBH. It produces the set of all probes in the spectrum that match a  $\lambda$ -character query string  $q$ . A probe  $p$  matches a query string  $q$  if, for  $i = 1.. \lambda$ ,  $p_i$  is complementary to  $q_i$ .*

Of course, a universal base will hybridize any natural base, so any position which contains a universal base in *either*  $p$  or  $q$  automatically fits the criteria. The only positions which can cause a mismatch between a query and a probe are those which contain natural bases in *both*  $p$  and  $q$ .

**Example 1.15.** A spectrum constructed with a (3,2)-probe reverse pattern (N...N...NNN) contains the probes A...A...CTC, A...G...TAC, A...T...TTA, C...T...TAC, G...G...GTA, T...A...AGG. It is queried with the string

‘AGG.A.T..’ The following table helps illustrate how the characters in the query string align with the probing pattern:

Position	1	2	3	4	5	6	7	8	9
Pattern	N	.	.	N	.	.	N	N	N
Query	A	G	G	.	A	.	T	.	.

Only positions 1 and 7 contain natural bases in both the query string and the probing pattern; all other positions have at least one universal base. Thus, any probe in the spectrum which contains the natural base A in position 1 and T in position 7 is a match to the query string. The resulting set contains the two probes  $\overline{\text{A}} \dots \text{G} \dots \overline{\text{T}} \text{AC}$  and  $\overline{\text{A}} \dots \text{T} \dots \overline{\text{T}} \text{TA}$ . ■

All of the probes in a spectrum share the same pattern, so when a  $\lambda$ -character query  $q$  is used to probe the spectrum, only the  $\kappa$  positions in  $q$  which contain natural bases can affect the search results. If a  $q$  contains natural bases in each of the specified positions in the pattern, then the spectrum query merely determines whether or not the exact probe which matches the query is present in the spectrum.

Every position in  $q$  which contains a universal base that aligns with a natural base in the probing pattern is called a *free position*, and increases the number of potentially matching probes in the spectrum by a factor of 4. A query which contains one free position matches at most 4 probes in the spectrum; a query which contains  $n$  free positions may produce as many as  $4^n$  matching probes. Of course, there can be at most  $\kappa$  free positions in a query string; if  $n = \kappa$ , then the query string will match every probe in the spectrum. The example above was a query with 3 (out of a maximum 5) free positions. From here on, we will use the character ‘?’ to signify the free positions in a query, to differentiate them from the other universal bases in the probing pattern.

Step 2 in the sequence-extension process for ungapped probes, wherein the spectrum is searched for probes with a  $(\kappa - 1)$ -prefix matching the  $(\kappa - 1)$ -suffix of the putative sequence, is equivalent to constructing a  $\kappa$ -character query string consisting of the  $(\kappa - 1)$ -suffix of the putative sequence with a single universal base appended. Such a query string always has the form  $N^{\kappa-1}?$ , and matches probes which contain the appropriate  $(\kappa - 1)$ -prefix.

Similar query strings are used in gapped-SBH sequence extension. They are constructed from the  $(\lambda - 1)$ -suffix of the putative sequence, with a single universal base appended. They have the form  $N^{\lambda-1}?$ . Since the last (right-most) position in a probing pattern is always a natural base, this implies that they always have a single free position; and  $(\kappa - 1)$  specified and non-free positions which align with the natural bases in the probing pattern. (The right-most of the  $\kappa$  specified position is free.) Thus there are at most 4 probes in the spectrum which match any extension-query.

**Example 1.16.** Using (3,2)-direct probes (NNN..N..N), which have  $\lambda = 9$  and  $\kappa = 5$ , the query strings consist of the 8-suffix of the putative sequence, with a universal base appended. If the putative sequence is AGTCGTGAGC, the query string is ‘TCGTGAGC?’. The query strings align with the probes as follows:

Position	1	2	3	4	5	6	7	8	9
Pattern	N	N	N	.	.	N	.	.	N
Query	N	N	N	N	N	N	N	N	?

A matching probe in the spectrum must contain the correct natural base in positions 1, 2, 3 and 6. ■

Now that spectrum queries have been defined, we move on to the process for gapped-probe sequencing. We assume here that the seed is the first  $\lambda - 1$  characters of the target sequence, although this need not be the case; a seed which occurs elsewhere in the sequence can be extended in both directions until the reconstruction process has completed. This process is functionally identical to the ungapped sequencing process, except that if there are two or more probes matching a query the *branching* mode of extension is initiated, whereas the ungapped sequencing algorithm simply fails.

### Gapped Sequencing by Hybridization

1. Let the putative sequence be equal to the  $(\lambda - 1)$ -character seed.
2. Query the spectrum with a query string constructed from the  $(\lambda - 1)$ -suffix of the putative sequence with a universal base appended.

3. If there is exactly one probe (denoted  $p_1$ ) matching the query, append the  $\lambda^{\text{th}}$  character of  $p_1$  to the putative sequence, and repeat Step 2.
4. If there are two or more matching probes, initiate the *branching* extend process.
5. If there are no matching probes, the reconstruction is complete.

■

If no ambiguous extensions are encountered during the sequencing process, then the gapped-probe algorithm works the same as the ungapped version. The gaps in the probing pattern have no effect on the probability of finding an ambiguous extension. However, with gapped probes, it is possible to resolve ambiguous extensions by means of the other probes which sample the ambiguous position: this is the *branching* extend process.

The branching extend process is simple: just construct a separate sequence for each of the possible extension characters, and continue extending each of these sequences breadth-first. This is somewhat similar to the Eulerian path resolution which can be performed for ungapped SBH, except that a spurious sequence is typically eliminated very quickly.

Using an ungapped  $\kappa$ -mer probing pattern, there are  $\kappa$  probes which sample each position in the target sequence. This is also true when using a gapped probing pattern with  $\kappa$  natural (specified) positions. However, when using gapped probes, the set of sampling probes have a high degree of independence. This is not the case for ungapped probes.

Consider once again the sequence GGAGGC **TATTA** TCGAA **TATTC** CC. An ambiguous extension occurs when the spectrum is queried with the string TATT?, since there are two matches in the spectrum. Note that the two matching probes correspond to two matches in the sequence. All of the  $(\kappa - 1)$  characters in each probe to the left of the ambiguous character are identical. The next probe that samples the ambiguous character just samples a shorter  $(\kappa - 2)$  substring to the left of it. The next probe that samples the correct extension (A) is ATT **A** T, and the next probe sampling the incorrect extension character (C) is ATT **C** C. Note that both of these match exactly in all positions to the left of the spurious character.

Now consider the spectrum  $\mathcal{S}$  of (3,2)-direct probes (NNN..N..N) of the sequence CGG **TAT** CC **T** AG **A** TC TAT GT T TT **C** CCAAG. When the spectrum is queried with the string ‘TATCCTAG?’, there are again two matches in the spectrum: TAT..T..**A** and TAT..T..**C**. However, the underlying sequences at the locations of each probe are not the same (TATCCTAGA  $\neq$  TATGTTTTC). The next probe that samples the ambiguous character samples an almost completely different set of characters to the left of the ambiguous position.

In fact, the next probe that samples the correct extension character (A) is CCT..**A**..T, whereas the next probe sampling the incorrect extension character is GTT..**C**..A. These two probes have only one character to the left of the ambiguous character in common, and this is what gives gapped probes an increased power of resolution over ungapped probes.

Observe that when the initial ambiguous extension is encountered, the putative sequence is  $s_p = \text{GGGTATCCTAG}$ . The two potential sequences which are created for the two potential extension characters are GGGTATCCTAG **A** and GGGTATCCTAG **C**. But while the next probe (CCT..**A**..T) that samples the ambiguous position in the correct path is (guaranteed to be) present in the spectrum, the next probe (CCT..**C**..T) that samples the ambiguous position in the spurious path is not present. There are no matches to the extension query for the spurious path, so we know that the *correct* path is the one which can be extended.

A major failure mode in gapped-probe sequencing requires that  $\kappa$  independent probes be present in the spectrum, each of which is a  $\kappa$ -character sequence, compared with a single probe in ungapped SBH. This is the essential reason why gapped SBH achieves performance of about 1/3 of the information theoretic bound. The next chapter explores this in far more detail, and carefully describes the events which can lead to sequencing failures. To conclude, we present a comparison of the performance of ungapped SBH and gapped SBH in Figure 1.10.



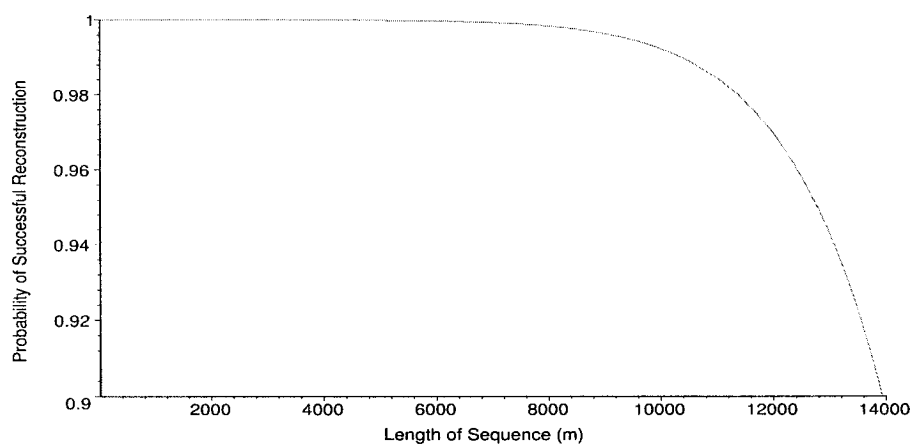


Figure 1.10: The gapped-probe method of SBH using (4,4)-probes can reconstruct target sequences nearly two orders of magnitude longer than the standard method using ungapped 8-mers. The gapped-SBH algorithm is represented by the gray curve. The ungapped curve is visible only as a slight thickening of the vertical axis of the plot.

## Chapter 2

# Implementation of the Gapped-SBH Algorithm

The basic gapped SBH algorithm was developed and described by Preparata and Upfal in [PU00]; the pseudo-code presented in this chapter implements a modified version of that algorithm. The focus of this chapter is not to produce an alternate method of reconstructing a sequence, but to describe an efficient implementation of their algorithm. Careful design yields an algorithm which runs in seconds instead of hours, and can be effectively used on much longer sequences.

### 2.1 C++ Pseudo-code Usage

The research described in this dissertation was conducted with a software simulation implemented in C++ using the STL (Standard Template Library). The pseudo-code snippets contained in this chapter describe both the data structures and algorithms used to construct the full software system. C++ allows variables to be grouped together in complex data structures—objects—which can also contain *member functions*. Objects with member functions can be treated as standard complex data structures: if a data structure object  $D$  contains a string  $s$ , an integer  $n$  and a list of integers  $i\text{-list}$ , they can be individually denoted as  $D.s$ ,  $D.n$ , and  $D.i\text{-list}$ . Member functions work the same as traditional procedures, except that a member function has access to all of the variables contained within the same object, as well as any

parameters it accepts. For instance, the procedure  $\text{SIZE}(D.i\text{-list})$  might return the number of elements contained in  $i\text{-list}$ . Conversely, if object  $D$  contains the member function  $\text{LISTSIZE}$ , then a call to  $D.\text{LISTSIZE}()$  would produce the same result. While this example is trivial, member functions in more complex data structures save us from having to pass long lists of parameters, and increase the legibility of both the pseudo-code and the C++ code itself.

The pseudo-code in this thesis follows a simplified C++ form, and has the following form for objects and data structures:

DATASTRUCTURE

```

variable LocalVar1 varType
variable LocalVar2
function MEMBERFUNCTION1(Arg1, Arg2)
function MEMBERFUNCTION2(Arg1)

```

In the object definition above, *LocalVar1* is a variable of type ‘varType’, which might be an integer, string, array, or another data structure. The second variable *LocalVar2* does not have a specified type; the type of such variables will be obvious from the context in which they appear. The two member functions (MEMBERFUNCTION1 and MEMBERFUNCTION2) accept two and one parameters, respectively. All member functions will have full pseudo-code provided separate from the object definition. When referring to an instance of DATASTRUCTURE called  $d$ , the member variables and functions will be denoted  $d.\text{LocalVar1}$  and  $d.\text{MEMBERFUNCTION1}(a, b)$ , respectively. The pseudo-code for functions (including the member functions of a data structure) will be written as:

MEMBERFUNCTION1(*Arg1*, *Arg2*)

```

1  if Arg1.Bool1 = true
2    then LocalVar2  $\leftarrow$  Arg2 + 1
3    else LocalVar2  $\leftarrow$  Arg1.FUNCTION1(Arg2 - 1)
4  return LocalVar2  $\times$  2

```

This pseudo-code describes an algorithm which takes two arguments: *Arg1* and *Arg2*. However, since `MEMBERFUNCTION1` is a member of the object defined above, it has access to the two local member variables in the data structure, *LocalVar1* and *LocalVar2*, in addition to the two parameters it accepts. Data types of the parameters accepted by functions will be explained in cases where not obvious. Here, *Arg1* is a complex data structure containing (at least) a boolean local variable named *Bool1* and a member function `FUNCTION1` which accepts a single integer as a parameter. `MEMBERFUNCTION1` performs two simple tasks: first, it sets the value of *LocalVar2* on line 2 or 3 depending on the value of *Arg1.Bool1*. It then returns an integer equal to twice *LocalVar2* (line 4).

The standard notation that will be used for strings throughout this document is defined as:

**Definition 2.1.** A string  $S$  has  $\ell$  characters numbered  $1 \dots \ell$ . The  $i$ th character of a string will be denoted as  $S_i$ . A substring of  $S$  with length  $(\kappa - j + 1)$  characters beginning with the  $j$ th and ending with the  $k$ th will be denoted  $S_{j\dots k}$ . The length of the string is denoted  $|S|$ .

**Example 2.1.** Consider the string  $S = A \dots CAGG \dots AA.T$ . The length  $\ell$  of the string  $|S|$  is 15 characters. The 5th character of the string,  $S_5$ , is C and the 14th character of the string  $S_{14}$  is ‘.’. The substring of  $S$  denoted by  $S_{6\dots 12}$  is the 7-character string  $AGG \dots A$ . ■

The ‘+’ operator will indicate the concatenation of two strings. The operation  $a = b + c$  thus produces a string  $a$  with length  $|a| = |b| + |c|$ , consisting of the characters in the string  $b$  followed immediately by the characters in the string  $c$ .

Arrays are defined similarly to strings, but have a slightly different notation:

**Definition 2.2.** An array  $A$  contains  $m$  elements numbered  $1 \dots m$ . The  $i$ th element of an array is denoted  $A[i]$ , and a range of  $(k - j) + 1$  consecutive members from the  $j$ th to the  $k$ th inclusive is denoted  $A[j \dots k]$ . The number of elements in (the size of) the array may be accessed by using the function  $\text{SIZE}(A)$ , which returns the integer value  $m$ .

Now that notation has been dealt with, Section 2.2 introduces the algorithm itself.

## 2.2 Overview of the Sequencing Algorithm

The SBH algorithm is designed to completely and unambiguously reconstruct a sequence of DNA (called the *target sequence*) from its spectrum. It begins with a short segment of known DNA (called the *seed*), such as the PCR primer. The algorithm then proceeds to *extend* the seed one character at a time until either the entire sequence has been reconstructed, or the algorithm fails.

The process of sequence reconstruction is conducted by a simple algorithm called EXTEND, which attempts to add a single character at a time to the putative sequence. The EXTEND algorithm, in turn, calls the QUERY function, which interrogates the spectrum to determine which probes match the  $(\lambda - 1)$ -character suffix of the current string.<sup>1</sup> Finally, the CHECK algorithm—which is used by QUERY to verify whether one specific probe is present in the spectrum—is the elementary operation of SBH; the complexity of the SBH algorithm overall will be analyzed in terms of the number of *spectrum checks* performed.

It follows that the data structure used to represent the spectrum is the single most important factor affecting the execution time of the algorithm. A spectrum check is nothing more than a search for a particular probe in the spectrum, so the complexity of the whole SBH method depends on an efficient spectrum object. In turn, the data structure used to represent probes can guide the design of the spectrum, so the probe data structure is developed in the next section.

### 2.2.1 Probe Structure

Probes are most obviously represented as fixed-length ( $\lambda$ -character) strings over a 5-character alphabet,  $\{A, C, G, T, *\}$ . If we excise the universal bases from the probes, we can reduce the probes to  $\kappa$ -character strings over the 4-character DNA alphabet. The process of producing such a *packed* probe reduces the length of the probe from  $\lambda$  to  $\kappa$  characters; a reduction of over 50% in length for all  $(s, r)$  gapped probing patterns of interest. Since the probing pattern is constant for all of the probes in a spectrum, a packed string (in the form of an integer) unambiguously represents a single probe. Thus, it is not necessary to store more than one copy of the pattern in

---

<sup>1</sup>Recall from Chapter 1 that  $\lambda$  is the length of the probing pattern used to generate the spectrum.

the spectrum.

**Example 2.2.** The direct (4,4)-probe ATGG . . . A . . . T . . . G . . . C is *packed* into the  $\kappa$ -character string ATGGATGC by removing all of the universal bases. The probe requires a 20-character string in its unpacked form, and only 8 characters when packed. ■

There is another advantage of using the packed-string representation of probes. Strings are typically represented internally as arrays of the internal *character* data type; a 1-byte variable type. By reducing the alphabet to only 4 characters, strings can be represented with only 2 bits per base using a simple binary coding. This can also be viewed as an integer from 0 to 3, according to the following table:

character	A	C	G	T
binary	00	01	10	11
integer	0	1	2	3

This binary coding of bases suggests a simple translation from  $\kappa$ -character string to a  $(2 \cdot \kappa)$ -bit numerical representation of the probe. By converting the character representing each base to its 2-bit binary encoding and concatenating the binary codes in sequence, any probe with  $\kappa \leq 16$  natural bases may be converted to a single 32-bit integer.

**Example 2.3.** For example, the (4,3)-direct probe AACG . . . T . . . A . . . G is first packed into the string AACGTAG. The binary conversion of the 7-character string AACGTAG yields the 14-bit sequence 00 00 01 10 11 00 10, corresponding to the decimal number 434. ■

The complete unpacked string representation of a probe is easily recovered from its decimal representation. First, the decimal number is converted back to a binary notation, and padded with leading 0's to have the correct length. The binary sequence can then be trivially decoded into the packed-string representation of the probe, which is then unpacked into the full  $\lambda$ -character string representation.

**Example 2.4.** Consider the integer 1159, the decimal representation of a (3,4)-reverse probe. To convert it back to its full string representation, we first convert it to the

binary number 10010000111, and then rewrite it as 7 pairs of bits: 00 01 00 10 00 01 11. The binary string can then be easily decoded as ACAGACT by referencing the above table. Finally, universal bases are inserted at the appropriate positions to convert the packed string to the full (3,4)-reverse probe A . . C . . A . . G . . ACT. ■

The three potential representations for probes—unpacked string, packed string, and decimal number—all contain the same information: namely, the identity and order of the natural bases in a gapped probe. Two of the representations are most convenient: the *unpacked string*, because it can be easily read by humans; and the *decimal number*, because it is the most compact way of storing and manipulating the probe internally to the program. No definitions of either structure are really required: the decimal form of a probe can be held in an integer, and the string form can be stored as a string, both of which are standard C++ data types. However, we do need a way of converting between the two.

The PROBEPATTERN object was created to enable simple translation between the two forms of probe representation (*string* and *decimal*). The object contains two functions: PACK, which translates from an unpacked string to a decimal probe, and UNPACK, which translates decimal probes back into unpacked strings. The object also contains an internal representation of the probe pattern itself, so that the UNPACK function always produces a probe of the correct form. The exact structure of the stored pattern is not important: it may be easily stored as a string (i. e. NNN . . N . . N . . N for a direct (3,3)-probe), or an array of numerical indices identifying the positions of the natural bases within the probe (i. e. {1, 2, 3, 6, 9, 12} for same (3,3)-pattern. The full definitions for the object and its two member functions follow:

PROBEPATTERN

```
variable Pattern
function PACK(ProbeString)
function UNPACK(ProbeDecimal)
```

By translating directly from the characters representing natural bases to integer values, we can skip a step in the conversion from a *string* probe to *decimal* probe.

This works as follows: let the integer translation of each natural base be denoted  $Integer(c)$ , and the reverse translation from integer to character notation be denoted  $Character(n)$ . Each character in the string from  $S_1 \dots S_\ell$  contributes an additive factor of  $4^i \cdot Integer(S_i)$  to the decimal representation of the probe.

```

PROBEPATTERN.PACK( $S$ )
1   $D \leftarrow 0$ 
2   $length = |S|$ 
3  for  $i \leftarrow 1$  to  $length$ 
4      do if  $S_i \neq UniversalBase$ 
5          then  $D \leftarrow D + Integer(S_i) \cdot 2^i$ 
6  return  $D$ 

```

It is often useful to pack a string containing no universal bases as though it were a standard probe. For example, the string AGGTTAGAGGGT can be treated as though it were the (3,3)-direct probe AGG . . A . . G . . T. Algorithmically, this is accomplished by refining the PACK function, so that it only considers positions in the *ProbeString* which correspond with natural bases in the probing pattern.

Note that the *Pattern* member variable of the PROBEPATTERN object must contain the positions of the  $\kappa$  natural bases within the probing pattern, as well as the length of the pattern. In the following functions, the positions of the natural bases from left to right are denoted as  $Pattern[1] \dots Pattern[\kappa]$ .

```

PROBEPATTERN.PACK( $S$ )
1   $D \leftarrow 0$ 
2  for  $i \leftarrow 1$  to  $k$ 
3      do  $j \leftarrow Pattern[i]$ 
4           $D \leftarrow D + Integer(S_j) \cdot 2^i$ 
5  return  $D$ 

```

Now that we have a way of easily computing the decimal representation of a probe from a string, we must define the UNPACK function, which translates a decimal probe



back into human-readable string form. The UNPACK function always returns a well-formed probe matching the PROBEPATTERN; the positions in the pattern containing universal bases will never contain any natural base. Recall that the character decoding of the integer representation of a natural base is denoted  $Character(n)$ .

PROBEPATTERN.UNPACK( $D$ )

```

1   $S \leftarrow$  string of  $\lambda$  universal bases
2  for  $i \leftarrow k$  to 1
3      do  $p \leftarrow Pattern[i]$ 
4           $d \leftarrow D \bmod 4$ 
5           $S[p] = Character(d)$ 
6           $D \leftarrow D/4$ 
7  return  $S$ 
```

Both the PACK and UNPACK algorithms require  $\kappa$  multiplication or division operations to convert a probe from one representation to another. They execute in  $O(\kappa)$  time, which remains constant for all microarrays of a particular size. Since  $\kappa$  is biochemically limited to values of 12 or less, the time required to convert from one form of probe to another can be considered to be constant. Nevertheless, the rest of the SBH algorithm should be designed in such a way as to restrict the number of conversions performed.

Now that we have defined the data structures used for probes, we can move on to the spectrum.

### 2.2.2 Spectrum Structure

The SPECTRUM of a target sequence  $S$  contains all of the probes which successfully hybridized with  $S$ . What data structure should be used to store them? We should be able to check whether or not a particular probe is present in the spectrum in  $O(1)$  time. We would also like to add probes to the spectrum in constant time, but since each probe is added to the spectrum only once, and may be accessed multiple times, the time taken to add a probe is not as important as the access time. Furthermore, no additional probes will be added to or removed from the spectrum after it is created,

so the number of probes in a spectrum remains constant.

For convenience, let  $n$  be the number of elements stored in a particular data structure. We can eliminate lists ( $O(n)$  access time) and trees ( $O(\log n)$  access time) from consideration since they have non-constant access times for arbitrary elements. Queues and stacks are more specialized containers which are not suited for our purposes: the order in which probes are added to the spectrum is irrelevant, and probes are *never* removed. Without bringing into consideration more obscure data structures, this leaves only hash tables and fixed-size arrays as potential spectrum containers.

Hash tables are extremely useful data structures. They have amortized  $O(1)$  access, insertion and deletion time, and they can be dynamically resized quite easily. On the other hand, a fixed-size array is the simplest possible data structure we might use to represent a spectrum. Since  $\kappa$  is always very small, it is feasible to use such an array. Rather than an amortized  $O(1)$  access time over all elements, it offers constant-time random-access addressing of any member. And although it cannot be expanded or contracted easily, spectra never change in size after construction.

A fixed-size array offers an additional advantage over a hash table. The decimal representation of a probe can be used as an array index, so only a single bit is required to indicate the presence or absence of each probe in the array. If the  $i$ th bit in the array is set, it indicates that the decimal probe  $i$  is present. Thus, although an array of  $4^\kappa$  elements is required to hold any spectrum for a probing pattern with  $\kappa$  natural bases, each element in the array is only a single bit. Although microarrays with over 1 million features can now be constructed, the required 1 Mbits is a trivial amount of memory in a modern computer. It is far more likely that biochemical, not computational constraints will prove to be the limiting factor in the size of the spectrum.

Most of the simulations we ran were conducted using probing patterns with  $\kappa = 8$ , requiring only  $4^8 = 65536$  *bits*, or 8 KBytes of space. Increasing  $\kappa$  to 12 still requires arrays only  $4^{12} = 2^{24} \approx 16$  million bits (2 MBytes), less than half a percent of the available memory on present-day commodity workstations.

**Example 2.5.** A sample spectrum for a (4,2)-reverse probing pattern ( $\kappa = 6$ ) is shown in the following table:

Array Index	0	1	2	3	4	5	6	7	...	4092	4093	4094	4095
Probe Present?	0	1	0	0	0	1	0	0	...	0	0	1	0

The three probes present are indicated by the ‘1’ entries in the second row. The three entries shown (decimal values 1, 5, and 4095) correspond to the three probes C...A...AAAA, C...C...AAAA, and G...T...TTTT. This array requires only 512 bytes of space to store. ■

In practice, arrays of bits were much slower to manipulate than arrays of the built-in C++ boolean data type. In C++, booleans require between 1 and 4 bytes of space, depending on the platform, but the additional memory required to store the spectrum is not significant for probes with  $\kappa < 12$ , and in exchange for using extra space, the program runs nearly 4 times as fast.

In addition to an array of booleans, the SPECTRUM object must hold a few additional members. Each SPECTRUM includes a PROBE PATTERN object which holds the spectrum’s associated probing pattern. A method to interrogate the spectrum is also required, so the CHECK function is added. CHECK takes only one parameter—the decimal representation of a probe—and returns *true* if the probe is present in the spectrum and *false* otherwise. Another function—CHECK-S—is also included, which accepts a string representation of a probe, but otherwise behaves identically to the CHECK function.

Finally, for purposes of simulation, we need a way to construct the spectrum of a target sequence—a software simulation of the biochemical step in the SBH process. This is implemented in the member function BUILD, which accepts a string and constructs the corresponding spectrum. The basic SPECTRUM object is defined as follows:

SPECTRUM

```

variable ProbePattern
variable ProbeArray[1...4 $\kappa$ ]
function BUILD(TargetSequence)
function CHECK(DecimalProbe)
function CHECK-S(StringProbe)

```

The SPECTRUM.BUILD function accepts only one parameter: the target sequence  $S$ . It then constructs the spectrum of  $S$  by adding all of the probes which match the target sequence to the spectrum. The pseudo-code is:

```
SPECTRUM.BUILD( $S$ )
1  for  $i \leftarrow 1$  to  $(|S| - \lambda)$ 
2      do  $p \leftarrow S_{i..i+\lambda}$ 
3           $d \leftarrow \text{ProbePattern.PACK}(p)$ 
4           $\text{ProbeArray}[d] \leftarrow \text{true}$ 
```

The pseudo-code for SPECTRUM.CHECK is very simple:

```
SPECTRUM.CHECK( $p$ )
1  if  $\text{ProbeArray}[p] = 1$ 
2      then return true
3  else return false
```

The code for the SPECTRUM.CHECK-S function requires only a minor modification to the code for CHECK:

```
SPECTRUM.CHECK-S( $p$ )
1   $d \leftarrow P.\text{PACK}(p)$ 
2  if  $\text{ProbeArray}[p] = 1$ 
3      then return true
4  else return false
```

The SPECTRUM.CHECK algorithm executes a single array index operation, requiring  $O(1)$ —with a very low constant—time to execute. The SPECTRUM.CHECK-S algorithm also includes a call to the PACK function, which executes  $O(\kappa)$  time; increasing the execution time for the CHECK-S function merely by a larger constant. (CHECK-S is really just a utility function which can be used in place of separate calls to PACK and CHECK.) Now that the basic data structures used by the program have

been defined, we can move on to the algorithms themselves.

## 2.3 The EXTEND Algorithm

The SBH reconstruction process works by making successive calls to EXTEND, each time adding a short segment of one or more characters to the current sequence. The process starts with a short segment of known DNA—a primer for the reconstruction process—called a *seed*. The seed segment can occur at any position in the target sequence, but must be at least  $\lambda - 1$  characters in length. At any point during sequencing, the portion of the sequence which has been reconstructed is called the *putative sequence*. Beginning with the seed, the EXTEND algorithm searches the spectrum for probes with a  $(\lambda - 1)$ -prefix matching<sup>2</sup> the  $(\lambda - 1)$ -suffix of the putative sequence. If only a single match is found in the spectrum, the matching character is appended to the end of the current putative sequence, and the reconstruction process continues. If more than one matching probe is found in the spectrum, EXTEND attempts to resolve the ambiguous extension by continuing *all* of the possible sequence extensions until only one remains.

The spectrum-search process is implemented in a separate function named QUERY, which accepts two parameters: the spectrum ( $\mathcal{S}$ ) and a  $\lambda$ -character query string ( $q$ ) containing a single wild-card character (denoted with the ‘?’ character in the string  $q$ ). The location of the wild-card character in the string  $q$  is called the *free position* in the query string.

Let’s examine how the QUERY function is used by the EXTEND algorithm. EXTEND first aims to find a single-character extension to the putative sequence. This can be done by querying the spectrum with a  $\lambda$ -character string composed of the  $(\lambda - 1)$ -suffix of the putative sequence ( $s$ ) with a wild-card character (the character ‘?’) appended, or

$$q = s_{|s|-(\lambda-1) \dots |s|} + ?$$

The right-most character of any probe in the spectrum which matches the query is a potential extension to the putative sequence. Consider the following example:

---

<sup>2</sup> *Matching* was defined in Chapter 1.

**Example 2.6.** The sequence reconstructed so far from a  $(3,2)$ -reverse spectrum is  $s = \text{AAGCGCATAGTAGAT}$ . For  $(3,2)$ -probes,  $\kappa = 5$  and  $\lambda = 9$ . The query string constructed from the 8-suffix of  $s$  and an appended wildcard character is  $\text{TAGTAGAT?}$ . This query string will match any probes in the spectrum with the form  $\text{TAG} \dots \text{G} \dots ?$ .

■

The query strings used to extend the sequence character-by-character always have a single free position. If the putative sequence is being extended to the right, the free position always corresponds to the rightmost character in the query string ( $q_\lambda$ ). If we wanted to extend the sequence to the left, we could use query strings with the leftmost ( $q_1$ ) position free (i. e.  $\text{?AG} \dots \text{C} \dots \text{A}$ ). In general, the free position could occupy any of the positions of natural bases within the probing pattern, but such queries have no purpose in SBH. (If the free position coincides with a universal base, it is effectively ignored, since all of the natural bases in  $q$  are fixed, turning the QUERY into a CHECK.)

The QUERY algorithm works by CHECKING the spectrum for each of the four possible matches to the query. For a query string  $q$  with a free position  $q_f$ , four checks are performed with  $q_f = A$ ,  $q_f = C$ ,  $q_f = T$ , and  $q_f = G$ . The results yield the set of characters which match the free position in the query string  $q$ , and hence the possible extensions to the putative sequence  $s$ . The pseudo-code for the QUERY function is:

```

QUERY( $\mathcal{S}, q$ )
1   $e \leftarrow \text{NIL}$ 
2   $f \leftarrow$  the free position in  $q$ 
3  for each  $c$  in  $\{ACGT\}$ 
4      do  $q_f \leftarrow c$ 
5          if  $\mathcal{S}.\text{CHECK}(q)$ 
6              then  $e \leftarrow e \cup c$ 
7  return  $e$ 

```

If  $e$  contains only a single character, then that character is appended to the current sequence, and the sequencing process continues—this is the *simple* mode of extension.

When more than a single character is returned by QUERY the problem is more interesting. We can picture the sequence as it is being reconstructed as a *path* through the space of possible sequences. The process of extending the sequence by a single character is equivalent to finding the next state in the path. When QUERY returns a single extension character, the next state in the path may be chosen unambiguously. However, multiple results to a QUERY, represent a *branch* in the search space (with up to 4 possible choices for the next state in the path, one branch for each character in *e*). This is the *branching* mode of extension. The correct path is the one which corresponds to the original sequence; the other paths are termed *spurious* paths. Figure 2.1 shows a representation of this process, in which there is an initial [C][G] branch in the sequencing process after the sequence GGCTAAGCGAGCGAT. Each of the two potential paths is extended by 10 additional characters (AGGATTCCAG) before another branch occurs in each of them. As the tree is extended, further branches occur, and some paths are eliminated from the tree.

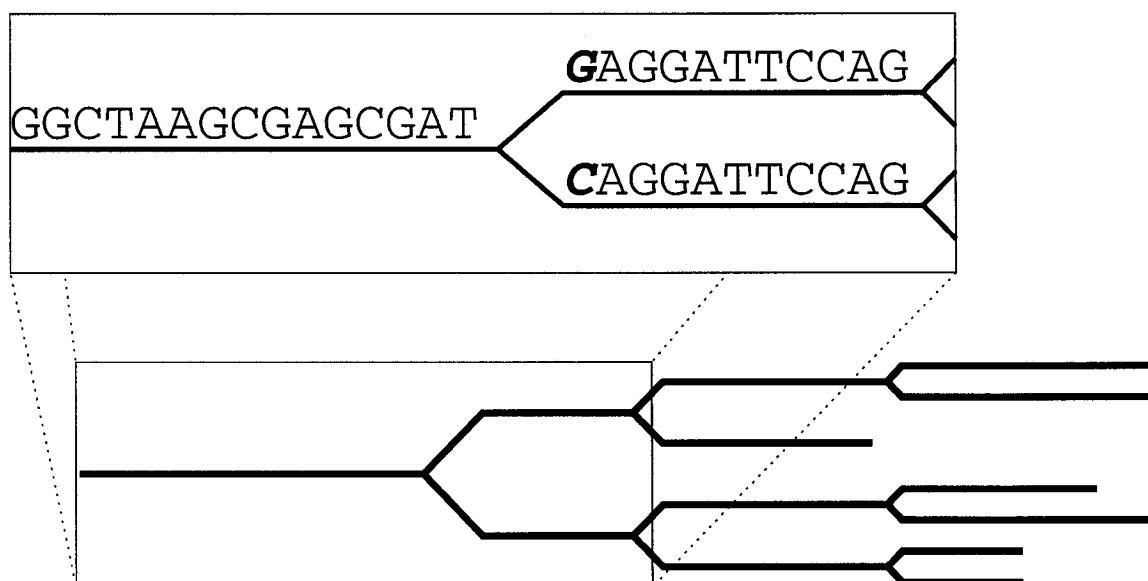


Figure 2.1: Diagram of a path tree created by the branching EXTEND algorithm with the initial portion blown up. The putative sequence is to the left of an initial [C][G] branch. Subsequent branches occur, and some potential paths are eliminated. Three paths survive at the deepest level in the tree.

When a branch occurs, the EXTEND algorithm attempts to continue extending

all of the possible paths  $p_1, \dots, p_n$ . At the point of the initial branch,  $n \leq 4$ , but additional branches may occur as all possible sequences are extended. The maximum number of potential paths after  $i$  characters (including the initial branch) is  $4^i$ . The branches are extended breadth-first until all of the remaining paths share a common non-empty prefix (corresponding to at least one edge in the tree), or the algorithm fails. We will examine the different modes of failure shortly, but first we want to discuss how branches are eliminated from the tree. We ignore the possibility of additional branches in the extended paths for the moment, and explore a detailed example.

Consider the putative sequence  $\dots \text{AAACCCCCCTTTT}$  which has been reconstructed so far by the SBH process from a reverse (3,3)-spectrum (with the structure  $N \dots N \dots N \dots \text{NNN}$  and parameters  $\kappa = 6$  and  $\lambda = 12$ ). The QUERY function returns the ambiguous result  $\{A, G\}$  to the query string  $\text{CCCCCCTTTT}^*$ , so the following two paths are created:

$$\begin{aligned} p_1 &= \text{CCCCCCTTTT} \text{A} \\ p_2 &= \text{CCCCCCTTTT} \text{G} \end{aligned}$$

Assume that the correct sequence is  $\dots \text{CCCCCCTTTT} \text{A} \text{AAAAG} \dots$ , corresponding to  $p_1$ . All of the probes necessary to continue extension of this path are, of course, present in the spectrum. On the other hand, we might be able to eliminate the path  $p_2$  from consideration by continuing sequence extension for an additional  $\lambda - 1$  characters, since the probes necessary to continue extension of path  $p_2$  are not guaranteed to be present. The probability of finding all of the necessary probes to confirm a spurious extension will be discussed in detail in the next chapter, along with a much more rigorous discussion of the various ways in which the reconstruction process may fail. Here, we simply provide a brief overview.

The initial branch in the example above could occur in two different ways. First, there may be two subsequences in the target sequence which match over  $\lambda - 1$  characters (i. e.  $\text{CCCCCCTTTT} \text{A}$  and  $\text{CCCCCCTTTT} \text{G}$ ). The second possibility is that there are two  $(\lambda - 1)$ -character subsequences in the target sequence which agree in only the  $\kappa - 1$  of their positions which correspond to the natural bases in the query, i. e.  $\text{CCC} \text{CCC} \text{TTT} \text{TTA}$  and  $\text{CAG} \text{CGG} \text{TAG} \text{TTG}$ . In the first case, both paths will continue to be extended indefinitely. The gapped SBH algorithm fails



in this case in the exact same manner and for the same reason that the ungapped algorithm fails: all of the probes required to confirm both the correct and spurious sequence are present in the spectrum.

**Example 2.7.** The sequence CC AGTAG ACAGC AGTAG CTTATT contains a repeated subsequence (AGTAG) of length 5. When attempting to reconstruct this sequence using (3,1)-direct probes (NNN...N), the query string AGT...? produces the ambiguous result {A, C}.

The two probes in the spectrum which match the query string, AGT...A and AGT...C occur in the target sequence at the locations of the duplicated subsequence. There are three subsequent spectrum queries which sample the ambiguous position. However, all three queries of the queries produces an ambiguous result, since each query matches two locations in the sequence. On the following lines, the probes which confirm each ambiguous character are displayed at the location in the sequence where they occur:

<i>Sequence:</i>	CC	AGTAG	ACAGC	AGTAG	CTTATT
<i>Probes:</i>		AGT...A		AGT...C	
		AG A...G		AG C...A	
		G AG...C		G CT...T	
		ACA...A		CTT...T	

This is just a simple example with a 5-character repeat and probes with length  $\lambda = 6$ . However, it illustrates the general case of a  $(\lambda - 1)$ -character duplicated subsequence causing a guaranteed sequencing failure for any probing pattern. ■

Fortunately, duplicate sequences of length  $\lambda - 1$  are highly unlikely (for sufficiently large  $\lambda$ ), since the likelihood of finding a duplicate sequence decreases by a factor of  $\frac{1}{4}$  with each additional character. The most likely reason for the initial ambiguous result to QUERY is that there are two strings which match only over the  $\kappa - 1$  positions which affect the spectrum search. In this case, we can usually eliminate the spurious branch by further extending both paths. This is because—unlike ungapped probes—gapped probes sample only  $\kappa$  out of  $\lambda$  positions within their span; adjacent probes typically have fewer than  $\frac{\kappa}{2}$  characters in common.

In our running example, there are  $\kappa - 1 = 5$  additional probes which sample the ambiguous position after the initial ambiguous query. These probes correspond to 1-, 2-, 5-, 8-, and 11-character shifts with respect to the position of the ambiguity. The character 'X' is used to represent any natural base.

Sequence: ...XXXXXXXXXX **X** XXXXXXXXXXXXXXX...

---

Probes:

N

..N

..N

..N

**NN**

N

..N

..N

**NN**

N

..N

**N**

..N

NN

N

..

**N**

..N

..N

NN

**N**

..N

..N

..N

NN

---

Again, assuming that there are no further ambiguous results to any calls to the QUERY algorithm, the spurious path can be eliminated if and only if one of the these five queries returns 0 results. As soon as a call to QUERY returns an empty set of characters, that path is eliminated and the other path may be used as the sequence extension. Note that once these 5 queries have been completed, there are no subsequent spectrum queries which sample the ambiguous character. If all 5 queries return at least a single result, there is no further chance of eliminating the false path. When this occurs, the algorithm cannot reconstruct an unambiguous sequence from the spectrum, and fails.

In practice, there is a chance that any of the paths (correct or spurious) may branch again before the initial ambiguity is resolved. In fact, as the length of the target sequence approaches the feasible limit for sequence reconstruction, the set of paths being extended can grow quite large. However, it is not necessary to reduce the set of paths to a single member; it suffices to eliminate all but one of the root branches. When all of the paths remaining in the set agree in at least their first position, the initial ambiguity is resolved, and sequencing can continue.

The actual circumstances which may result in sequencing failure are significantly more complex than we have described. The different modes of failure will be discussed later in this chapter, and the probability of each mode will be analyzed in Chapter 4. For now, we describe only how the EXTEND algorithm decides to stop extending a tree of paths. EXTEND has two bounding parameters: a depth-bound  $H$  and a breadth-bound  $B$ , which are necessary to limit the running time and space of the algorithm

when a failure occurs. If the tree of possible path extensions reaches either of these bounds when attempting to resolve an ambiguous extension, the process fails.

In summary, there are two modes of extension possible within the EXTEND algorithm. The simplest mode occurs only when there is only a single probe in the spectrum which matches the  $(\lambda - 1)$ -character suffix of the putative sequence. The branching mode of extension is initiated when there are two or more probes in the spectrum which match the  $(\lambda - 1)$ -character suffix of the putative sequence. In the branching mode, all possible paths are extended until all but one of the initial branches is eliminated.

The pseudo-code for the EXTEND algorithm and a helper function called PATHSPLIT, is presented here. PATHSPLIT is a utility function which accepts a single path string  $s$  (the string representing a possible path in the tree of potential paths) of length  $\ell$  and a set of  $2 \leq n \leq 4$  characters ( $e$ ) as arguments.  $e$  contains the set of possible extension characters to the putative sequence  $s$ . The function returns a set of  $n$  path strings of length  $\ell + 1$  (one for each possible extension) by appending each character in  $e$  to a copy of  $s$  in turn.

PATHSPLIT( $s, e$ )

```

1   $P \leftarrow \text{NIL}$ 
2  for each  $c$  in  $e$ 
3      do  $p = s + c$ 
4           $P \leftarrow P \cup p$ 
5  return  $P$ 
```

Finally, we reach the EXTEND function itself. It accepts two arguments: the spectrum  $\mathcal{S}$  and the putative sequence  $s$ . It returns a (possibly empty) string containing the extension to  $s$ .

EXTEND( $\mathcal{S}, s$ )

```

1   $q \leftarrow s_{|s|-(\lambda-1) \dots |s|} + ?$ 
2   $e \leftarrow \text{QUERY}(\mathcal{S}, q)$ 
3  if  $|e| = 0$  OR  $|e| = 1$ 
```

```

4   then return  $e$ 
5    $P \leftarrow \text{PATHSPLIT}(s, e)$ 
6    $Q \leftarrow \text{NIL}$ 
7   while  $i < H$  and  $|P| < B$ 
8     do for each  $p$  in  $P$ 
9       do  $q \leftarrow p_{|p|-(\lambda-1)\dots|p|} + ?$ 
10       $e \leftarrow \text{QUERY}(\mathcal{S}, q)$ 
11       $Q \leftarrow Q \cup \text{PATHSPLIT}(p, e)$ 
12     $P \leftarrow Q$ 
13    if  $|\text{Prefix}(P)| > 0$  OR  $|P| = 0$ 
14      then return  $\text{Prefix}(P)$ 
15  error "failure"

```

The EXTEND algorithm is the heart of the SEQUENCE algorithm, which actually stitches together all of the short segments produced by EXTEND. The SEQUENCE algorithm will be defined in the next section.

When the spectrum is free of errors (noise), the reconstruction is guaranteed to be error-free, with one notable exception. When a branch occurs within the depth bound of  $H$  characters of the end of the sequence, it is possible that the final segment produced by the EXTEND algorithm contains one or more incorrect characters. Consider the following example:

**Example 2.8.** A 10000-base target sequence is being reconstructed from its spectrum of (4,4)-direct probes. When the putative sequence is 9991 characters in length, there is an ambiguous query in EXTEND, and a branch occurs. EXTEND continues both the correct and spurious path for 9 characters. At this point, the correct path contains the characters necessary to complete the sequence reconstruction process, and the spurious path contains at least one incorrect character. However, further extension of the correct path is impossible, since it has reached the end of the target sequence.

If the putative path is ...GGACCGTG then the correct and spurious paths (with the ambiguous character highlighted) are:

```

...GGACCGTG A AAGAGTTCAC
...GGACCGTG T AAGAGTTCAC

```

In order for either of these sequences to be extended by an additional character, there must be a probe in the spectrum which matches ‘GGAC...G...G...T...?’ Note that since this probe doesn’t sample the ambiguous character, a single matching probe suffices to extend both the correct and spurious paths.

If there is no probe in the spectrum which matches the query, both paths are eliminated simultaneously, and the entire tree is eliminated. In this case, the *Prefix* of the tree is empty, and EXTEND returns a 0-character extension, signalling the end of reconstruction. The last 9 characters of the sequence are lost, but the 9991-character sequence which is successfully produced contains no errors. On the other hand, if there is a probe in the spectrum which matches the query, then *both* paths are extended. Probes that allow the extension of spurious paths are called *fooling probes*, which are defined in Section 2.6 and discussed in detail in Section 3.1.

The next 1-character extension of the paths requires probes in the spectrum which sample the ambiguous character. The correct path is extended if there is a probe in the spectrum matching ‘GACC...A...A...C...?’, and the spurious path is extended if the spectrum contains a probe which matches ‘GACC...T...A...C...?’ There are four possible outcomes to this pair of queries:

1. Neither query produces a response; there are no fooling probes in the spectrum which extend either path.
2. The correct path is extended by a fooling probe, and the spurious path is eliminated.
3. The spurious path is extended by a fooling probe, but the correct path does not get extended, and is eliminated.
4. There are fooling probes in the spectrum which confirm both paths, and they are both extended by one more character.

We’ll consider the consequences of each potential result separately. If Option 1 occurs, both paths are eliminated simultaneously, and EXTEND returns an empty segment. The reconstruction of the target sequence halts at 9991 characters, just as though the previous query had produced no responses.

When Option 2 takes place, then the spurious path is eliminated, and only the correct path remains. EXTEND produces an 11-character segment: the first 9 characters correspond to the last 9 bases of the target sequence, but the last 2 are just algorithmic artifacts, produced by a pair of fooling probes in the spectrum. In the software SBH simulation, we know the precise length of the target sequence, and so we can discard these extra bases, but this is a luxury not afforded by real biological data.

Option 3 is the worst case in terms of the number of errors added to the sequence, and it is just as likely as Option 2 since either case results from a single fooling probe. When there is a fooling probe which extends the false path, but no corresponding probe which extends the correct path, then the *correct* path is eliminated. EXTEND produces an 11-character sequence; but, in addition to the two extra characters beyond the end of the target sequence, the first character of the segment—the 9992<sup>nd</sup> character of the target sequence—is also incorrect.

In the event that Option 4 occurs—this is the most unlikely result, since it requires the co-occurrence of 2 different fooling probes—both paths are extended by a single character. The next three queries do not sample the ambiguous position, so if either path is extended then both paths are extended. If any one of the next three queries produces no responses, then neither path is extended and EXTEND produces an empty segment. The fourth query after this again samples the ambiguous character, so potential outcomes of that extension attempt are the same as the four outcomes described here. ■

The preceding example shows how incorrect characters may be included in the reconstructed sequence, as well as how the reconstructed sequence might end up a few bases longer or shorter than the actual sequence. The extra (or missing) bases at the ends of a reconstructed sequence are called the overflow (or underflow). While none of these minor errors are very serious—they almost always occur within  $\lambda$  bases of the end of a sequence—it is possible to prevent them from occurring at all.

Recall that PCR requires that short primers *at each end* of the amplified sequence (the target sequence) be known. If one of the PCR primers is used to begin the sequencing process, the other PCR primer can be used as a *terminating* segment, used to indicate that the sequencing should halt. PCR primers are typically at least

20 bases in length and they are specifically chosen to be unique, so the probability of finding one of the PCR primers in the middle of the target sequence is negligible (and would cause problems during the amplification process before sequencing had even begun). Therefore, we can assume that when sequencing begins from one of the primers, it is finished when the other primer has been reached.

Alternately, since sequencing errors produced by fooling probes are limited to the  $\lambda$  characters at the end of the sequence, we can simply choose to ignore them if PCR primers are not available. The reconstructed sequences are still guaranteed to be correct over nearly their entire length; the last  $\lambda$  characters produced by EXTEND can simply be discarded or tagged to indicate the potential presence of an incorrect base.

Using the second PCR primer as a terminating segment requires that the EXTEND algorithm continually compare the last  $n$  characters of each of the tree paths to the terminating segment. If any of the paths matches the terminating segment (when appended to the putative sequence), that path *must* be the correct extension: that path can be selected as the correct path in the tree. The terminating segment must of course be passed as an argument (*term*) to EXTEND:

EXTEND( $\mathcal{S}, s, term$ )

```

1   $q \leftarrow s_{|s|-(\lambda-1)...|s|} + ?$ 
2   $e \leftarrow \text{QUERY}(\mathcal{S}, q)$ 
3  if  $|e| = 0$  OR  $|e| = 1$ 
4    then return  $e$ 
5   $P \leftarrow \text{PATHSPLIT}(s, e)$ 
6   $Q \leftarrow \text{NIL}$ 
7  while  $i < H$  and  $|P| < B$ 
8    do for each  $p$  in  $P$ 
9      do  $q \leftarrow p_{|p|-(\lambda-1)...|p|} + ?$ 
10      $e \leftarrow \text{QUERY}(\mathcal{S}, q)$ 
11      $Q \leftarrow Q \cup \text{PATHSPLIT}(p, e)$ 
12    $P \leftarrow Q$ 
13   for each  $p$  in  $P$ 
```

```

14         do  $t \leftarrow s + p$ 
15         if  $t_{|t|-|term|...|t|} = term$ 
16         then return  $p$ 
17     if  $|Prefix(P)| > 0$  OR  $|P| = 0$ 
18     then return  $Prefix(P)$ 
19 error "failure"

```

The EXTEND algorithm produces a segment which extends a given sequence; it is left to another algorithm to stitch all of these segments together into a complete sequence. The next section introduces exactly such a function, appropriately named SEQUENCE.

## 2.4 The SEQUENCE Algorithm

The real sequencing work is performed by the EXTEND algorithm. Nonetheless, another function is needed to assemble the segments returned by EXTEND into a complete sequence. The following pseudo-code defines the SEQUENCE function, which takes a spectrum  $\mathcal{S}$  and a seed as its two parameters, and returns the completely reconstructed target sequence. If a failure occurs within the EXTEND algorithm, the entire sequencing process fails and no sequence is produced at all.

```

SEQUENCE( $\mathcal{S}, seed$ )
1   $s \leftarrow seed$ 
2   $e \leftarrow \text{EXTEND}(\mathcal{S}, s)$ 
3  while  $|e| > 0$ 
4      do  $s \leftarrow s + e$ 
5       $e \leftarrow \text{EXTEND}(\mathcal{S}, s)$ 
6  return  $s$ 

```

Integrating the concept of a terminating segment requires only a pair of additional lines in the code, to compare the  $n$ -suffix of the putative sequence to the terminator. The EXTEND algorithm will catch all of the terminating segments which occur in the



branching mode; the check in SEQUENCE is really only necessary after EXTEND has produced a 1-character extension without branching. The terminating segment must also be passed as an argument to SEQUENCE:

```

SEQUENCE( $\mathcal{S}$ , seed, term)
1   $s \leftarrow \textit{seed}$ 
2   $e \leftarrow \text{EXTEND}(\mathcal{S}, s)$ 
3  while  $|e| > 0$ 
4      do  $s \leftarrow s + e$ 
5          if  $s_{|s|-|term| \dots |s|} = \textit{term}$ 
6              then return  $s$ 
7           $e \leftarrow \text{EXTEND}(\mathcal{S}, s)$ 
8  return  $s$ 

```

As presented, this algorithm can only extend a sequence to the *right*; this mode of sequencing is called *forward* sequencing. There are several locations in the code where the left-to-right order is enforced: whenever a query string is created with  $q_{|q|}$  as the free position, and whenever a character or segment is appended to an existing sequence.

## 2.5 Changing Direction: Reverse Sequencing

We could modify the functions so that they perform *reverse* sequencing (right-to-left) by making the appropriate algorithmic changes, but we can also use the existing functions as-is to perform reverse sequencing by modifying the spectrum itself. We will discuss both options in turn.

Algorithmic changes must be made to the EXTEND, PATHSPLIT and SEQUENCE algorithms to allow them to perform reverse sequencing. First, the query strings used in EXTEND must be modified so that the putative sequence is extended to the left instead of to the right. The query string to extend a sequence  $s$  one character to the left is  $q = ? + s_{1 \dots \lambda-1}$ . The set of characters resulting from a query performed with such a string contains the set of possible extensions to the left of the current string.

Second, wherever a character or segment of characters is *appended* to an existing sequence or path, the code must be modified to *prepend* the character instead. The PATHSPLIT function must be modified to prepend possible extension characters to existing path strings, and the SEQUENCE function must be modified to prepend segments as they are produced by EXTEND.

The modified versions of PATHSPLIT, EXTEND, and SEQUENCE to perform reverse sequencing are presented in that order. The only change made to the PATHSPLIT function is on line 3, where the character  $c$  is prepended instead of appended to  $s$ :

PATHSPLIT-REVERSE( $s, e$ )

```

1   $P \leftarrow \text{NIL}$ 
2  for each  $c$  in  $e$ 
3      do  $p = c + s$ 
4       $P \leftarrow P \cup p$ 
5  return  $P$ 

```

In the EXTEND-REVERSE algorithm, lines 1 and 9 have been modified to change the position of the free character in the query string. Also,  $\text{Suffix}(P)$  must be used in place of  $\text{Prefix}(P)$ . That is, we must find the characters common to all strings in the path tree starting at the right-most character of each of the string and moving to the left, instead of left to right.

EXTEND-REVERSE( $\mathcal{S}, s$ )

```

1   $q \leftarrow ? + s_{1 \dots \lambda-1}$ 
2   $e \leftarrow \text{QUERY}(\mathcal{S}, q)$ 
3  if  $|e| = 0$  OR  $|e| = 1$ 
4      then return  $e$ 
5   $P \leftarrow \text{PATHSPLIT}(s, e)$ 
6   $Q \leftarrow \text{NIL}$ 
7  while  $i < H$  and  $|P| < B$ 
8      do for each  $p$  in  $P$ 
9          do  $q' \leftarrow ? + p_{1 \dots \lambda-1}$ 

```

```

10           $e \leftarrow \text{QUERY}(\mathcal{S}, q')$ 
11           $Q \leftarrow Q \cup \text{PATHSPLIT}(p, e)$ 
12           $P \leftarrow Q$ 
13          if  $|\text{Suffix}(P)| > 0$  OR  $|P| = 0$ 
14              then return  $\text{Suffix}(P)$ 
15 error "failure"

```

The only change made to `SEQUENCE` is on line 4, where the extension segments produced by `EXTEND-REVERSE` are added to the beginning of the putative sequence instead of to the end:

```

SEQUENCE-REVERSE( $\mathcal{S}, seed$ )
1   $s \leftarrow seed$ 
2   $e \leftarrow \text{EXTEND}(\mathcal{S}, s)$ 
3  while  $|e| > 0$ 
4      do  $s \leftarrow e + s$ 
5       $e \leftarrow \text{EXTEND}(\mathcal{S}, s)$ 
6  return  $s$ 

```

With the addition of these three `REVERSE` functions, we can start from a seed which occurs anywhere in the target sequence, and extend it in both directions until the original sequence has been reconstructed. However, there is one implicit difference between the forward and reverse versions of the algorithms. If a spectrum is constructed using a direct  $(s, r)$ -probing pattern, then forward sequencing is conducted with the direct pattern. But reverse sequencing is effectively conducted with the reverse of the pattern. Consider the following example:

The spectrum  $(\mathcal{S})$  of (3,3)-direct probes for the sequence  $s = \text{GCGATGAG-TATTGA}$  contains the 6 probes  $\{\text{GCG} \dots \text{G} \dots \text{T}, \text{CGA} \dots \text{A} \dots \text{A}, \text{GAT} \dots \text{G} \dots \text{T}, \text{ATG} \dots \text{T} \dots \text{T}, \text{TGA} \dots \text{A} \dots \text{G}, \text{GAG} \dots \text{T} \dots \text{A}\}$ . A reverse query of the spectrum with the string  $q = \text{?GAGTATTG}$  matches only the probe  $\text{CGA} \dots \text{A} \dots \text{A}$ .

The spectrum  $(\mathcal{S}')$  constructed using (3,3)-reverse probes for the reverse of the string  $s$ ,  $s' = \text{AGTTATGAGTAGCG}$  contains the 6 probes  $\{\text{A} \dots \text{T} \dots \text{GAG},$

G . . A . . AGT, T . . T . . GTA, T . . G . . TAG, A . . A . . AGC, T . . G . . GCG}. If this spectrum is queried with the reverse of the query string  $q$ ,  $q' = \text{GTTATGAG?}$ , there is a single match in the spectrum: A . . A . . AGC. Note that probe matching this query is the exact reverse of the probe which matched the forward query. In fact, each of the probes in the spectrum  $\mathcal{S}'$  is the reverse of a probe in  $\mathcal{S}$ . This means that any query of the spectrum  $\mathcal{S}$  a query string  $q$  can be functionally duplicated by querying the spectrum  $\mathcal{S}'$  with the reverse of  $q$ : the set of characters which match the free position in the forward and reverse versions of the query are identical.

**Theorem 2.1.** *Reconstructing a target sequence  $s$  by means of reverse sequencing with a direct probing pattern is equivalent to reconstructing the reverse of  $s$  ( $s'$ ) by means of forward sequencing with the corresponding reverse probing pattern.*

The equivalence of forward and reverse queries simplifies the implementation of the SBH software simulation. Instead of including multiple versions of the PATH-SPLIT, EXTEND and SEQUENCE algorithms, we simply construct forward and reverse spectra for the target sequence. Forward reconstruction from a seed to the end of the target sequence uses the forward spectrum. Similarly, the reverse spectrum is used to perform reverse sequencing. (Of course, the final result of the reverse sequencing process is the reverse target sequence  $s'$ , which must then be reverse again to yield  $s$ .)

The only necessary changes to the pseudo-code is in the initial call to SEQUENCE. If the forward and reverse spectra are denoted  $\mathcal{S}$  and  $\mathcal{S}'$ , then forward sequencing is initiated with the standard call  $\text{SEQUENCE}(\mathcal{S}, \text{seed})$ . Reverse sequencing can be started by calling  $\text{SEQUENCE}(\mathcal{S}', \text{Reverse}(\text{seed}))$ .

The next section introduces the different failure modes of the algorithm.

## 2.6 Sequencing Failure Modes

We previously introduced the two bounding parameters  $B$  (the breadth bound) and  $H$  (the depth bound) of the path tree. This section contains a discussion of the circumstances in which these bounds are reached. Virtually all sequencing failures occur because the depth-bound  $H$  is reached. As long as the breadth-bound  $B$  is set to a sufficiently high value, it is never reached, although some probing patterns

do result in wider (and therefore *larger*) path trees than others. Chapter 3 contains an analysis of *why* this is the case. We will discuss the depth-failures first. These failures may be divided into two different classes, which we will call *Mode 1* and *Mode 2* failures.

Mode 1 failures are characterised by a single ambiguous character, followed by strings which are identical in each of the branches. For instance, the sequences

...GA GGACCCAGTTAGGATGCAAAGCGT  
and  
...GA CGACCCAGTTAGGATGCAAAGCGT

differ in only a single position, after which there is a long sequence of identical characters. Once  $\lambda - 1$  consecutive matching characters have been observed, there is no possibility of resolving the ambiguity, since the query strings which extend both sequences are guaranteed to be identical subsequent to the ambiguous position. Mode 1 failures occur when there are  $\kappa$  fooling probes in the spectrum which confirm the spurious extension character. We now give a formal definition of a fooling probe:

**Definition 2.3.** A fooling probe matches a query over all  $\kappa - 1$  specified positions in a query at position  $i$ , but occurs at some other position  $j \neq i$  in the target sequence.

**Example 2.9.** If the correct extension to the putative sequence  $s = \text{AAGGAG-TATATC}$  is G, then the (3,3)-reverse probe which produces the correct extension is  $\text{A} \dots \text{A} \dots \text{TC} \text{ G }$ . There are three possible fooling probes which might exist in the spectrum:  $\text{A} \dots \text{A} \dots \text{TC} \text{ A }$ ,  $\text{A} \dots \text{A} \dots \text{TC} \text{ C }$  and  $\text{A} \dots \text{A} \dots \text{TC} \text{ T }$ . ■

**Example 2.10.** When attempting to extend a putative sequence  $s = \text{AGAGATT-GAGGGT}$  with (3,3)-direct probes, an ambiguous extension is produced:  $\{A, G\}$ . Assume that the correct sequence is  $\dots \text{AGAGATTGAGGGT} \text{ AGGATAG }$ . The first three probes which confirm the *correct* extension including the initial ambiguous extension are  $\text{TTG} \dots \text{G} \dots \text{ A }$ ,  $\text{AGG} \dots \text{ A } \dots \text{G}$  and  $\text{GT} \text{ A } \dots \text{G} \dots \text{T}$ . Additional extension-queries must be performed in order to extend both the correct and spurious path by 6 characters beyond the branch, however these additional queries do not sample the ambiguous character: they contain a universal base in the ambiguous position.

The spurious sequence will only be extended further if there is a valid set of fooling probes present in the spectrum, such as:  $\{ \text{TTG} \dots \text{G} \dots \text{G}, \text{AGG} \dots \text{G} \dots \text{G}, \text{GT} \text{G} \dots \text{G} \dots \text{T} \}$ . Note that each of these three fooling probes differs by only a single character (highlighted in grey) from the corresponding *correct* probe. For instance, the second query which samples the ambiguous character produces the correct probe  $\text{AGG} \dots \text{A} \dots \text{G}$  and the fooling probe  $\text{AGG} \dots \text{G} \dots \text{G}$ . These two probes confirm the correct and spurious path, respectively, and differ only in the 6<sup>th</sup> position. ■

The set of fooling probes which confirm a spurious extension may be considered to be independent of one another. They are typically scattered uniformly along complete length of the target sequence, although there is a small chance that more than one of them occurs in the same location [PU00]. This will be discussed in more detail in the next chapter.

The other type of depth-bound failure (Mode 2) observed when reconstructing sequences is characterized by two sequences which diverge completely at some point within  $\lambda$  characters of the initial ambiguous character. Divergence can be measured by counting the positions which match between two sequences. Approximately  $\frac{1}{4}$  of the characters in two unrelated sequences can be expected to match. For example, the sequences

$\dots \text{GA} \text{G} \text{GACCCAGTT} \text{AG} \text{G} \text{AT} \text{CA} \text{AAGCCG} \text{T}$   
 and  
 $\dots \text{GA} \text{T} \text{GACCCAGTT} \text{GA} \text{G} \text{TC} \text{CA} \text{GGTAG} \text{G} \text{C}$

appear to diverge after the AGTT block of characters, 10 characters past the initial  $\{G, T\}$  ambiguity.

The simplest Mode 2 failure occurs when there is a  $(\lambda - 1)$ -character string duplicated somewhere in the target sequence. In this case, a branch occurs and both paths are extended indefinitely, since each extension is a valid subsequence of the target. However, as we mentioned earlier, exact duplicates of  $\lambda$ -character segments are exceedingly unlikely to occur in random sequences.

In practice, for a Mode 2 failure to occur at position  $i$ , two conditions must be satisfied. First, there must be a  $(\lambda - 1)$ -character subsequence of the target sequence at position  $j \neq i$  which is very similar to the  $(\lambda - 1)$ -character subsequence occurring

at  $i$ . We denote the number of differences between the two subsequences  $\delta$ , where  $\delta \leq \lambda - 1$ . If  $\delta = 0$ , a Mode 2 failure is guaranteed to occur. However, when  $\lambda > 2 \cdot (s + r)$ , this is extremely unlikely.

If  $\delta > 0$ , then a Mode 2 failure occurs only if there is a set of fooling probes in the spectrum which compensate for the differences between the two subsequences. The number of fooling probes required depends on the value of  $\delta$  and the placement of the differences between the two similar subsequences. The analysis of the likelihood of this type of failure is complex, as presented in [HP01]. The next chapter contains a thorough exploration of the probability of Mode 2 failures.

Finally, we want to consider the possibility that the total number of paths being extended from an ambiguous branch reaches the breadth-bound  $B$ . When reconstructing maximum entropy random DNA sequences, this type of failure only occurs with a particular class of probing pattern, and only when  $B$  is set to a relatively small value (256 or lower). If we restrict our view to only  $(s, r)$ -probes, the two broad classes of probing patterns are *direct* and *reverse* patterns. We find—perhaps unintuitively—that breadth-bound failures occur only when using *direct* probing schemes. We now explain why this is the case.

Preparata and Upfal [PU00] show that (4,4) and (5,3) direct and reverse probing patterns are all virtually identical in terms of expected sequencing performance. Extensive simulations bear out their calculations when  $B$  is set sufficiently high ( $B = 2048$ ). However, when  $B$  is reduced to 256 (in the interests of algorithmic efficiency), the direct probing patterns begin to encounter breadth-bound failures. The reason for this is fairly simple.

When the tree of paths originating from an ambiguous extension is extended further, there are only  $\kappa - 1$  probes which can eliminate a spurious path. Each of these probes corresponds to a shift of  $i$  characters with respect to the initial probe position. The  $\lambda - \kappa$  shifts within  $\lambda$  positions of the initial ambiguity which do not sample the ambiguous position cannot eliminate a path from the tree, but they *can* spawn additional branches.

The following example illustrates the difference between (4,3) direct and reverse probing patterns. The putative path is displayed as a series of X's, the ambiguous character as a Y, and the characters after the ambiguous position as Z's. Probes

which may result in the elimination of the spurious branch are labeled with *elim* at the beginning of the line. The first 4 probe shifts after an ambiguous branch are shown for (4,3)-direct and reverse patterns. The forward pattern is shown first:

<i>Sequence:</i>	XXXXXXXXXXXXXXXXXX	Y	ZZZZ
<i>elim</i>	XXXX . . . X . . . X . . .	Y	
	XXXX . . . X . . . X . . .		Z
	XXXX . . . X . . . X . . .		Z
	XXXX . . . X . . . X . . .		Z
<i>elim</i>	XXXX . . . X . . .	Y . . .	Z

And the first 4 shifts for the reverse (4,3)-pattern are:

<i>Sequence:</i>	XXXXXXXXXXXXXXXXXX	Y	ZZZZ
<i>elim</i>	X . . . X . . . X . . . XXX	Y	
<i>elim</i>	X . . . X . . . X . . . XX	Y	Z
<i>elim</i>	X . . . X . . . X . . . X	Y	ZZ
<i>elim</i>	X . . . X . . . X . . .	Y	ZZZ
	X . . . X . . . X . . .		ZZZZ

Let's compare the difference between direct and reverse patterns over these initial shifts. Using a reverse pattern, each of the first three shifts *after* the initial ambiguous character has the potential to eliminate the spurious branch, since they do not sample the ambiguous position. On the other hand, when direct probes are used, the first three shifts after the ambiguity cannot possibly eliminate the spurious branch. Moreover, if there is another branch in *one* of the paths, *all* of the paths in the tree will split, resulting in at least a doubling in the number of paths in the tree, rather than just an additive increase. (This will be explored in more detail in Chapter 3.)

Of course, once the paths in the tree have been extended by  $\lambda = 16$  characters, there have been  $\kappa = 7$  queries which could potentially eliminate any spurious paths. The only difference between forward and reverse patterns is in the maximum width of the path tree. While this affects execution time of the algorithm in a very significant way, the difference between the direct and reverse versions of probing patterns has a much more subtle effect on the length of sequence which can be reconstructed by the SBH algorithm.



In particular, direct and reverse (4,4) and (5,3) probing patterns are all approximately equivalent in terms of sequencing performance. Conveniently, most of the analysis and simulation results presented in this dissertation deal with (4,4) and (5,3) probes, since they are the optimal  $(s, r)$ -probes for  $\kappa = 8$ . We provide a brief survey of results from sequencing simulations in the next section.

## 2.7 Logging Algorithmic Behaviour

The algorithms and data structures presented thus far are sufficient to conduct the reconstruction of a target sequence from its spectrum. Theoretically, they are complete. In practice, we would like to have a detailed record of the nature of the work performed by SEQUENCE and related algorithms. This record serves a double purpose: it can offer insight into the behaviour of the algorithms themselves, and allow us to verify analytical predictions regarding their behaviour and performance. To this end, we add more data members to the SPECTRUM data structure, introduce a new SEGMENT structure for use by the EXTEND algorithm, and modify the existing algorithms to make use of these changes. The modifications to the SPECTRUM object are discussed first.

Each probe in the spectrum originates at a particular location in the target sequence, which can be identified by the position of the leftmost character in the probe in the sequence. Probes may occur multiple times in the target sequence, and thus be said to occur at a corresponding number of locations within it. It can be useful to track the source locations of each of the probes in the spectrum. This allows us to determine how many of the probes in the spectrum are duplicated (or triplicated, or  $n$ -plicated) in the sequence. Since the number of locations matched by each probe varies, a list (or similar data structure) must be used to store the source locations for each probe. An array of lists is added to the SPECTRUM structure to track the probe locations, and a function LOCATION is added to return the list of locations matched by a probe. The new SPECTRUM object is:

```
SPECTRUM
  variable ProbePattern
```

```

variable ProbeArray[1...4κ]
variable ProbeLocations[1...4κ]
function BUILD(TargetSequence)
function CHECK(DecimalProbe)
function CHECK-S(StringProbe)
function LOCATIONS(DecimalProbe)

```

The pseudo code for the new LOCATIONS function is only a minor modification of the CHECK function; the only difference is the array which is accessed:

```

SPECTRUM.LOCATIONS(d)
1  return ProbeLocations[d]

```

And the SPECTRUM.BUILD function must also be modified to store the locations of probes when the spectrum is constructed. There is one new bit of notation introduced here: the PUSH-BACK function. The STL offers a standard interface to most standard containers, and PUSH-BACK adds an element to the end of most containers (single and doubly-linked lists, arrays, stacks, etc...). We will use these functions in our pseudo-code.

```

SPECTRUM.BUILD(S)
1  for i ← 1 to (|S| − λ)
2      do p ← Si...i+λ
3          d ← ProbePattern.PACK(p)
4          ProbeLocations[d].PUSH-BACK(i)
5          ProbeArray[d] ← true

```

Later chapters introduce algorithms which require further modifications to the spectrum. These changes will be introduced and described as needed, but in general they require changes similar to those made to integrate the *ProbeLocation* array. In contrast, the SEGMENT object requires significant changes to EXTEND.

The EXTEND algorithm is a fairly simple algorithm with complex behaviour. The

analysis of SBH conducted in Chapters 3 and 4 depends largely on a detailed record of EXTEND's execution. Everything from the number of responses to the initial call to QUERY to the ultimate size of the path tree can be of use in guiding and verifying the algorithmic analysis, so virtually everything is recorded. Each call to EXTEND produces a segment of one or more characters to be added to the putative sequence, so the data describing the production of that segment is kept in an object also containing the segment itself. The object is called a SEGMENT, and the data collected within is listed here:

- *String* - The segment must at least contain the characters produced by EXTEND.
- *Type* - The SEGMENT object can be used for the initial *seed* segment, the terminating (*term*) segment, and the single-character *simple* and multi-character *branch* segments produced by EXTEND. This is an enum variable which holds the type of the segment.
- *Direction* - Segments can be produced from either the *forward* or *reverse* spectrum. When sequencing begins in the middle of the target sequence, some segments will be produced from each. This variable records the spectrum from which a segment was produced.
- *InitialQueryResponses* - This is used to count the number of probes which match the initial call to QUERY. It will always be 1 for *simple* segments, but will be 2, 3 or 4 for *branching* segments.
- *TreeDepth* - A number in the range  $1 \dots H$  which records the depth of the tree created in the branching mode of EXTEND.
- *TreeWidth* - This is an array of *TreeDepth* integers, which records the width of the tree at each depth from 1 to *TreeDepth*. The total size of the tree can be calculated by adding the width of the tree at each level.

In later chapters, further data will be added to this list, but these values are the most useful for Chapter 3's analysis of SBH. The pseudo-code for the SEGMENT object is simple:

SEGMENT

```

variable String
variable Type
variable Direction
variable InitialQueryResponses
variable TreeDepth
variable TreeWidth[1... TreeDepth]
function TREESIZE()
  .

```

The EXTEND algorithm must be modified to set the appropriate values in the SEGMENT it produces, and to return a segment, rather than a string. SEQUENCE no longer tracks the putative sequence simply as a string of characters, but rather as a linked-list of SEGMENT objects. The pseudo-code for the modified—and final—EXTEND and SEQUENCE algorithms follows.

EXTEND( $\mathcal{S}, s, term$ )

```

1  seg ← new SEGMENT
2  q ←  $s_{|s|-(\lambda-1)...|s|} + ?$ 
3  e ← QUERY( $\mathcal{S}, q$ )
4  seg.InitialQueryResponses ← e.SIZE()
5  if  $|e| = 0$  OR  $|e| = 1$ 
6    then seg.String ← e[0]
7         seg.Type ← simple
8    return seg
9  P ← PATHSPLIT(s, e)
10 Q ← NIL
11 while  $i < H$  and  $|P| < B$ 
12   do seg.TreeWidth.PUSH-BACK( $|P|$ )
13     for each p in P
14       do q ←  $p_{|p|-(\lambda-1)...|p|} + ?$ 
15         e ← QUERY( $\mathcal{S}, q$ )
16         Q ← Q ∪ PATHSPLIT(p, e)

```

```

17       $P \leftarrow Q$ 
18      for each  $p$  in  $P$ 
19          do  $t \leftarrow s + p$ 
20              if  $t_{|t|-|term|...|t|} = term$ 
21                  then  $seg.String \leftarrow p$ 
22                       $seg.TreeDepth \leftarrow i$ 
23                       $seg.Type \leftarrow term$ 
24                      return  $seg$ 
25      if  $|Prefix(P)| > 0$  OR  $|P| = 0$ 
26          then  $seg.String \leftarrow Prefix(P)$ 
27               $seg.TreeDepth \leftarrow i$ 
28               $seg.Type \leftarrow branching$ 
29              return  $seg$ 
30      error "failure"

```

In the SEQUENCE algorithm, the *seed* parameter must now be a seed SEGMENT (the first segment in the linked list forming the sequence). All subsequent segments are appended to the list with the PUSH-BACK function. For convenience, we assume that the last  $n$  characters in the list/sequence can be accessed with a call to SUFFIX( $n$ ).

SEQUENCE( $\mathcal{S}, seed, term$ )

```

1   $s \leftarrow newListofSEGMENT$ 
2   $s.PUSH-BACK(seed)$ 
3   $e \leftarrow EXTEND(\mathcal{S}, s)$ 
4  while  $|e.String| > 0$ 
5      do  $s.PUSH-BACK(e)$ 
6          if  $s.SUFFIX(|term|) = term$ 
7              then return  $s$ 
8               $suffix \leftarrow s.SUFFIX(\lambda)$ 
9               $e \leftarrow EXTEND(\mathcal{S}, suffix)$ 
10     return  $s$ 

```

A few additional steps take place after sequencing has been completed, to verify that the reconstructed sequences do indeed match the original, count the overflow or underflow if terminating segments are not used, and calculate some aggregate statistics over the length of the sequence. The average length of the SEGMENT strings, the average number of responses to EXTEND's initial query, and other average values are computed. All of this data is printed to an XML format file for later analysis. In all, a single successful sequencing attempt for a 12Kbp string produces about 1.5MBytes of log data. Some of these logged results are presented in the last section of this chapter.

## 2.8 Initial Performance Observations

The performance of any sequencing method is measured in terms of the longest sequences which may be reconstructed using the technique. Electrophoresis techniques have inherent limitations which restrict their use to DNA fragments of about 600bp. The nature of the SBH algorithm is such that it is possible to construct a very short sequence which will cause the method to fail. On the other hand, SBH with gapped probes can be used to reconstruct extremely long sequences. Consequently, we make the following definition of sequencing performance:

**Definition 2.4.** *The performance measure for the SBH algorithm is defined to be the length of maximum-entropy sequences which may be correctly reconstructed with some constant confidence factor  $\epsilon$ .*

Except where it is explicitly stated otherwise, the confidence factor used in this dissertation is 0.9. Thus, when we say that a given technique is effective for sequences up to 12Kbp in length, we mean that the technique has at least a 90% probability of successfully reconstructing any given uniformly randomly-generated DNA sequence of length 12Kbp or less. Experimental performance results were derived from at least 250 attempts to reconstruct different randomly generated sequences; the performance is measured by the proportion of correct reconstructions. Analytical results, on the other hand, are measured by the expected probability of successful reconstruction. In either case, the *performance* of the SBH technique for sequences of a given length  $m$

can be stated as a fraction between 0 and 1, indicating the proportion of expected or observed successes.

When the length of random target sequences is less than 10 Kbp, Mode 1 failures dominate, accounting for well over 90% of all observed sequencing failures. As  $m$  increases, the proportion of Mode 2 failures increases. For natural DNA sources, Mode 2 failures dominate Mode 1 from the outset, due to a much higher frequency of long repeats. The incidence of repeats in natural DNA was studied in some detail, and a model of the repeat structure of natural DNA will be presented later in this thesis.

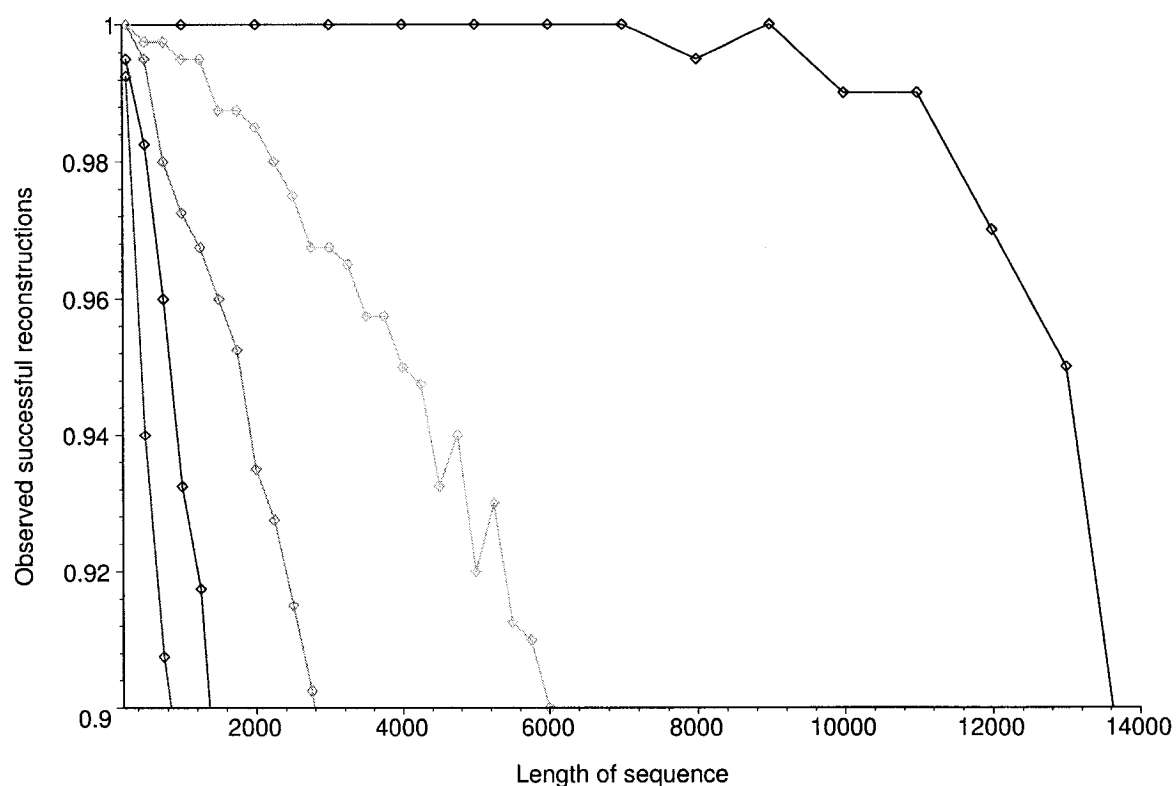


Figure 2.2: Observed proportion of sequences successfully reconstructed by the branching SBH algorithm using reverse (4,4) probes. The top curve is random DNA, the remaining four are *salmonella h. influenzae*, human chromosome 3, and *a. thaliana* respectively from right to left.

When running on random data, the Gapped-SBH algorithm can successfully reconstruct sequences of length  $m \approx 14000$ . This is nearly  $1/2$  of the information

theoretic bound of 32768, and about 7 times the performance of the first gapped-probe method introduced in [PFU99]. More impressively, it is nearly 50 times the performance achieved by the oligonucleotide-based SBH method proposed in 1988 and over 20 times as long as electrophoresis methods currently in use.

Figure 2.2 displays the performance of the SBH algorithm on random data is compared to its performance on a selection of natural DNA. It can be easily seen that performance on natural DNA degrades gracefully on certain sequences (e. g. *salmonella*, *h. influenzae*). When reconstructing other sequences (such as *a. thaliana* and the human genome), a more catastrophic falloff is observed. Some of these problems can be dealt with by additional algorithmic trickery. Others are probably hopeless, although the hopeless cases by no means render this sequencing method useless.

The approximate length  $m$  of sequences which may be reconstructed with confidence  $\epsilon = 0.9$  using reverse (4, 4)-probes is presented in the following table.

Sequence Source	Maximum $m$
Random DNA	13800
<i>e. coli</i>	3700
<i>h. influenzae</i>	2900
<i>p. falciparum</i>	200
<i>h. sapiens</i> chr. 3	400

In contrast to the failures observed for random data, the algorithm frequently fails because the breadth-bound  $B$  is reached when attempting to reconstruct natural DNA sequences. However, increasing  $B$  from 256 to 2048 or even 10240 generally results in a negligible increase in performance: typically less than 2%. This indicates that the failures accounting for the bulk of the performance difference are qualitatively different from the two modes observed in random DNA. We will describe these failures and introduce some methods to overcome them in subsequent chapters. The next chapter contains a detailed analysis of the computational work performed by the gapped SBH algorithm.



## Chapter 3

# Analysis of the Gapped-SBH Algorithm

This chapter contains an analysis of the behaviour and performance of the gapped-probe SBH algorithm. At the highest level, this can simply be the amount of computational work required to produce a complete sequence. However, the amount of work performed depends on the structure of path trees which are explored while resolving ambiguous extensions, which in turn depends on the probing pattern used.

When we discuss the behaviour of the SBH algorithm, we include all of the steps taken to completely reconstruct a target sequence from its spectrum. On the other hand, when we discuss the performance of the algorithm, we also mean to consider the method's effectiveness at reconstructing DNA fragments. A sequencing technique's usefulness can be measured in terms of the length of fragments which can be sequenced. Thus, we make the following definition:

**Definition 3.1.** *The performance metric used is the maximum length of an  $m$ -character maximum-entropy target sequence which can be reconstructed with some fixed confidence, denoted  $\epsilon$ .*

This definition allows gapped-SBH to be compared with other sequencing methods: traditional electrophoresis, and ungapped-SBH, for instance. Typically, we set  $\epsilon = 0.9$ , and this convention will be held throughout this chapter. Thus, the *performance* of the method can be stated in terms of the longest DNA fragment which has a 90% chance of being successfully reconstructed. Conversely, we could say that a

sequencing method is effective for fragments up to the length where there is a 10% chance of encountering a sequencing failure. An experimental way of obtaining this value is to run thousands of simulated sequencing trials—several hundred for each value of  $m$ —and count the number of successful and unsuccessful trials. This is the method which was used to produce Figure 2.2 at the end of the previous chapter. It shows that gapped-probe SBH with (4,4)-probes can be used to reconstruct randomly generated sequences of up to length  $m \approx 13000$ . Beyond that length, the probability of failure exceeds 10% and rapidly increases with  $m$ . However, a more mature understanding of the circumstances which cause sequencing failure can be reached only with a much more detailed analysis, which is contained in Chapter 4.

In this chapter, we explore in detail the path trees which are explored by the EXTEND algorithm as it attempts to resolve ambiguous extensions. Ambiguous extensions can occur very frequently during the reconstruction process, and virtually all of them may be resolved successfully. By understanding the nature of the path trees, we can develop a more mature understanding of the work performed by the SBH algorithm, and the causes of sequencing failure. We begin, in Section 3.1 with a general discussion of fooling probes, and the likelihood of finding one in a target sequence of a particular length.

### 3.1 Finding a Fooling Probe

First, we discuss the probability of finding a fooling probe in the spectrum of a target sequence.

When using an  $(s, r)$ -probing pattern to reconstruct a sequence of length  $m$ , there are at most  $m - s(r + 1)$  unique probes in the spectrum. Since  $m$  is typically much greater than the length of the probing pattern, this can be approximated simply as  $m$ . We want to define a parameter  $\alpha_{(k,m)}$  which represents the probability of finding a particular sequence of  $k$  symbols in an  $m$ -character random DNA string. Recall from Chapter 1 that by ‘random,’ we refer to a string which has been generated by an i.i.d. memoryless process, where each symbol is chosen independently with equal probability ( $\frac{1}{4}$ ). Note that wildcard characters in a sequence just introduce ‘gaps’ between the non-wildcard characters. The number of non-wildcard characters in the

sequence is the important parameter, so  $\alpha_{(k,m)}$  is actually the probability of finding a sequence with  $k$  specified characters in an  $m$ -character target sequence.

The probability of finding a  $k$ -character sequence at *one* particular position in the sequence is  $1/4^k$ . In a randomly-generated  $m$ -character string, the probability of finding the sequence at *any* position in the  $m$ -string is equal to  $1 - (1 - \frac{1}{4^k})^m$ . Note that

$$1 - (1 - \frac{1}{4^k})^{4^k \frac{m}{4^k}} \approx 1 - e^{-1 \cdot \frac{m}{4^k}} \quad (3.1)$$

This leads to the following definition:

**Definition 3.2.** *The parameter  $\alpha_{(k,m)}$  denotes the probability of finding a particular  $k$ -character subsequence in an  $m$ -character randomly generated DNA sequence and is defined as:*

$$\alpha_{(k,m)} = 1 - e^{-m/4^k}$$

**Example 3.1.** The probability that the 9-character sequence AGGATTTAC occurs somewhere within a particular 19000 character randomly generated DNA string is  $\alpha_{(9,19000)} = 1 - e^{-19000/4^9} \simeq 0.07$ . In this case,  $k = 9$  and  $m = 19000$ . ■

**Example 3.2.** The probability that the 7-character sequence A . . . G . . . C . . . TTAT (or any other sequence with 7 specified characters) occurs somewhere in a random target string of length 4000 is  $\alpha_{(7,4000)} = 1 - e^{-4000/4^7} \approx 0.22$ , where  $k = 7$  and  $m = 4000$ . ■

The probability of finding a particular fooling probe in the spectrum of a target sequence is, of course,  $\alpha_{(\kappa,m)}$ ; they are simply  $\kappa$ -character sequences. The value of  $\alpha_{(\kappa,m)}$  as a function of  $m$  for  $\kappa = 6, 7, 8, 9$  is shown in Figure 3.1. Virtually all of the simulations and analysis in this dissertation from this point on will deal with probes having  $\kappa = 8$ , which, in a sense, reflects the state-of-the-art of microarray technology.

## 3.2 Overlapping Probes

Typically, we make the assumption that all of the probes in a target sequence's spectrum are independent and disjoint. This is only a slight simplification, which will

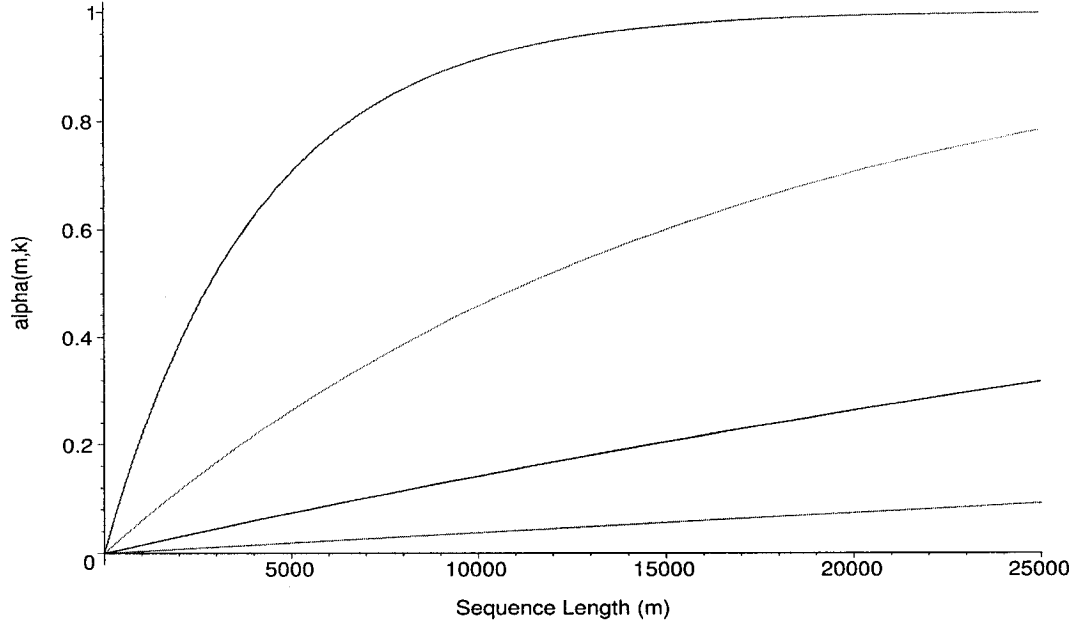


Figure 3.1:  $\alpha_{(\kappa,m)}$  as a function of sequence length ( $m$ ). The four curves correspond to  $\kappa = 6, 7, 8, 9$ , from top to bottom.

be dealt with in more detail below. Assuming that probes are independent, we can also examine the likelihood of finding duplicate probes within a target sequence, as well as fooling probes.

If a random  $k$ -character subsequence ( $b$ ) is chosen from a randomly generated  $m$ -character DNA string  $a$ , what is the probability of finding an identical  $k$ -character subsequence at some other point in  $a$ ? When  $m \gg k$ , there are  $m - k \approx m$  positions at which  $b$  can occur within  $a$ . If we know that it occurs once, at location  $i$ , then there are still  $m - k - 1 \approx m$  locations within the string  $a$  at which it may also occur. If we ignore overlapping subsequences, the probability of finding another instance of a  $k$ -character sequence in  $a$ , given that it occurs once in the sequence is approximately  $\alpha_{(k,m)}$ .

**Example 3.3.** Assume that the direct (3,3)-probe  $p$  (AGA . . C . . G . . T is located at position  $i$  in a 1500-character target sequence and therefore present in the spectrum. Given that  $p$  is present once in the sequence, the probability of finding  $p$  at another point in the sequence is also  $\alpha_{(6,1500)} \approx 0.22$ . ■

In practice, the probability of finding a second occurrence of a  $k$ -character sequence

$b$  within  $a$  is *at least*  $\alpha_{(k,m)}$ . While they are unlikely to occur in randomly generated sequences, periodic sequences of DNA significantly increase the chance of finding a second copy of a particular segment  $b$ . In such a case (i. e.,  $b = \text{ATG ATGATGATG}$ ), the probability of finding a second copy of  $b$  can be dramatically higher than  $\alpha_{(k,m)}$ .

Consider the pathological case where  $b = \text{AAAAAAAA}$  (eight ‘A’ characters in a row) occurs as a subsequence of a 8000-character target sequence at position  $i$ . This occurrence of  $b$  within the target sequence will be denoted  $b_i$ . The likelihood of finding another incidence of  $b$  within the target sequence is at least  $\alpha_{(8,8000)} = 1 - e^{-\frac{8000}{4^8}} \approx 0.11$ . However, if the character immediately to the left of  $b_i$  (position  $(i-1)$ ) is an ‘A,’ then the sequence  $b$  is also found at position  $(i-1)$  in the target sequence. Similarly, if the character immediately to the right of  $b_i$  (position  $(i+8)$ ) contains an ‘A,’ then the sequence  $b$  occurs at position  $(i+1)$  in the target sequence. Each of these events occurs with probability  $\frac{1}{4}$ , so there is a probability of  $(1 - \frac{1}{4})^2 \approx 0.56$  that at least one character adjacent to  $b_i$  is also an ‘A,’ which is much larger than  $\alpha_{(8,8000)}$ .

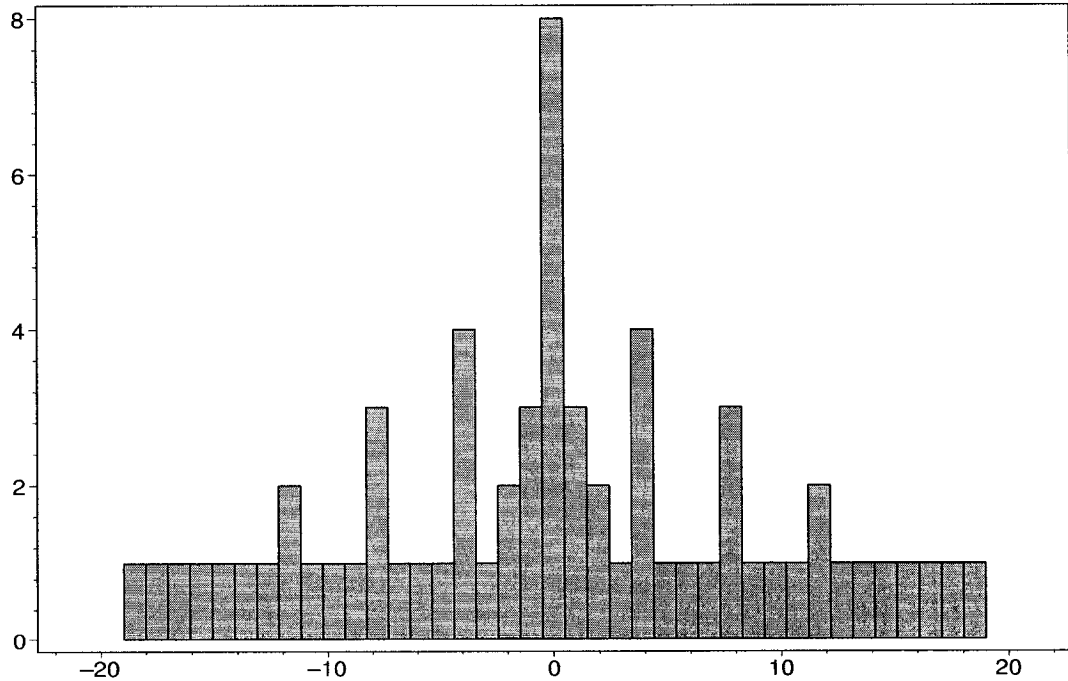


Figure 3.2: Autocorrelation function for (4,4)-probing pattern.

When considering subsequences with gaps, and specifically  $(s, r)$  gapped probing patterns, overlapping periodic subsequences are less of a concern. Take the 20-character (4,4)-direct probe  $p = \text{AAAA} \dots \text{A} \dots \text{A} \dots \text{A} \dots \text{A}$ . The autocorrelation function for (4,4)-probes is shown in Figure 3.2. If  $p$  occurs at position  $i$  in a target sequence, only 3 (out of a possible 7) characters overlap with the adjacent probes at position  $(i+1)$ , and 2 characters at  $(i+2)$ . The probes at positions  $(i+4)$  and  $(i-4)$ , have a 4-character overlap with the probe at  $i$ . At positions  $(i+1)$ ,  $(i-1)$ ,  $(i+8)$  and  $(i-8)$ , there is a 3-character overlap. At  $(i+2)$  and  $(i-2)$ , it is only 2 characters. Probes at any other location in the range  $[i-\lambda+1, i+\lambda-1]$  share only a single character overlap with the probe at  $i$ . The probability of finding another instance of the probe  $\text{AAAA} \dots \text{A} \dots \text{A} \dots \text{A} \dots \text{A}$  at one of the locations with more than a single overlapping character is higher than at other locations in the target sequence, but not nearly as great as for solid probes. The analysis of sequencing failures presented in Sections 4.2 and 4.3 takes overlapping fooling probes into account. However, we ignore the existence of fooling probes which overlap with the first  $\lambda$  characters following a branching: fooling probes are assumed to occur at locations in the target sequence such that they do not intersect at all with the first  $\lambda$ -characters of the correct path following an ambiguous extension.

From this point on, the discussion of sequences and subsequences will be concerned virtually entirely with probes having  $\kappa$  specified bases, and  $m$ -character target sequences. Thus, the parameter  $\alpha$ , without any subscript, will be used to denote the probability of finding a random  $\kappa$ -probe within an  $m$ -character target sequence. If the context is at all ambiguous, the full  $\alpha_{(k,m)}$  notation will still be used. We make the following slightly modified definition for  $\alpha$  with no subscript:

**Definition 3.3.** *The parameter  $\alpha$  denotes the probability of finding a particular probe with  $\kappa$  specified positions in an  $m$ -character randomly generated DNA sequence.  $\alpha$  is defined as:*

$$\alpha = 1 - e^{-m/4^\kappa}$$

With notational issues dealt with, we move on to the discussion of the work performed by the sequencing algorithm as the putative sequence is extended.

### 3.3 Tree Size and Work

In [PU00], Preparata & Upfal demonstrate an  $O(m)$  running time for the branching SBH reconstruction algorithm. While this bound holds for all probing patterns and different  $H$  and  $B$  bounds on the tree size, the specific pattern and bounds used can have a very significant effect on the constants affecting performance. The average size of the tree explored in the branching mode of EXTEND determines the constants for the  $O(m)$  execution time of the method.

The choice of probing pattern has the most significant effect on the size of path trees. Before we continue, it is important to note that the probing pattern is implicitly dependent on the sequencing direction. A ‘direct’ probe is effectively a ‘reverse’ probe if sequencing is proceeding from right to left. When we discuss probing patterns in this chapter, we assume that sequencing is proceeding from left to right.

With this in mind, the path trees constructed using direct and reverse (4,4)- and (5,3)- probing patterns differ in size by an order of magnitude or more. Reverse patterns have the advantage, producing smaller trees.

Before beginning a detailed analysis of the phenomenon, a quick overview is helpful. The most important difference between the direct and reverse  $(s, r)$ -probing patterns is the speed with which false paths may be eliminated. Consider that for any  $(s, r)$ -probing pattern, there are always  $\kappa$  probes which sample any position of the target sequence. This leaves  $(\lambda - \kappa)$  probes which contain the ambiguous position within their span, yet *do not* sample that position. It is a reasonable first approximation to say that only the sampling probes can eliminate a path from the tree. A query which is executed with such a probe will be called a *constraining* query. The remaining  $(\lambda - \kappa)$  probes cannot possibly eliminate a spurious path from the tree, but they can cause additional branches. The net result is that, for reverse probing patterns, the first  $(s - 1)$  shifts after an ambiguous extension can potentially eliminate a spurious path. On the other hand, using direct probes, the first  $(s - 1)$  shifts are guaranteed to extend both the correct and spurious paths. Moreover, additional branches in the path tree may be spawned by these non-constraining shifts, and if there are additional branches, both the correct *and* spurious paths will branch simultaneously, at least *doubling* the number of paths in the tree.

Consider the following two examples, comparing direct and reverse (3,3)-probes.

Each example follows a specific case of an ambiguous extension from the initial branch to the first shift after the branching which samples the ambiguous character.

**Example 3.4.** A 150-character sequence is being reconstructed with a reverse (3,2)-probing pattern. The putative sequence  $S_p$  is ...GCGGCATATGAGTT when an ambiguous extension occurs, with 2 feasible extensions. (Ambiguous extensions are expected with a probability of approximately  $\alpha \approx 0.136$ .) The two feasible-extension probes that match the query T...G...TT? are T...G...TT **C** and T...G...TT **T**, so two paths are spawned: GCGGCATATGAGTT **C** and GCGGCATATGAGTT **T**.

An attempt is made to extend each of these paths, using the queries A...A...T **T**? and A...A...T **C**?. There is guaranteed to be a match to the query corresponding to the correct path, whereas the query corresponding to the spurious path will produce a response only with probability  $\alpha \approx 0.136$ . If there is no response to the spurious-path query, then the initial ambiguity is resolved—whether or not there is a spurious match to the correct-path query.

If there *is* a match to the spurious-path query, then each path is extended by a single character, and two more queries are made. Again, there must be a match to the correct-path query, while the spurious-path query produces a response only with probability  $\alpha$ . ■

**Example 3.5.** Direct (3,2)-probes are being used to reconstruct a 200-character sequence. At the point of an ambiguous extension, the putative sequence is  $S_p =$  TTGCGCGATGCCGGC. The two probes matching the extension query ATG...G...? are ATG...G... **A** and ATG...G... **T**. These two probes spawn two paths: TTGCGCGATGCCGGC **A** and TTGCGCGATGCCGGC **T**.

The next extension query (for both the correct and spurious paths) is TGC...G...? There must be a probe in the spectrum that will extend the correct path, and the extension query is identical for both paths, so both paths are guaranteed to be extended. Let's assume that the correct extension probe is TGC...G... **A**. In addition to the guaranteed probe, there is also a chance that one of three spurious extensions will be found; in this case, each spurious extension has probability  $\alpha \approx 0.177$ . If such a probe is found (e.g. TGC...G... **T**), then it spawns another spurious branch from both the correct and spurious paths, resulting in four paths which must be extended further. The four paths, with  $S_p$  representing the putative sequence up to the point of



the ambiguous branching, are  $S_pAA$ ,  $S_pAG$ ,  $S_pTA$ , and  $S_pTG$ . Of course, if there were two or three spurious extension probes, then each path would spawn a corresponding number of branches, tripling or quadrupling the number of paths in the tree.

The same rules govern the next shift: all *four* paths share a single extension query (GCC . . C . . ?), and the probe which extends the correct path also extends the spurious paths. And again, if there is one or more spurious matches to the query, then *each* of the paths in the tree spawns additional branches, resulting in a corresponding  $n$ -fold increase in the the number of paths in the tree.

It is not until the third shift after the initial branching that an extension query samples the the ambiguous character. In this case, there are two extension queries: CCG . . **A** . . ? and CCG . . **T** . . ?. A single probe matching either query will extend exactly half of the paths in the tree. (Conversely, if there are no probes matching one of the queries, then half of the paths in the tree are eliminated.) If only one of these queries produces a response, then the ambiguous extension is resolved, since all of the paths beginning with the spurious character will be eliminated at once. If both queries produce responses from the spectrum, then extension of all paths continues. In this case, the ambiguous extension cannot be resolved until the next query which samples the branching position, requiring another  $s = 3$  shifts. ■

Note in Example 3.5 that there is only a probability of  $\alpha \approx 0.177$  chance that the spurious paths will be extended past the 3<sup>rd</sup> (or  $s^{\text{th}}$ ) character after the branch. This is identical to the probability that an ambiguous path will be extended by more than *one* character beyond the ambiguous extension for reverse probes. In each case, the first probe that samples the ambiguous character provides an identical chance of eliminating the false paths from the tree; the only difference is in the amount of work performed—calculated by the number of spurious paths that are extended, and by how many characters—before the ambiguous extension can be resolved.

Figure 3.3 shows a possible path tree constructed using (4,3)-direct probes. The initial ambiguous branching is at depth 1, and represented by the right-most bar in the image. The darker, hatched bars correspond to constraining queries during tree contruction, and the lighter unhatched bars to queries which can freely extend the spurious paths. Note that the deepest level in the tree is represented by the the left-most bar in the figure, and the first level of the tree (the initial branching) is at

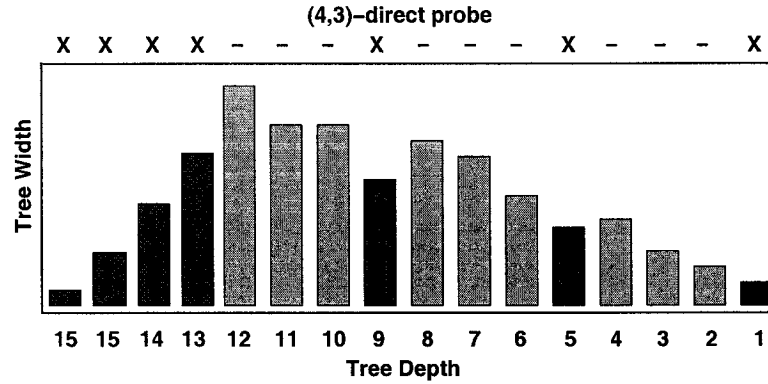


Figure 3.3: An example path tree constructed with a (4,3)-direct probing pattern. The height of the bars is merely a qualitative indication of the expected width of the path tree. Constraining queries (including the initial branching) are shown shaded and hatched.

the right.

In summary, an ambiguous extension during sequencing occurs at position  $t$  when there are two or more feasible-extension probes for the query at  $t$ . The initial fooling probes are the *first* probes sampling  $t$ , they intersect  $t$  in their right-most position. To a first approximation, opportunities to resolve the ambiguity (by eliminating the spurious path) occur when one of the probes used to extend the branching paths also samples the initial ambiguous character. Such extension queries can be called *constraints* on the branching extend process. The sooner the constraints occur after an ambiguous extension, the smaller the trees are likely to be. Thus, reverse probes, which follow the ambiguous extension with an additional  $(s - 1)$  constraining queries, tend to allow faster sequence reconstruction, due to their smaller trees.

### 3.3.1 Branching Events

Before we begin a discussion of the exact nature of the path trees constructed using direct and reverse probing patterns, it is useful to consider the different branching events which may be encountered while constructing such trees. Branching events, are classified by the number of descendants expected at depth  $i$  in the tree for *each* path at depth  $i - 1$ . There are five basic events, corresponding to different conditions occurring during EXTEND's breadth-first path extension. As we will see in the following two

sections, these events are often mixed; some proportion of the paths at depth  $i$  will be described by one branching event, while the remaining paths are described by another. However, by introducing the basic branching events here, the analysis of the trees themselves can be made much more elegant.

For each event  $a \dots e$ , we define a corresponding parameter  $\beta_a \dots \beta_e$  called the *multiplier* which denotes the number of descendants expected for each path. The parameter  $\beta$  essentially describes how much *larger* or *smaller* we expect the path tree to be at depth  $i$  than it was at depth  $i - 1$  of the tree.

**Event  $a$  - Guaranteed Extension** When the EXTEND algorithm makes its initial call to QUERY to determine the next character in the sequence, we know (unless sequencing has completed) that there is at least one match in the spectrum, and there may be as many as three feasible-extension fooling probes.

Furthermore, when the EXTEND algorithm is attempting to resolve an ambiguous extension by conducting the breadth-first expansion of the path tree, the spurious paths in the tree *may* fall into this category. If the extension-query for a spurious path  $p$  matches the extension-query for the correct path  $c$ , then the same probes which extend  $c$  also extend  $p$ . Since there must be an extension to  $c$ ,  $p$  must be extended as well.

In both cases, a path is guaranteed to have at least one descendant. Each of the three potential fooling probes is present in the spectrum with probability  $\alpha$ . Thus, we define the multiplier for event  $a$  to be

$$\beta_a = 1 + 3\alpha$$

**Event  $b$  - Probabilistic Extension** When a spurious path in the path tree *does* sample the initial branching position, it must be confirmed by a fooling probe or be eliminated. There are potentially up to four probes in the spectrum which will match the query, each of which is present with probability  $\alpha$ . Therefore we define the multiplier event  $b$  to be

$$\beta_b = 4\alpha$$

**Event  $c$  - Branches from Correct Path** The initial call to QUERY made by the EXTEND algorithm is guaranteed to produce at least one response: the probe

corresponding to the correct path. If we wish to count only the number of spurious responses to the query, we count only the three potential fooling probes. Each of these is present with probability  $\alpha$ , so the multiplier for event  $c$  is

$$\beta_c = 3\alpha$$

**Event  $d$  - Branches from Correct Path, Given that a Branch Occurred**

Event  $c$  predicts the average number of fooling branches from the correct path, including the possibility that there are *no* spurious branches. This event instead covers the case where there may have been up to 3 spurious branches, and we have observed at least one. This branching event applies to the initial branch from the correct sequence: we want to predict the average number of spurious extensions to the sequence, given that the extension was ambiguous.

There are four possible outcomes to a guaranteed-extension query, resulting in 0, 1, 2 and 3 fooling probes, respectively. The probability of each event is given in the following table:

# of fooling probes	Probability
0	$(1 - \alpha)^3$
1	$3 \cdot \alpha \cdot (1 - \alpha)^2$
2	$3 \cdot \alpha^2 \cdot (1 - \alpha)$
3	$\alpha^3$

The event “there is at least one fooling probe which matches a guaranteed-extension query” has probability  $1 - (1 - \alpha)^3$ . There are 3 fooling probes which may potentially match any query, each of which is present with probability  $\alpha$ . Thus, the expression  $\frac{3\alpha}{1 - (1 - \alpha)^3}$  yields the expected number of fooling probes which match a query, given that at least one fooling probe was found, which is the multiplier for event  $d$ :

$$\beta_d = \frac{3\alpha}{1 - (1 - \alpha)^3}$$

**Event  $e$  - Probabilistic Extension, Given that a Fooling Probe was Found**

Event  $b$  handles the case where up to four fooling probes may match a query, and we don’t know whether or not there will be *any* matches. This event ( $e$ )

handles the case where there may have been up to four fooling probes matching a query (none of which were guaranteed in the spectrum), and we have observed at least one matching probe. This branching event applies during the breadth-first expansion of the path tree, at depths where the corresponding query was a constraint. When we are estimating the size of the tree at depth  $i$ , we know that the tree was extended to at least depth  $i$ , so at least one spurious path must have been extended by a fooling probe.

Similarly to event  $d$ , we need to exclude the event that 0 fooling probes were found. The event “there is at least one fooling probe found which matches a probabilistic extension query” has probability  $1 - (1 - \alpha)^4$ . There are potentially 4 matching probes, each of which are present with probability  $\alpha$ , so the multiplier for event  $e$  is

$$\beta_e = \frac{4\alpha}{1 - (1 - \alpha)^4}$$

When the EXTEND algorithm is conducting the breadth-first expansion of the path tree, the expansion of the tree is always governed by some combination of the branching events listed above. To predict the size of the path tree at each level, all that needs to be done is to determine which branching event (or mix of events) applies at each level.

### 3.3.2 Reverse Probes

Recall from Definition 1.10 that reverse probes have the pattern  $(10^{s-1})^r 1^s$ , where ‘1’ represents a natural base, and ‘0’ represents a universal base. For instance, a (3,3)-reverse probe has the form 100100100111, or N..N..N..NNN using ‘N’ and ‘.’ in place of ‘1’ and ‘0’. In this section, we wish to describe the trees produced while resolving ambiguous extensions using reverse  $(s, r)$ -probing patterns.

Specifically, we want to count the total number of paths, correct and spurious, which exist at any depth in the path tree. This is not a trivial problem, so we break it down into components. The first step is to count only the paths which contain a spurious character at the position of the initial branching (we ignore the correct path, and assume that there is only a single spurious extension at depth 1). This is a fairly straightforward problem, requiring the following assumptions:

1. **Independence.** We assume (as before) that individual fooling probes are completely independent. (This hypothesis is not rigorous, but acceptable with good confidence.)
2. **Single spurious character.** We assume that the initial spurious branching is always a two-character choice between the correct character and a *single* spurious character. (e.g. A-T, G-T) We also count only the spurious paths descended from the spurious character; the correct path is not counted. Thus, the width of the path tree at depth 1 is always 1.
3. **Only one root.** We only want to count the number of paths originating at the initial spurious character. Subsequent branches from the correct path (which ultimately add to the overall size of the path tree) are ignored.
4. **Guaranteed extension.** We assume that spurious paths may only be eliminated by queries which sample the initial ambiguous position—such queries are *constraints* on the further branching of the path tree. By this assumption, non-constraining queries are guaranteed to extend all paths; such queries fall into branching event  $a$ .
5. **Indefinite extension.** We ignore the possibility of the entire tree being eliminated (by the elimination of all paths in the tree). In reality, if the initial ambiguous character is resolved, then the tree will no longer be extended. Moreover, an actual tree cannot contain a fractional number of elements at any level: there must always be a positive, integral number of paths in the tree. We ignore these two constraints, and allow trees which contain fractions of paths, even if the fraction is less than 1.

Finally, we denote the root (depth 0) of the path tree as the last (unambiguous) character of the putative sequence, so the initial branching position corresponds to depth 1 in the tree. Given these assumptions, the width of the path tree at depth  $i$  is given by the following theorem:

**Theorem 3.1.** *Let the probing pattern be denoted  $P$ , and the  $i^{\text{th}}$  position of the pattern be  $P_i$ . Let  $c$  denote the number of natural bases in the probing pattern from  $P_{\lambda-i} \dots P_{\lambda-1}$ .*

Denoting the expected width of the path tree at depth  $i$  by  $\zeta_i$ , we have

$$\zeta_i = \beta_a^{i-c-1} \cdot \beta_b^c \quad (3.2)$$

*Proof.* Since the initial ambiguous position is level 1 of the path tree, then the width  $\zeta_i$  of the tree (by Assumption 2 above) is always 1. At depth  $i$ , the  $(\lambda - i)^{\text{th}}$  character of the probing pattern coincides with the initial branching position. Thus, the expected width  $\zeta_i$  of the tree at depth  $i$  is given by the following recursion:

$$\zeta_i = \begin{cases} 1 & \text{if } i = 1, \\ \beta_b \cdot \zeta_{i-1} & \text{if } i > 1 \text{ and } P_{\lambda-i} \text{ is a natural base,} \\ \beta_a \cdot \zeta_{i-1} & \text{if } i > 1 \text{ and } P_{\lambda-i} \text{ is a universal base.} \end{cases} \quad (3.3)$$


At depth  $i$  of the tree, position  $\lambda - i$  of the probing pattern aligns with the branching position<sup>1</sup>.

When a natural base in the probing pattern aligns with the initial ambiguous character, then every spurious path in the tree must be confirmed by a fooling probe. If there is only a single path in the tree at depth  $i - 1$ , there are 4 fooling probes, any one of which may confirm it at depth  $i$ . Each of these probes may be present with probability  $\alpha$  in the spectrum. Probabilistically, the single path spawns  $4\alpha = \beta_b$  children, so there are an expected  $\beta_b$  paths in the tree at depth  $i$  if  $\zeta_{i-1} = 1$ .

The same reasoning holds true for *each* path in the tree at depth  $i - 1$ . Every path in the tree spawns  $\beta_b$  children, so the size of the tree at depth  $i$  is  $\zeta_i = \zeta_{i-1} \cdot \beta_b$ , if  $P_{\lambda-i}$  is a natural base.

Now we consider the case where a universal base in the probing pattern aligns with the initial ambiguous character. We assume that because such queries do not sample the spurious character, they cannot eliminate false paths from the tree. A spurious path in the tree at depth  $i - 1$  is guaranteed to have at least one child at depth  $i$ . Furthermore, there are three fooling probes which could spawn additional branches at depth  $i$ . Each of these probes is present with probability  $\alpha$ . Thus, a single spurious path at depth  $i - 1$  spawns  $1 + 3\alpha = \beta_a$  children at depth  $i$ . Since this holds for all paths in the tree, the size of the tree at depth  $i$  is  $\zeta_i = \zeta_{i-1} \cdot \beta_a$ , if  $P_{\lambda-i}$  is a universal base.

---

<sup>1</sup>For example, using (3,3)-reverse probes (N . . N . . N . . NNN), with  $\lambda = 12$ , at depth 3 of the tree, the  $12 - 3 = 9^{\text{th}}$  character of the pattern (N . . N . .  NNN) aligns with the branching.

Note that the  $\beta$ -multipliers at each depth are independent. The width of the tree at any depth  $i$  is simply the product of the multipliers at depths  $1, \dots, i$ . The multiplier at depth  $i$  depends only on whether  $P_{\lambda-i}$  is a universal base. If there are  $c$  natural bases in the  $P_{\lambda-i} \dots P_{\lambda-1}$ , then there are  $c-i$  universal bases in the same span. Each universal base contributes a factor of  $\beta_a$ , and each natural base contributes a factor of  $\beta_b$  to the size of the tree, thus yielding Equation 3.2.  $\square$

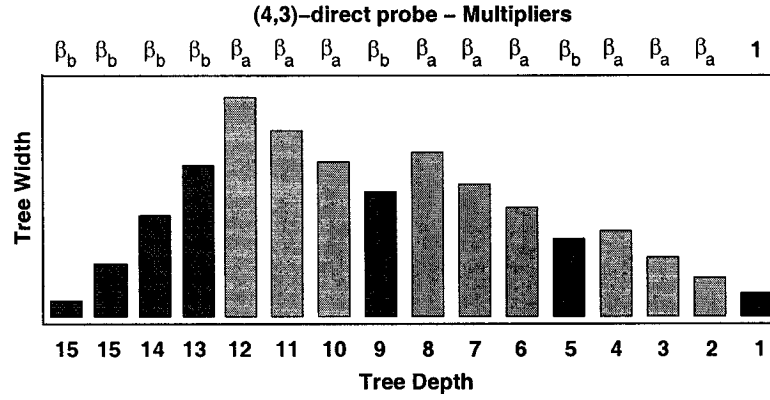


Figure 3.4: Probabilistic path tree showing independent multipliers ( $\beta_a, \beta_b$ ) at each depth for a (4,3)-direct probing pattern. Constraining queries (including the initial branching) are shown shaded and hatched.

Figure 3.4 illustrates the simple model of path tree expansion we have just explored. Level 1 in the tree contains a single element, and at each level  $i$ , the size of the tree is multiplied by  $\beta_a$  or  $\beta_b$ , depending on whether or not  $P_{\lambda-i}$  is a constraint or not.

Unfortunately, our assumptions oversimplify things. We will consider each of the assumptions in turn, and describe how the analysis changes as a result of eliminating the assumption.

The first assumption—the *independence* of fooling probes—does not significantly affect the analysis of tree size, since fooling probes are very nearly independent.

The second assumption—that there is a *single spurious character*—is simple to eliminate. We want to account for the possibility of three- and four-way branches (e.g. A-G-T or A-C-G-T) in the sequence. This is fairly simple, as it only requires modifying the value of  $\zeta_1$ . Ambiguous extensions occur whenever there are one or



more fooling probes at a particular position in the sequence. Since there are always three potential fooling probes, and each fooling probe is present with probability  $\alpha$ , we expect to find an average of  $3 \cdot \alpha$  fooling probes each time the EXTEND algorithm queries the spectrum. This corresponds to branching event  $c$ . However, we are considering only cases where we *know* that there was at least one fooling probe present. Thus branching event  $d$  applies, and the expected size of the tree at depth 1 is

$$\zeta_1 = \beta_d = \frac{3\alpha}{1 - (1 - \alpha)^3}$$

Assumption 3—that there is *only one root*—can also be eliminated in a straightforward manner. So far we have considered only a single path tree originating at a single ambiguous position ( $i$ ), which will be called the *primary* tree. However, the correct sequence must also be extended along with the tree of spurious paths. At every position  $j > i$ , the correct path may produce a new spurious path tree, independent of the first. We will call these trees *secondary* path trees, and denote the width of a secondary tree at depth  $i$  as  $\zeta'_i$ . In the secondary trees, there are three potential fooling probes at depth 1. Each of these three fooling probes is present with probability  $\alpha$ , and none of them are guaranteed to be present, so  $\zeta'_1 = 3\alpha = \beta_c$ . At depths  $i > 1$ , the width of the tree can be calculated similarly to the primary tree. Figure 3.5 shows an example of how the primary and secondary trees contribute to the total number of spurious paths. It is important to remark that all such diagrams are meant only to illustrate the width of trees *qualitatively*: the height of the bars do not represent expected width, or even the relative width at different depths. Only a general comparison (*bigger* or *smaller*) between consecutive levels in the tree is intended.

When the EXTEND algorithm is resolving an ambiguous extension, the total number of spurious paths at depth  $i$  includes the correct path, the paths in primary tree, and all of the paths in the  $i - 1$  secondary trees. Thus the total width of the complete path tree at depth  $i$  can be denoted as  $\omega_i$ , where

$$\omega_i = 1 + \zeta_i + \sum_{j=1}^{i-1} \zeta'_j$$

The fourth and fifth assumptions are somewhat more complex to eliminate. We want to consider more precisely when spurious paths in the primary and secondary

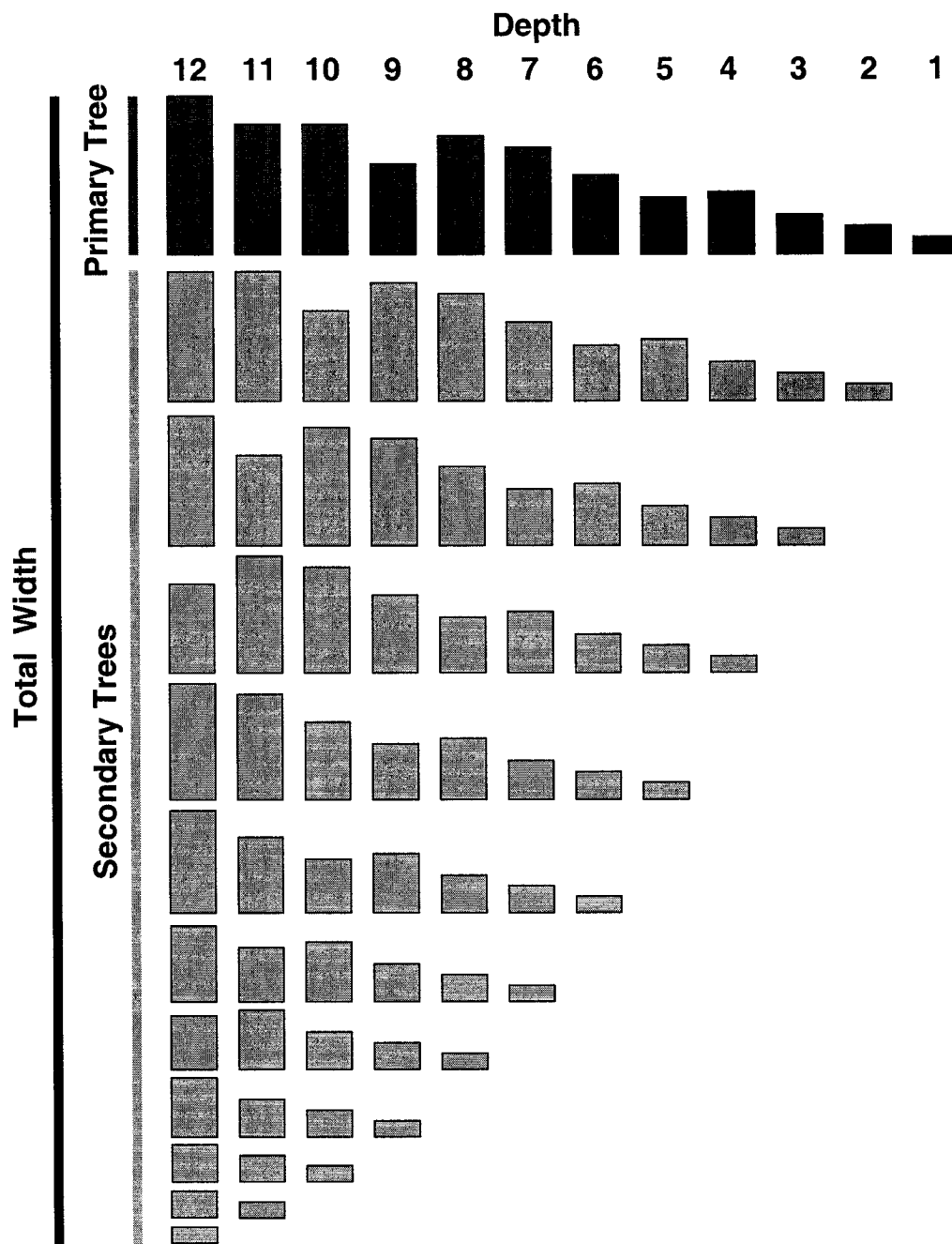


Figure 3.5: Primary and secondary path trees must be added to calculate the total number of paths. Note that only the initial (right-most) component of the initial tree is guaranteed to exist; the rest are all present only with some probability  $< 1$ . The total width of the path tree at depth  $i$  is equal to the width of the primary tree at depth  $i$ , plus the  $i - 1$  secondary trees which may have been spawned up to that depth.

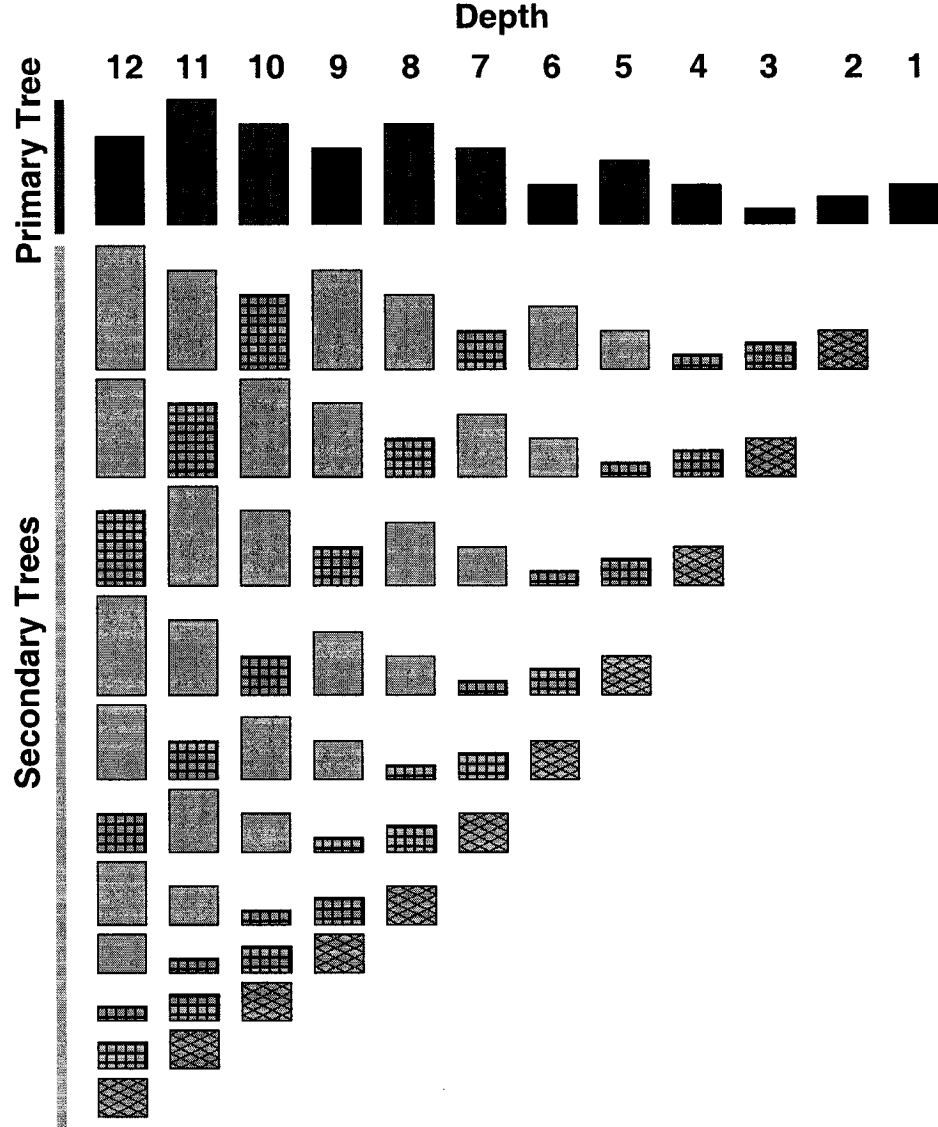


Figure 3.6: Primary and secondary trees for (3,3)-reverse probes. The initial branch in the primary tree is black, and is the only component of the tree which is guaranteed to exist. The initial branch in each secondary tree is diagonally cross-hatched. The queries which may resolve the initial ambiguity in the primary tree are darkly shaded, and the constraining queries in the secondary trees are shown hatched.

trees may be freely extended, and when they may be eliminated. Figure 3.6 shows the composition of a path tree to depth 12 using (3,3)-reverse probes. The differences between constraining and unconstraining queries in the primary and secondary trees—four different classes of queries—can all be considered separately. Note that at any level in the tree, several different branching events may apply.

First, we tackle the fourth assumption: that queries which do not sample the initial branching position allow the *guaranteed extension* of false paths.

False paths are freely extended if and only if there are *no* sampled characters in the false path which disagree with the correct path. Constraining queries—queries which sample the initial branching—eliminate false paths if they are not confirmed by a fooling probe. However, a disagreement between the correct path and the spurious path at *any* sampled position must be confirmed by a fooling probe.

Consider a (4,3)-reverse probing pattern (N...N...N...NNNN). The first  $s - 1 = 3$  queries after an initial branching are constraining queries; the fourth (N...N...N...NNNN) is not. If all of the false paths agree with the correct path in each of the three characters subsequent to the initial branching, then they are all guaranteed to be extended by this query. On the other hand, if any false path differs from the correct path at any of the three positions subsequent to the initial branching, that path may be eliminated. Put differently, the  $s$ th query after the branching only allows the free extension of false paths that agree with the correct path over the previous  $s - 1$  characters. There are  $4^{s-1}$  possible combinations of  $s - 1$  characters; of these, only one combination agrees with the correct path. For (4,3)-reverse probes, the probability of such an event is approximately  $1/4^3 = 1/64$ . Consider also the following specific example:

**Example 3.6.** The following path tree originates from a 3-way branching while reconstructing a sequence using (4,3)-reverse probes. The path tree has been expanded to depth 4, with the initial branch being depth 1.

Depth $i$	0	1	2	3	4	5
...	G	G	A	A	T	A
	C	G	A	G	T	G
	[C]	C	(T)	G	A	G
	[T]	C	(A)	G		
	[G]	C	T	G		

The first row shows the correct sequence; the second and third rows contain false

paths originating from the initial ambiguous character. The 3-way branching is indicated in square brackets:  $[C][T][G]$ . Note that there is a second mismatch between the correct (top) path and the false path on the second row. It is shown in parentheses, at depth 3:  $(T)(A)$ . The false path on the third row agrees with the correct path at that location.

Using (4,3)-reverse probes, the queries which will extend each of the three paths at depth 5 are:

<i>Path</i>	
1	G . . . T . . . A . . [.] C (T) G ?
2	G . . . T . . . A . . [.] C (A) G ?
3	G . . . T . . . A . . [.] C (T) G ?

Note that the query strings for the first (correct) path and the third (spurious) path are the same. Since there must be a probe which extends the correct sequence, path 3 is guaranteed to be extended as well. However, the second path must be supported by a fooling probe; if none is found, then it is eliminated even though the query string did not sample the initial ambiguous position. ■

In general,  $1/4^{s-1}$  of all spurious paths will be guaranteed extension at depth  $s$ . The same proportion of paths ( $1/4^{s-1}$ ) will be automatically extended by the next two queries:  $(N \dots N \dots N \dots \text{NNNN})$  and  $(N \dots N \dots N \dots \text{NNNN})$ . If a path does not agree with the actual sequence at all of the non-free positions in the query  $(N \dots N \dots N \dots \text{NNNN}?)$ , then that path must be supported by a fooling probe, or be eliminated. In general, if a query contains  $n$  natural bases to the right of the initial ambiguous position, then there must be  $n - 1$  matches between the false and correct paths to guarantee free extension of a path. For reverse  $(s, r)$ -probing patterns, the number of constrained positions can be calculated as a function of the depth in the tree (recall that the initial branching position is depth 1). We denote the number of constrained positions at depth  $i$  as  $n_i$ , calculated as follows:

$$n_i = \begin{cases} i - 1 & \text{if } i \leq s, \\ s + \lfloor \frac{i}{s} \rfloor - 2 & \text{if } s(r + 1) > i > s, \\ s + r - 1 & \text{if } i > s(r + 1). \end{cases} \quad (3.4)$$

Spurious paths that agree with the correct path in all of their constrained positions are guaranteed to be extended, and thus fall into branching event  $a$ . The remaining paths must be confirmed by fooling probes, and thus fall into branching event  $b$ . Using Equation 3.4, we can calculate the size of the tree at depth  $i$ , when  $\lambda - i$  is a universal base, as

$$\zeta_i = \zeta_{i-1} \cdot \left( \frac{1}{n_i^4} \right) \cdot \beta_a + \zeta_{i-1} \cdot \left( \frac{\zeta_{i-1} - 1}{n_i^4} \right) \cdot \beta_b$$

Finally, we consider the fifth assumption, which allows *indefinite extension* of all trees. In practice, if the primary tree is completely eliminated, then the extension of the entire path tree stops. There may still be spurious paths in secondary trees (which diverge from the correct sequence at a point  $j > i$ ), but as soon as the initial ambiguity is resolved, the tree expansion stops. To calculate the expected size of the path tree, we must take into account tree elimination: the average size of a tree at depth  $i$  is calculated by including only trees which reached at least depth  $i$ , and thus had a non-zero width at that depth. If a tree is eliminated at a depth  $< i$ , it does not contribute to the calculation of tree width at depth  $i$ .

The rules governing the extension of paths when  $P_{\lambda-i}$  is a natural base in the primary tree are subtle. When a query samples the initial ambiguity, all spurious paths in the primary branching must be confirmed by fooling probes, or be eliminated. However, consider the case where the query at depth  $i - 1$  is a constraining query. Since we consider, at depth  $i$ , only those trees which *were* expanded to at least depth  $i$ , we know that that at least *one* of the paths in the primary tree at depth  $i - 1$  *was* confirmed by a fooling probe at depth  $i$ . That path, for which there was at least one fooling probe, is governed by branching event  $e$ . Thus, the path at depth  $i - 1$  which is known to have been extended has an expected  $\beta_e = \frac{4\alpha}{1-(1-\alpha)^4}$  children at depth  $i$ . The remaining paths, falling into branching event  $b$ , each produce  $\beta_b = 4\alpha$  descendants. Thus, if position  $\lambda - i$  is a natural base in the probing pattern, we have

$$\zeta_i = \left( \frac{1}{\zeta_{i-1}} \right) \cdot \beta_e + \left( \frac{\zeta_{i-1} - 1}{\zeta_{i-1}} \right) \cdot \beta_b$$

We can now calculate the size of the primary tree more quite accurately:

$$\zeta_i = \begin{cases} \beta_d & \text{if } i = 1, \\ \left(\frac{1}{\zeta_{i-1}}\right) \cdot \beta_e + \left(\frac{\zeta_{i-1}-1}{\zeta_{i-1}}\right) \cdot \beta_b & \text{if } i > 1 \text{ and } P_{\lambda-i} \text{ is a natural base,} \\ \zeta_{i-1} \cdot \left(\frac{1}{n_i^4}\right) \cdot \beta_a + \zeta_{i-1} \cdot \left(\frac{\zeta_{i-1}-1}{n_i^4}\right) \cdot \beta_b & \text{if } i > 1 \text{ and } P_{\lambda-i} \text{ is a universal base.} \end{cases} \quad (3.5)$$

The secondary path trees may be eliminated without affecting the overall expansion of the tree as a whole, so when  $P_{\lambda-i}$  is a natural base, *all* paths must be confirmed by fooling probes—none are guaranteed to be extended. The size of the secondary trees can be expressed by the following recursion, where the case for natural bases is the same as in the original Equation 3.3:

$$\zeta'_i = \begin{cases} \beta_c & \text{if } i = 1, \\ \beta_b \cdot \zeta'_{i-1} & \text{if } i > 1 \text{ and } P_{\lambda-i} \text{ is a natural base,} \\ \zeta'_{i-1} \cdot \left(\frac{1}{n_i^4}\right) \cdot \beta_a + \zeta'_{i-1} \cdot \left(\frac{\zeta'_{i-1}-1}{n_i^4}\right) \cdot \beta_b & \text{if } i > 1 \text{ and } P_{\lambda-i} \text{ is a universal base.} \end{cases} \quad (3.6)$$

Unfortunately, due to the complexity of path extension when constraining queries occur, there does not appear to be a simple closed-form equation to express the width of the primary path tree. Instead, the recurrences given in Equations 3.5 and 3.6 must be used.

Finally, recall that the sum of all of the primary and secondary trees, at depth  $i$  is

$$\omega_i = 1 + \zeta_i + \sum_{j=1}^{i-1} \zeta'_j$$

Using this recursive calculation, the size of trees at depth  $i$  can be computationally predicted quite accurately. Figure 3.7 shows the predicted tree size as well as the tree sizes observed during three sequencing attempts, for fragment sizes of  $m = 15000$ . There is a very good match between the predicted and observed values.

### 3.3.3 Direct Probes

Recall from Definition 1.10 that direct  $(s, r)$ -probes to have the form  $1^s(0^{s-1}1)^r$ , with ‘1’ representing a natural base, and ‘0’ a universal base. For example, 111001001001

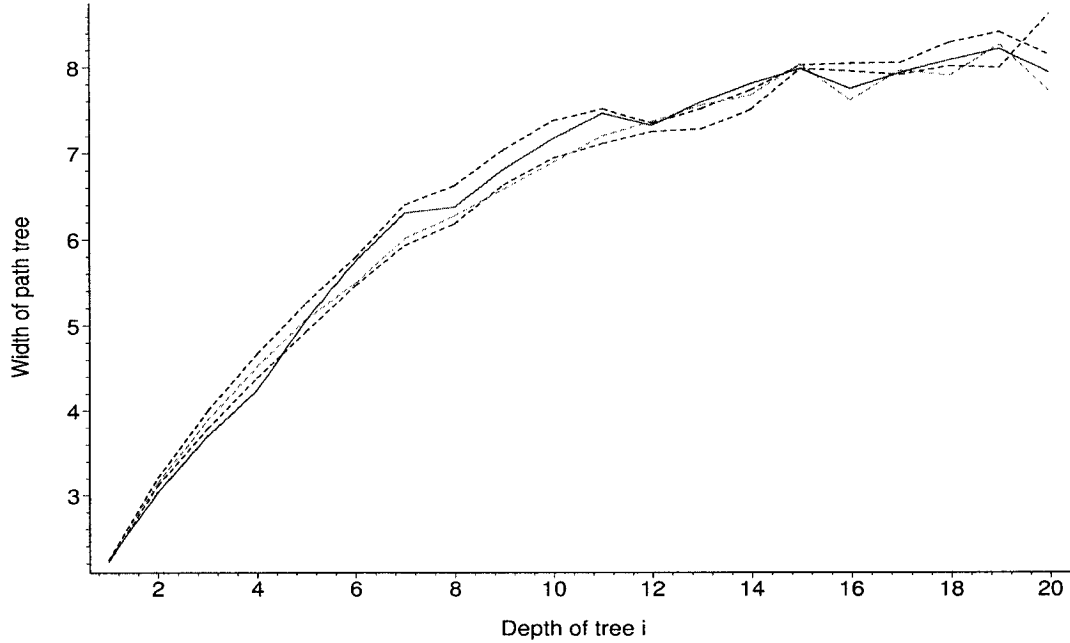


Figure 3.7: Predicted vs. observed tree widths for (4,4)-reverse probes at  $m = 15000$ . The solid line is the predicted width; the dotted lines represent three separate observations.

is a (3,3)-direct probe, which we represent as  $NNN \dots N \dots N \dots N$  using ‘N’ and ‘.’ in place of ‘1’ and ‘0.’

Direct probes have the disadvantage of allowing the *guaranteed extension* of spurious paths for  $(s - 1)$  characters after a branching. Each of these  $(s - 1)$  queries may also produce further branches in the path tree, both from the correct and spurious paths. While the same general reasoning leads to an accurate prediction of the path tree size for probes, the structure of the probe requires a more precise calculation: the approximation used for reverse probing patterns is too coarse to be used here.

Figure 3.8 shows the qualitative composition of the path tree for (3,3)-direct probes (compare with Figure 3.6 for the (3,3)-reverse pattern). Recall that these diagrams indicate only qualitative tree size, not specific expected tree sizes. With this in mind, note that at depth 2 and 3, the size of the primary tree is expected to increase—at these depths, a decrease in tree size is expected for the reverse probing pattern.

We will begin by looking at a single example of an ambiguous branch using (3,2)-direct probes.



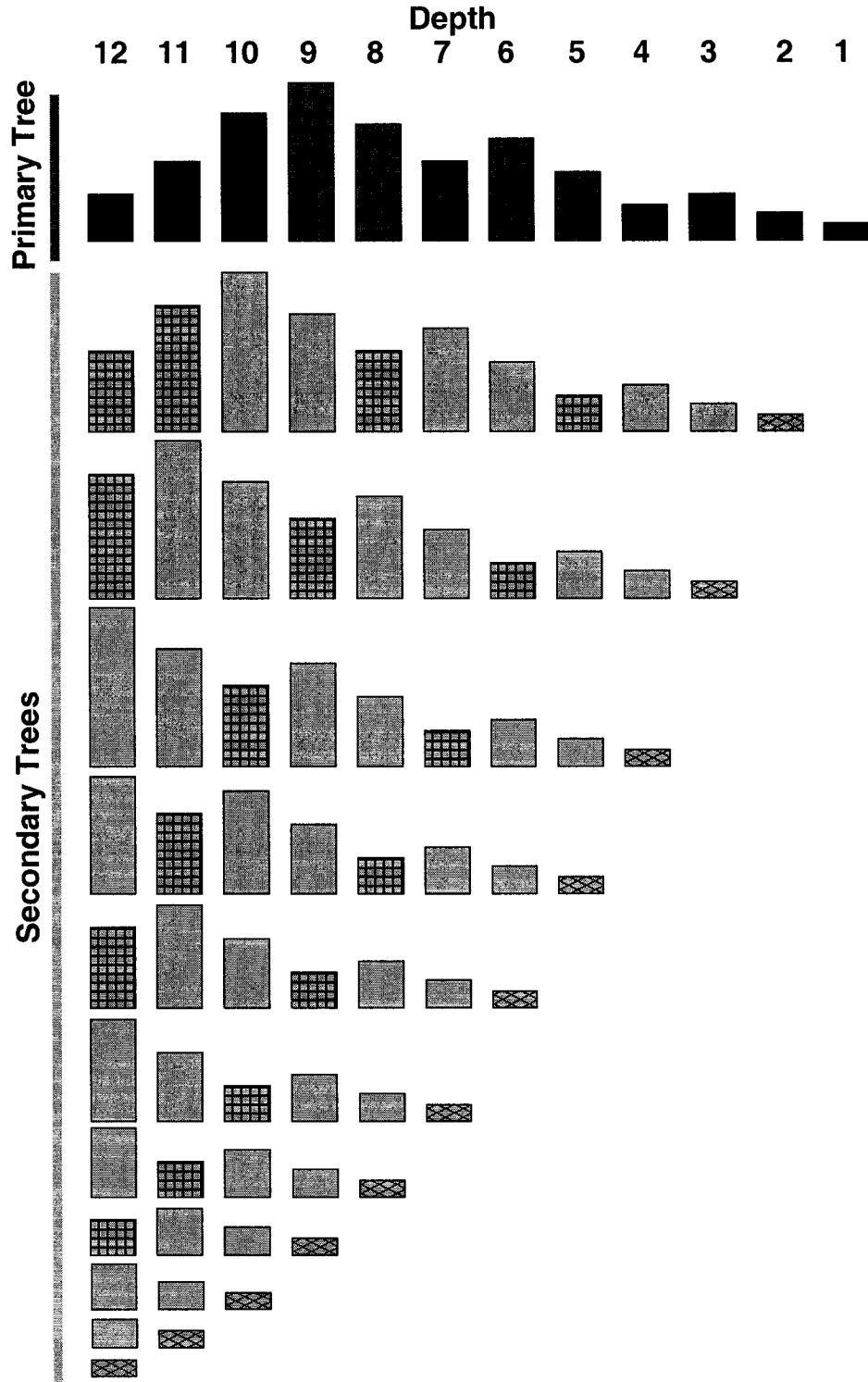


Figure 3.8: Primary and secondary trees for (3,3)-direct probes. The initial branch in each tree is diagonally cross-hatched, with the guaranteed initial branch in the primary tree shaded solid black. The queries which may resolve the initial ambiguity in the primary tree are darkly shaded, and the constraining queries in the secondary trees are shown hatched.

**Example 3.7.** In this example, the initial ambiguous branch is shown in square brackets:  $[C][G]$ . The putative sequence is to the left of the square brackets, and a secondary branching (which occurs in both the correct and spurious paths) is shown in parentheses:  $(A)(G)$ . There are three spurious paths:  $a$ ,  $b$ , and  $c$ , and 8 relevant probes which support these paths, numbered  $1 \dots 8$ . Fooling probes are labeled  $fp$  in the last column.

<i>Path Tree</i>																
Depth $i$										1	2	3	4	5	6	8
...	G	A	G	C	T	A	A	G	A	[C]	T	(A)	C	T	A	G ...
$a$												(G)	C	T	A	G ...
$b$										[G]	T	(A)	C	T	A	G ...
$c$												(G)	C	T	A	G ...
<i>Probes</i>																
1		A	G	C	.	.	A	.	.	[C]						
2		A	G	C	.	.	A	.	.	[G]						$fp$
3				C	T	A	.	.	A	.	.	(A)				
4				C	T	A	.	.	A	.	.	(G)				$fp$
5					T	A	A	.	.	[C]	.	.	C			
6					T	A	A	.	.	[G]	.	.	C			$fp$
7							A	G	A	.	.	(A)	.	.	A	
8							A	G	A	.	.	(G)	.	.	?	$fp$

The correct path is the topmost path, containing the characters  $[C]$  and  $(A)$  at the primary and secondary branchings, respectively. Thus, probes 1, 3, 5 and 7 are guaranteed in the spectrum. Probes 2, 4 and 6 all contain one spurious character: they are expected to be present in the spectrum with probability  $\alpha$ . And there are four probes of the form of probe 8; each of the four has probability  $\alpha$  of being in the spectrum.

The primary tree originates at position 1 with the initial ambiguity  $[C][G]$ . Path  $b$  is guaranteed to be extended by  $s - 1 = 2$  characters beyond the branching, since the next two probes do not sample the initial ambiguous position. Probe 4 causes a branching event in the path tree at depth 3. Since probe 4 is a feasible-extension probe for both the correct and spurious paths, it initiates a branch from both the correct and spurious paths, creating path  $c$  in the primary tree and path  $a$  which initiates a secondary tree. This event doubles the total size of the tree to 4 paths.

At depth 4, the first constraining query occurs. At this point, both of the paths in the primary tree ( $b$  and  $c$ ) could be eliminated, but the presence of probe 6 allows them to be extended further. Notice that probe 6 allows the extension of *both* of the false paths in the primary tree, since these two false paths are identical over all sampled positions. At the same depth, probe 5 allows the secondary false path ( $a$ ) to be extended.

The initial branching can only be resolved (by eliminating all of the paths containing the initial spurious character) by queries which sample the initial branching. However, it is possible to eliminate *some* of the spurious paths even when a query does not sample the initial ambiguous character. At depth 5, all of the characters sampled agree with the correct sequence, so no paths may be eliminated. At depth 6, the situation is more complicated.

The query at depth 6 samples the secondary branching. Probe 7 is guaranteed in the spectrum, and allows path  $b$  to be freely extended. The other two spurious paths must be confirmed by a probe of the type of probe 8. There are potentially four fooling probes which can confirm these paths, each present with probability  $\alpha$ : if none is found, then paths  $a$  and  $c$ , which differ from the correct sequence at the secondary branching, will be eliminated.

We are most interested in the expected size of the primary tree at depth 6, so let's look at the two paths in the primary tree:  $b$  and  $c$ . Path  $b$ —containing the correct character [A] at the secondary branching—is guaranteed to be extended to depth 6, and *may* produce further branches if there are fooling probes in the spectrum. This path falls into branching event  $a$ , and has an expected  $\beta_a = 1 + 3\alpha$  children at depth 6. Path  $c$ —containing the spurious character [G] at the secondary branching—may be eliminated. It falls into branching event  $b$ , and has an expected  $\beta_b = 4\alpha$  children at depth 6. The primary tree (rooted at the initial spurious character) has size  $\zeta_5 = 2$  at depth 5. Since one child produces  $\beta_a$  children and the other produces  $\beta_b$  children, the tree will have an expected size  $\zeta_6 = 1 \cdot \beta_a + 1 \cdot \beta_b$  at depth 6. ■

For reverse probing patterns, it was possible to estimate the fraction of false paths which would be extended by a non-constraining query simply by counting the number of constrained positions in a query ( $c_i$ ). For direct probes, a more precise calculation is needed.

We need to calculate the expected proportion of the paths at depth  $i$  which may be eliminated by a non-constraining query due to positions of disagreement with the correct path. If we denote the fraction of paths which are guaranteed to be extended at depth  $i$  (since they agree with the correct path in all sampled positions) as  $\rho_i$ , then we can calculate the width of the primary tree as

$$\zeta_i = \rho_i \cdot \zeta_{i-1}\beta_a + (1 - \rho_i) \cdot \zeta_{i-1}\beta_b$$

It remains to calculate  $\rho_i$ . A path may potentially be eliminated if it differs from the correct path at a position which is sampled by the current query. False paths can disagree with the correct path only at positions to the right of the initial branching. The structure of direct  $(s, r)$  probing patterns means that a path may be eliminated at depth  $i < s \cdot r$  if and only if it contains a spurious character at some position  $j$  such that  $j = i - s, i - 2s, \dots, i - rs$  (i. e., where  $(i - j) \bmod s = 0$ ).

**Example 3.8.** Consider the (4,4)-direct probing pattern NNNN...N...N...N...N. At depth 14, 15, and 16, there are three positions sampled by the query which fall to the right of the initial ambiguous character, and are sampled by the query. The initial branching position is indicated in square brackets [T][A], and the potential positions of disagreement in the query are lightly shaded:

<i>Depth</i>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16								
	...A	C	C	G	G	A	C	[T]	A	T	C	A	C	C	T	C	G	A	T	C	G	C	A	...
								[A]	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
14		N	N	N	N	.	.	N	.	.	.	N	.	.	.	N	.	.	.	.	.	.	.	?
15			N	N	N	N	.	.	.	N	.	.	.	.	N	.	.	.	.	.	.	.	.	?
16				N	N	N	N	.	.	.	N	.	.	.	N	.	.	.	.	N	.	.	.	?

■

We want to precisely calculate the effect that non-constraining queries have on the width of the path tree. We will use direct (4,4)-probes as an illustrative example of direct probing patterns. Example 3.8 showed that at depth 14, 15 and 16, there are exactly 3 positions where a query can disagree with the correct path. At these depths in the tree, the proportion of paths which require fooling probe support is

constant, so  $\rho_{14} = \rho_{15} = \rho_{16}$ . By analagous argument, at depths 10, 11 and 12, the queries contain exactly 2 positions which can disagree with the correct path.

For any direct  $(s, r)$ -pattern, the first  $rs$  queries can be divided into  $r$  regions (for  $(4,4)$ -probes,  $r = 4$ ). Within each region, there are  $s - 1$  non-constraining queries for which  $\rho_i$  is constant, since the number of potential disagreements between the correct and spurious paths is the same. Each region with a constant  $\rho_i$  value contains  $s - 1$  queries.

The 4 regions for direct  $(4,4)$ -probes are explicitly defined below. For each region, the  $s - 1 = 3$  included queries are aligned with a symbolic DNA string. In the DNA string, the character 'Y' represents the branching position, and the character 'X' an arbitrary natural base. Each query string is labeled with the depth of its right-most character. The first region, region 1 corresponds to depths  $i = 2, 3, 4$  in the tree. These queries do not sample any characters to the right of the branching position, so all paths are guaranteed to be extended, indicating that  $\rho_i = 1$ . (In other words, in region 1, the probes that allow the extension of the correct path also permit all spurious paths to be extended as well.)

<i>Depth</i>													1	2	3	4
													[Y]	X	X	X
2	N	N	N	N	.	.	.	N	.	.	.	N	.	.	.	?
3		N	N	N	N	.	.	.	N	.	.	.	N	.	.	?
4			N	N	N	N	.	.	.	N	.	.	.	N	.	?

In region 2, corresponding to  $i = 6, 7, 8$ , there is a single sampled character to the right of the initial branching position:

<i>Depth</i>																	1	2	3	4	5	6	7	8
																	[Y]	X	X	X	X	X	X	X
6	N	N	N	N	.	.	.	N	.	.	.	N	.	.	.	N	.	.	?					
7		N	N	N	N	.	.	.	N	.	.	.	N	.	.	.	N	.	?					
8			N	N	N	N	.	.	.	N	.	.	.	N	.	.	.	N	?					

In region 3, where  $i = 10, 11, 12$ , each query contains 2 sampled positions to the right of the initial branching:

<i>Depth</i>		1	2	3	4	5	6	7	8	9	10	11	12
		[Y]	X	X	X	X	X	X	X	X	X	X	X
10	N	N	N	N	.	.	.	N	.	.	.	?	
11		N	N	N	N	.	.	.	N	.	.	.	?
12			N	N	N	N	.	.	.	N	.	.	?

And region 4, where  $i = 14, 15, 16$ , is the same as for the above example, where each query contains three sampled positions to the right of the initial branching:

<i>Depth</i>		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		[Y]	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
14	N	N	N	N	.	.	.	N	.	.	.	N	.	.	.	?	
15		N	N	N	N	.	.	.	N	.	.	.	N	.	.	.	?
16			N	N	N	N	.	.	.	N	.	.	.	N	.	.	?

All of the regions can be summarized quite simply. In the following table, the regions themselves are shaded, and shown on the 3<sup>rd</sup> line of the following table. The second row contains the (4, 4)-direct probing pattern, and the first row shows the depth at which the corresponding character in the probing pattern intersects with the branching position. The depths in the tree at which queries sample the initial branching are shown in *italics*.

<i>Depth</i>	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	X	X	X	X	.	.	.	X	.	.	.	X	.	.	.	X	.	.	.	X
<i>Region</i>							4	-		3			2				1			-

In general, a direct  $(s, r)$ -probing pattern contains  $r$  regions of  $s - 1$  positions. The queries in region  $j$  sample  $j - 1$  positions beyond the initial branching—these are the positions in the queries above which are highlighted in grey boxes. If a false path differs from the correct sequence any sampled position, that path may be eliminated by the corresponding query. We just need to determine what fraction of paths agree (or disagree) with the correct sequence at a sampled position; this is the value  $\rho_i$  (or  $1 - \rho_i$ ).

Note that although the *width* of the path tree may increase or decrease at every step  $i$ , the expected *fraction* of paths that agrees with the correct sequence is determined only by the steps which could have produced mismatches between the correct and spurious paths. For instance, in region 2, the fraction  $\rho_1$  of paths in the tree

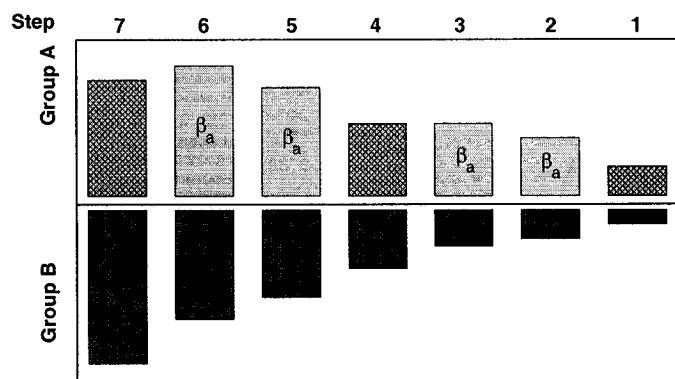


Figure 3.9: A path tree consisting of two groups of paths: A and B. At depth 2, 3, 5 and 6, the two groups are each expected to increase in size by a factor of  $\beta_a$ . At depths 1, 4 and 7, the multipliers for groups A and B are different. From depth 1-3 (and again from depth 4-6), the ratio of paths  $|A| : |B|$  remains constant.

which agree with the correct path is determined uniquely by the number of branches produced at depth  $i - s$ . In region 3, where there are 2 sites of potential mismatch, the fraction of spurious paths which agree with the correct path is determined by how many branches were produced at depths  $i - s$  and  $i - 2s$ . At all other depths  $1 \dots i - 1$ , although the tree may have grown or shrunk in width, the fraction of matching paths is not affected. This is due simply to the fact that there are essentially two groups of paths: those that agree with the correct sequence at the relevant locations and those that do not. Both groups of paths produce the same expected number of descendants at the intervening depths in the tree, so the fraction of matching paths does not change. Figure 3.9 illustrates this phenomenon.

We can now move on to the  $\rho_i$  coefficients themselves, beginning with the simplest possible. Queries in region 1 do not sample any sites of possible disagreement, so they are guaranteed to extend all spurious paths.

Figure 3.10 shows a typical region 2 query with a (4,4)-direct probing pattern. The query shown (depth 7) samples a single character to the right of the initial ambiguity. Although the number of paths in the tree increases after depth 3, the fraction of paths which differ from the correct path does not. The paths which disagree with the correct sequence at depth 3 are expected to produce the same number of children at every intermediate depth as other paths (as illustrated in Figure 3.9).

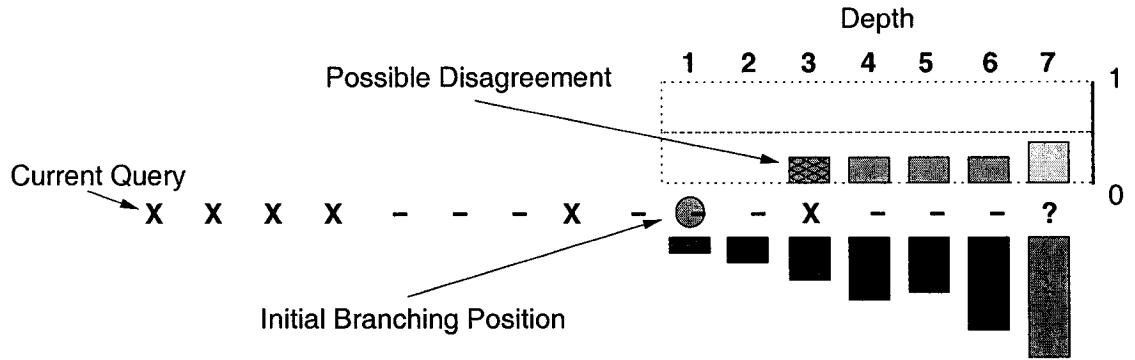


Figure 3.10: A query in region 2 samples each path in the tree at one position to the right of the initial branching position. The depth-7 query is shown in the middle, with the initial branching position highlighted. The bars beneath the query show the expected size of the primary tree; the lighter bars above the query show the fraction of paths which might contain a spurious character at the sampled position.

How do we calculate  $\rho_i$  for queries in region 2? Consider for a moment the case where the query at depth  $i - s$  spawned only a single child. In practice, it is expected to produce  $\beta_a = 1 + 3\alpha$  descendants. However, if there were no fooling probes which produced a branch for then *all* of the paths at depth  $i$  are guaranteed to be extended: a false path can be eliminated by a query in region 2 only if it differs from the correct path at the sampled position  $i - s$ .

We expect each path in the tree at depth  $i - s$  to have spawned  $\beta_a = 1 + 3\alpha$  descendants. One descendant is guaranteed to exist, and to match the correct sequence. There are 3 descendants which are created with probability  $\alpha$  and which *do not* match the correct sequence. Each of the paths spawned at depth  $i - s$  spawns the same number of descendants at depths  $i - s - 1, i - s - 2, \dots, i - 1$ . Thus, if there were 2 descendants (and therefore a single false path) spawned at depth  $i - s$ , then approximately  $\frac{1}{2}$  of the queries at depth  $i$  match the correct sequence, since they are descendants of the paths which matched the correct sequence at depth  $i - s$ . If 2 false paths were spawned (for total of 3 descendants) at depth  $i - s$ , then only  $\frac{1}{3}$  of the queries at depth  $i$  match the correct sequence. In general, there are  $\beta_a = 1 + 3\alpha$  paths spawned at depth  $i - s$ , so  $\frac{1}{\beta_a}$  of the paths at depth  $i$  match the correct sequence over all sampled characters. Thus, in region 2,  $\rho_i = \frac{1}{\beta_a}$ .

The multipliers  $\rho_i$  for region 3 are yet more complex to calculate. There are two



sampled positions to the right of the branching position. To determine the value of  $\rho_i$  in region 3, we need to calculate the fraction of paths in the tree which differ from the correct sequence at either of the sampled positions.

Again, we consider only the paths which descended from queries at positions  $i - s$  and  $i - 2s$ . Out of these paths, only one can match the correct sequence; the rest must all be confirmed by fooling probes. We thus need to calculate the product of the multipliers at depth  $i - s$  and  $i - 2s$ . This product yields the total number of paths which *might* differ in a sampled position from a query in region 3. Figure 3.11 shows a typical region-3 query using (4,4)-direct probes.

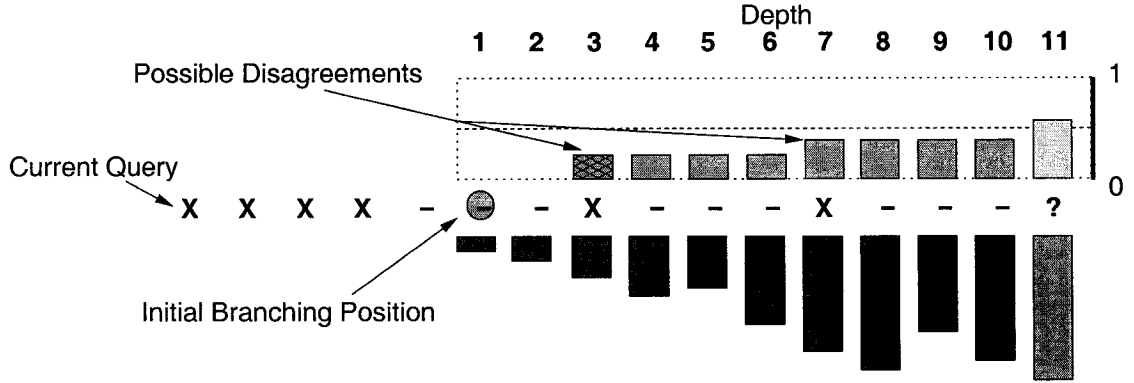


Figure 3.11: A query in the region 3 samples each path in the tree at two positions to the right of the initial branching position. The depth-11 query is shown in the middle, with the initial branching position highlighted. The bars beneath the query show, qualitatively, the expected size of the primary tree; the lighter bars above the query show the fraction of paths which might contain a spurious character at a sampled position.

Note that position  $i - 2s$  falls in region 1, and position  $i - s$  falls into region 2. This observation will help calculate the number of paths which may contain a mismatch at those two positions. In region 1, each path produces  $\beta_a = 1 + 3\alpha$  descendants. In region 2, only the fraction  $\rho_i$  of paths which agree with the correct path produce  $\beta_a$  descendants; the rest produce  $\beta_b$  children. The expected number of children produced

by each path in region 2 is

$$\begin{aligned}
 & \rho_i \cdot \beta_a + (1 - \rho_i) \cdot \beta_b \\
 = & \left( \frac{1}{\beta_a} \right) \beta_a + \left( 1 - \frac{1}{\beta_a} \right) \beta_b \\
 = & 1 + \beta_b - \frac{\beta_b}{\beta_a}
 \end{aligned}$$

Since the queries in region 1 produced  $\beta_a$  descendants each, the fraction of paths at depth  $i$  which *might* differ from the correct sequence at one of the sampled positions to the right of the branching is the product of the multipliers (number of descendants) at  $i - s$  and  $i - 2s$ ,

$$\left( 1 + \beta_b - \frac{\beta_b}{\beta_a} \right) \cdot \beta_a = \beta_a + \beta_a \beta_b - \beta_b$$

Finally, by the same argument we made for region 2, we can say that in region 3,

$$\rho_i = \frac{1}{\beta_a + \beta_a \beta_b - \beta_b}$$

We now calculate a general formula for  $\rho_i$ . Since  $\rho_i$  is constant within each region, we first define a parameter  $\psi_i, i = 1 \dots r$ , such that  $\rho_i = \frac{1}{\psi_i}$  within any given region.  $\psi_i$  is the total number of paths which *may* differ from the correct sequence at sampled positions: paths which could have branched at depth  $i - s, i - 2s, \dots, i - rs$ .

$$\begin{aligned}
 \psi_1 &= 1 \\
 \psi_i &= \psi_{i-1} \left[ \frac{1}{(\psi_{r-1})} \beta_a + \left( 1 - \frac{1}{\psi_{r-1}} \right) \beta_b \right]
 \end{aligned} \tag{3.7}$$

In Equation 3.7, the value  $\psi_{i-1}$  accounts for branches which occurred at sampled positions prior to the current one. The rest of the equation  $\left[ \frac{1}{(\psi_{r-1})} \beta_a + \left( 1 - \frac{1}{\psi_{r-1}} \right) \beta_b \right]$  accounts for the number of descendants expected for each path at depth  $i$ . The total number of paths at depth  $i$  which might have branched at depth  $i - 2, i - 2s, \dots, i - rs$  is the product of these two factors.

As we did for reverse probes, we calculate the total width of the path tree at depth  $i$  by adding the width of the primary tree at depth  $i$ , plus the width of secondary trees at depth  $1 \dots i - 1$ . The total number of paths in the complete tree at depth  $i$

is

$$\omega_i = \zeta_i + \sum_{j=1}^{i-1} \zeta'_j$$

The width of a secondary tree at depth  $i$  is given by

$$\zeta'_i = \begin{cases} 1 & \text{if } i = 1, \\ \beta_b \cdot \zeta'_{i-1} & \text{if } i > 1 \text{ and } P_{\lambda-i} \text{ is a natural base,} \\ \zeta'_{i-1} \cdot \rho_i \cdot \beta_a + \zeta'_{i-1} \cdot (1 - \rho_i) \cdot \beta_b & \text{if } i > 1 \text{ and } P_{\lambda-i} \text{ is a universal base.} \end{cases} \quad (3.8)$$

where  $\rho_i$  is  $1/\psi_i$ , and  $P_i$  denotes the  $i^{\text{th}}$  character of the probing pattern.

Now we need to calculate the probability of extending a path when  $P_{\lambda-i}$  is a natural base. Recall that we only count the width of a tree at depth  $i$  if the path tree is extended to at least depth  $i$ . Therefore, even if  $P_{\lambda-i}$  is a natural base, at least one of the paths in the tree at depth  $i - 1$  must have been confirmed by a fooling probe.

For reverse probing patterns, we claimed that if there were  $\zeta_{i-1}$  paths in the tree at depth  $i - 1$ , at least one of those paths must have been extended to depth  $i$ , and the rest all required a fooling probe. That was a reasonable approximation for reverse probing patterns. For direct probing patterns, we need to be more precise. Observe that if one path ( $p_1$ ) at depth  $i - 1$  is extended to depth  $i$ , then any other path ( $p_2$ ) that matches the extended path over all sampled positions will also be extended. The paths  $p_1$  and  $p_2$  may contain disagreements in positions which are not sampled by the current query. When using reverse patterns, the first  $s - 1$  queries after a branching all sample the ambiguous position, so for the first  $s - 1$  levels there are no unsampled positions after the branching. When a single path is extended during these first  $s - 1$  positions, it is the *only* path which is guaranteed to be extended; all other paths require supporting fooling probes.

Using direct probing patterns, the situation is much different. After the initial branching, there are  $s - 1$  queries that are guaranteed to extend every false path, and each of these queries may produce additional branches from. Furthermore, at depth  $s + 1$  in the tree (the first constraining query after the initial branch), if there is a single probe which confirms a spurious extension, then *all* paths which descended from that particular initial extension will also be extended. Thus, the fraction of paths which are guaranteed to be extended to depth  $s + 1$  from depth  $s$  in the tree, is

$\frac{1}{\# \text{ of initial branches}}$ . Since the expected number of initial branches in the primary tree is known to be  $1/\beta_d = \frac{3\alpha}{1-(1-\alpha)^3}$ , the fraction of paths guaranteed to be extended (given that at least one path was extended) at depth  $s$  is  $\beta_d$ . These paths are governed by branching event  $e$ : each may potentially be confirmed by 4 fooling probes, and we know that at least one fooling probe *is* present. They are expected to spawn  $\beta_e = \frac{4\alpha}{1-(1-\alpha)^4}$  children each. The remaining paths must all be confirmed by a fooling probe or be eliminated, and thus fall into branching event  $b$ .

After the second constraining query, there are an additional  $s - 1$  queries which do not sample the initial branching—the behavior of the path tree at these depths is described above. Following these queries, at depth  $2s + 1$  in the tree another constraining query occurs. Again, we want to determine the fraction of the paths in the tree that are guaranteed extensions, given that at least one path is extended to depth  $2s + 1$ . Here, we need only consider the paths which may have branched at depth  $s + 1$ . At depth  $s + 1$ , there were

$$\frac{1}{\beta_d} \cdot \beta_e + \left(1 - \frac{1}{\beta_d}\right) \cdot \beta_b$$

descendants to each path in the tree. By multiplying by the initial number of paths in the tree ( $\beta_d$ ) by this second branching factor, we can calculate the the total number of paths in the tree which may potentially be extended by the same query that extends a single path in the set:

$$\left[ \frac{1}{\beta_d} \cdot \beta_e + \left(1 - \frac{1}{\beta_d}\right) \cdot \beta_b \right] \cdot \beta_d = \beta_e + \beta_d \beta_b - \beta_b$$

Analogous to our calculation for the non-constraining queries, the proportion of paths which are guaranteed to be extended at depth  $i$ , when  $i$  is a constraining query can be expressed as a simple recurrence. We consider only the positions sampled by the current query which fall to the right of the initial branching position. If there are  $j$  such positions, then the proportion of paths which produce  $\beta_e$  expected descendants is just  $1/\psi_j$ , where the value of  $\psi_j$  can be calculated using Equation 3.7.

The expected width of the primary path tree using direct  $(s, r)$ -probes is given by the following recurrence:

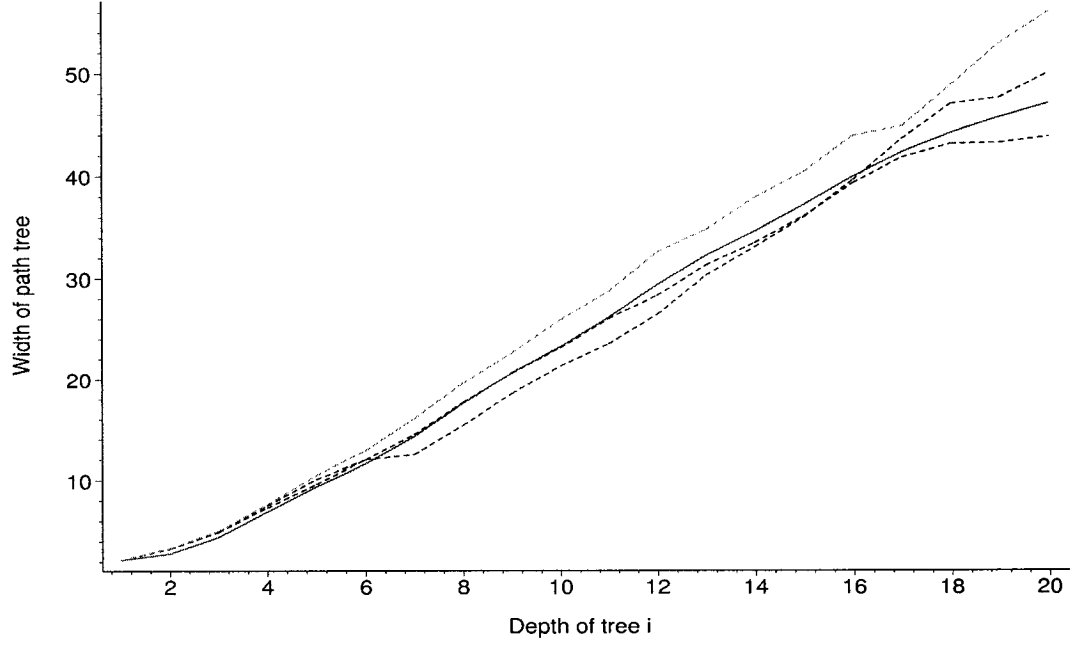


Figure 3.12: Predicted vs. observed tree widths for (4,4)-direct probes at  $m = 12000$ . The solid line shows the predicted tree width, while the three dotted lines correspond to three separate observations.

$$\zeta_i = \begin{cases} 1 & \text{if } i = 1, \\ \zeta_{i-1} \cdot \psi_i \cdot \beta_e + \zeta_{i-1} \cdot (1 - \psi_i) \cdot \beta_b & \text{if } i > 1 \text{ and } P_{\lambda-i} \text{ is a natural base,} \\ \zeta_{i-1} \cdot \rho_i \cdot \beta_a + \zeta_{i-1} \cdot (1 - \rho_i) \cdot \beta_b & \text{if } i > 1 \text{ and } P_{\lambda-i} \text{ is a universal base.} \end{cases} \quad (3.9)$$

Finally, the total width of the path tree at depth  $i$ , including the correct path, and all spurious paths contained in the primary path tree and all  $i - 1$  secondary secondary path trees is

$$\omega_i = 1 + \zeta_i \sum_{j=1}^{i-1} \zeta'_j$$

Figures 3.12 and 3.13 show the observed vs. predicted tree sizes for (4,4)-direct probing patterns for  $m = 12000$  and  $m = 15000$ , respectively. In each graph, the analytical curve is presented along with the observed tree sizes from three sequencing attempts. The predicted values show a remarkable match to the observed tree widths.

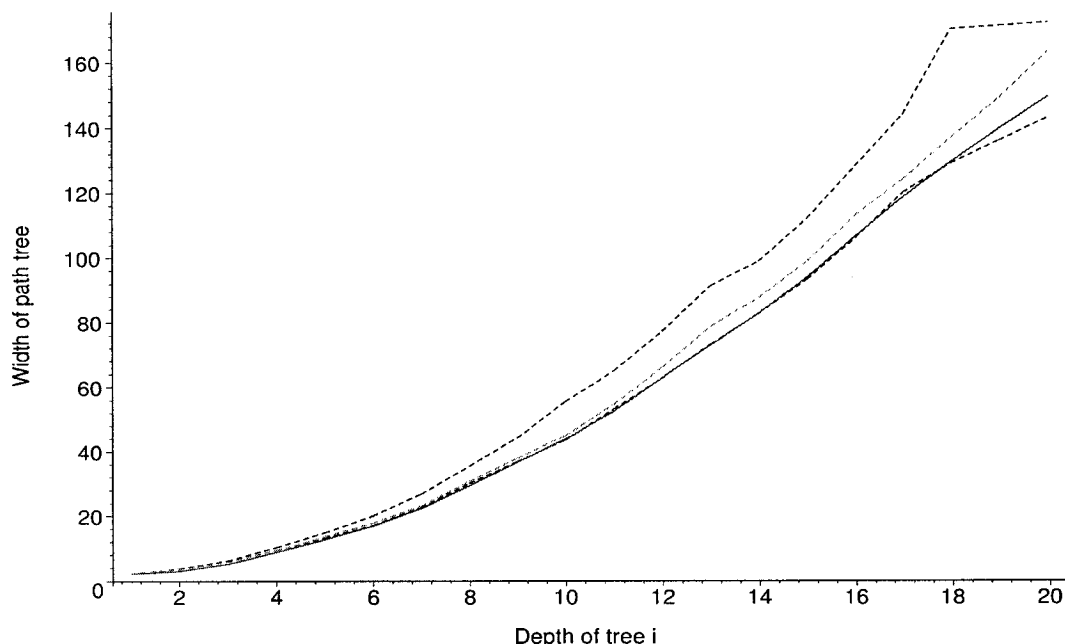


Figure 3.13: Predicted vs. observed tree widths for (4,4)-direct probes at  $m = 15000$ . The solid line shows the predicted tree width, while the three dotted lines correspond to three separate observations.

### 3.3.4 Segment Length, Tree Size and Work

The total number of nodes created in the path tree when EXTEND attempts to resolve ambiguous extensions is one of the most significant factors determining the amount of work, in terms of total spectrum queries, the gapped-SBH algorithm performs while reconstructing sequences. Two other parameters affect the total work as well: the frequency with which the branching event is initiated, and the average depth reached before an ambiguous extension is resolved.

If each character added to the putative sequence was independent of the previous one, we would expect that approximately  $(1 - \alpha)^3$  of all initial queries would produce an unambiguous extension. If we constrain the branching EXTEND algorithm to produce segments of only a single character in length, this is precisely the behaviour that is observed. When  $m = 12000$ ,  $\alpha \approx 0.167$ , and  $(1 - \alpha^3) \approx 0.58$ . Over several hundred sequencing trials, we find that about 57% of all initial queries produce a single, unambiguous extension.

However, the behaviour of the EXTEND algorithm significantly alters the incidence

of the *simple* and *branching* extension modes. When the initial ambiguity is resolved, EXTEND produces a segment consisting of the unambiguous prefix derived from the path tree. This means that, if a  $k$ -character segment is produced—beginning at depth 1 and extended to depth  $k$  in the tree—there must have been a branch in the tree at depth  $k + 1$ . If the path did not branch at depth  $k + 1$ , then the path to depth  $k$  would have had only one descendant in the tree, indicating an unambiguous ‘next’ character, and the segment would have been longer than  $k$  characters.

When the  $k$ -character segment is appended to the putative sequence, the next query made by EXTEND is virtually guaranteed to produce a branch. In fact, the branch will be the same one that occurred in the path tree. Thus, after a the first ambiguous character is resolved by EXTEND’s branching mode, almost *no* characters will ever be added by the *simple* extension mode. EXTEND’s next query is guaranteed to produce a branch unless *all* of the spurious paths in a path tree terminate simultaneously, leaving only a single correct path alive.

We want to calculate how much ‘work’ is performed by the gapped-SBH algorithm to reconstruct a sequence of length  $m$  with a particular probing pattern. A good metric for measuring the work performed is  $\left(\frac{\# \text{ spectrum queries}}{\text{nucleotide}}\right)$ . Since virtually every character in the reconstructed sequence is produced by EXTEND’s branching mode, we can estimate this value by calculating two values:

1. The average length, in characters, of an extension segment.
2. The average size, in nodes, of a path tree.

We begin by estimating the size of a path tree. Each node in the tree requires 4 spectrum queries: one for each potential child-node, so the average tree size yields the average number of spectrum queries performed to produce an extension segment. By dividing the average tree size by the average segment length, we arrive at an estimate for  $\left(\frac{\# \text{ spectrum queries}}{\text{nucleotide}}\right)$ .

The initial ambiguous extension in a path tree can be resolved only by a constraining query. Such queries will eliminate the tree if there are *no* fooling probes which confirm the spurious paths. Each individual spurious path in the tree is eliminated with probability  $(1 - \alpha)^4$ . However, note that when the first constraining query after the initial branch is executed, *all* of the paths which share the same initial (spurious)

character are confirmed or eliminated by the same query. If there are one or more fooling probes which confirm one such path, then all of them are extended (or produce another branch). And if there are no such fooling probes, then all of the paths sharing the same initial character are eliminated simultaneously.

If we assume that at depth 1, the primary path tree has width 1 (there is only a single initial spurious character), then the probability of resolving the branch with the first constraining query is

$$(1 - \alpha)^4$$

However, since there are  $\beta_d = \frac{3\alpha}{1-(1-\alpha)^3}$  initial spurious characters, and each of the subtrees sharing the same initial character may be extended by independent fooling probes, the query at depth 2 resolves the initial ambiguity if and only if *all* spurious paths are eliminated. The probability of this event is most easily calculated using a generating function.

The generating function  $G_i(z)$  of the probability distribution of a variable  $v \geq 0$  is a polynomial function that allows us to compactly represent the distribution,  $p(v = j)$ , for  $j = 1, 1, \dots$ . In our case, we have a random variable describing the width of the path tree, which is governed by a different probability distribution at each depth  $i$ . If we want to calculate the probability that there are 0 items in the tree at depth 1, we compute  $G_1(0)$ . If we want to determine the probability that there are 0 children in the tree at depth 2, we must first compute  $G_2(z)$ , since the number of paths in the tree at depth 2 is governed by a different probability function. Once  $G_2(z)$  is determined, the probability that there are no spurious paths remaining in the tree at depth 0 is determined by computing  $G_2(0)$ .

We will first compute  $G_1(z)$ . To begin, we define a parameter  $\delta = 1 - \alpha$ , for the sake of legibility. There is always at least one spurious path in a tree, which contributes a factor  $z$  to the generating function, and there may be two other paths, with probability  $\alpha$ . Each potential path gives rise to a factor  $(\delta + \alpha z)$ . This gives us

$$\begin{aligned} G_1(z) &= z(\delta + \alpha z)^2 \\ &= \delta^2 z + 2\alpha\delta z^2 + \alpha^2 z^3 \end{aligned} \tag{3.10}$$

The coefficient of  $z^n$  gives the probability that there are  $n$  paths in the tree at that depth. The probability of finding a single path in the tree is  $\delta^2$ ; the probability



of 2 paths is  $2\alpha\delta$ , and the probability of 3 paths is  $\alpha^2$ . Note that the probability of finding 0 or 4 spurious paths is 0, since we are considering only spurious trees (of which there can be only 3) which *exist* at depth 1 (and consequently contain at least one path). At depths  $i > 1$ , the generating function grows rapidly more complex.

When the query performed at depth  $i$  is not a constraining query, a spurious path is freely extended only if it agrees with the correct path at all sampled positions. Equation 3.4 yields the approximate proportion of paths that are freely extended in this case. For  $(s, r)$ -reverse probing patterns, the proportion of paths which are freely extended by a non-constraining query is  $\frac{1}{s-1}$ , and for the (4,4)- and (5,3)-patterns that are typically used,  $\frac{1}{s-1} \leq \frac{1}{64}$ . For the purposes of determining the ultimate tree depth, this value is small enough that it may be safely ignored. We can assume that all paths in the tree are equally likely to be eliminated, even by *non*-constraining queries. For  $(s, r)$ -reverse probes, we assume that all queries are just as likely to resolve the initial extension and eliminate the entire path tree. Thus, each individual path at depth  $i > 2$  is governed by the following generating function:

$$R(z) = (\delta + \alpha z)^4 \quad (3.11)$$

If there is only one path in the tree at depth 1, then  $G_2(0) = R(0) = \delta^4 = (1 - \alpha)^4$  gives the probability that the tree is resolved by the query at depth 2, as expected. The generating function for  $G_2(z)$  is just  $R(G_1(z))$ , which can be calculated by substituting the generating function  $G_2(z) = (\delta + \alpha z)^4$  for  $z$  in  $G_1(z)$ . This gives us

$$G_2(G_1(z)) = (\delta + \alpha z)^4 (\delta + \alpha(\delta + \alpha z)^4)^2$$

The probability that a random path tree is eliminated at depth 2 can be determined by computing  $G_2(0)$ . At depths  $i > 2$ , the generating function  $R(z)$  must be applied recursively,  $i - 1$  times. Note that

$$R_n(z) = R(R_{n-1}(z)) = R(R(R_{n-2}(z))) = \dots = R(R(\dots R(z) \dots))$$

and that consequently,

$$G_i(z) = R_{i-1}(G_1(z)) \quad (3.12)$$

so the generating function for the number of paths in the tree at depth  $i$  can be computed manually in a simple (although tedious) manner. In fact, at depths  $i > 2$ ,



there is at least one fooling probe matching the queries  $q_a$  and  $q_b$ , respectively. There are 4 potential fooling probes that will match either query; each query produces an expected  $4\alpha$  responses in the spectrum. The probability of finding 0 matching probes for  $q_a$  is  $(1 - \alpha)^4$ ; the same as for  $q_b$ . However, since each of the fooling probes confirming path  $a$  is independent of the fooling probes confirming path  $b$ , the probability of finding no fooling probes to confirm *either* path is  $((1 - \alpha)^4)^2$ . If there were 3 fooling probes matching the single spurious path at depth 4, there would be 3 independent paths at depth 7, and hence 12 independent probes which could confirm one or more of them. In general, when there are  $n$  different constraining queries performed, the probability of eliminating the entire tree is  $(1 - \alpha)^{4n}$ .

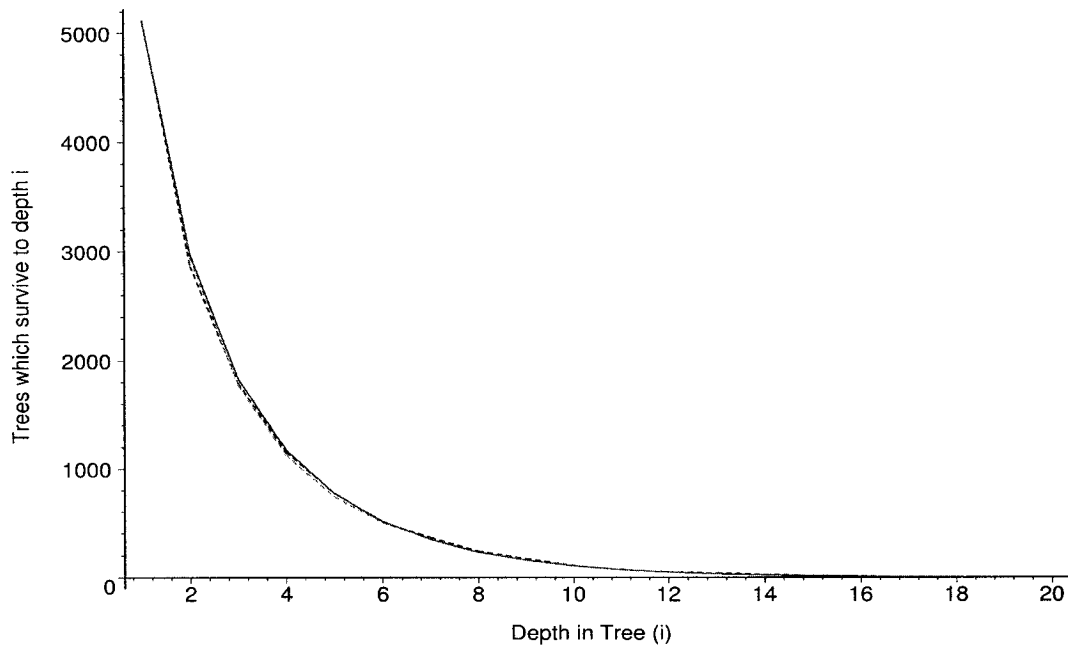


Figure 3.14: Predicted vs. observed tree depth for (4,4)-reverse probes at  $m = 12000$ . The solid line shows the predicted value, and the two dotted lines show the observed values for two sequencing attempts.

All that remains to be done is to calculate the expected number of independent spurious paths—which differ in at least one sampled position—and the probability of eliminating the tree at every constraining level can be easily calculated. We simply need to determine the proportion of paths which differ from the correct path at a sampled position. A slight modification of Equation 3.7 yields the expected number

of independent queries after  $i$  constraining queries have been performed (recall that for direct probing patterns, every query is considered to be a constraining query). Denoting the number of paths with  $i$  sites of potential disagreement as  $\eta_i$ , we have:

$$\begin{aligned}\eta_1 &= \beta_d \\ \eta_i &= \eta_{i-1} \left[ \frac{1}{\eta_{i-1}} \beta_e + \left( 1 - \frac{1}{\eta_{i-1}} \right) \beta_b \right]\end{aligned}\tag{3.13}$$

At the  $i^{\text{th}}$  constraining query, the probability of eliminating the primary tree and thereby resolving the initial branch is  $(1 - \alpha)^{4\eta_i}$ . The probability  $\theta_i$  that a tree is extended to depth  $i$  is simply the product

$$\begin{aligned}\theta_i &= \prod_{j=1}^i (1 - \alpha)^{4\eta_j} \\ &= (1 - \alpha)^{4i} \prod_{j=1}^i (1 - \alpha)^{\eta_j}\end{aligned}\tag{3.14}$$

which does not have a simple closed form solution.

Figures 3.14 and 3.15 show the proportion of trees extended to at least depth  $i$  by Equations 3.13 and 3.14 for (4,4)-direct and reverse probing patterns at  $m = 12000$ . Each figure shows the predicted analytical curve and three experimental curves obtained from three sequencing trials. The correspondence between our model and the experimental data is remarkable.

Now that we have developed a reasonable model for predicting the likelihood of extending a tree to depth  $i$ , we can calculate the expected number of queries performed before resolving an ambiguous extension. Recall from Chapter 2 that a spectrum query determines the presence or absence of a particular probe, and so 4 queries are required for each node in the tree. The probability that a tree is extended to depth  $i$  is  $\theta_i$ , and the expected size of the tree at depth  $i$  is  $\omega_i$ . If the tree *does* reach depth  $i$ , we expect there to be  $\omega_i$  nodes in the tree at that depth. Since the probability of reaching depth  $i$  is  $\theta_i$ , there is a probability of  $\theta_i$  that  $4\omega_i$  queries at depth  $i$  will be performed. Thus, in a tree of unknown depth, we expect  $4\omega_i\theta_i$  queries to be performed at depth  $i$ . We denote the total number of expected queries in an average tree created for a sequence of length  $m$  as  $Q(m)$ ,

$$Q(m) = \sum_{i=1}^{\infty} 4\omega_i\theta_i\tag{3.15}$$

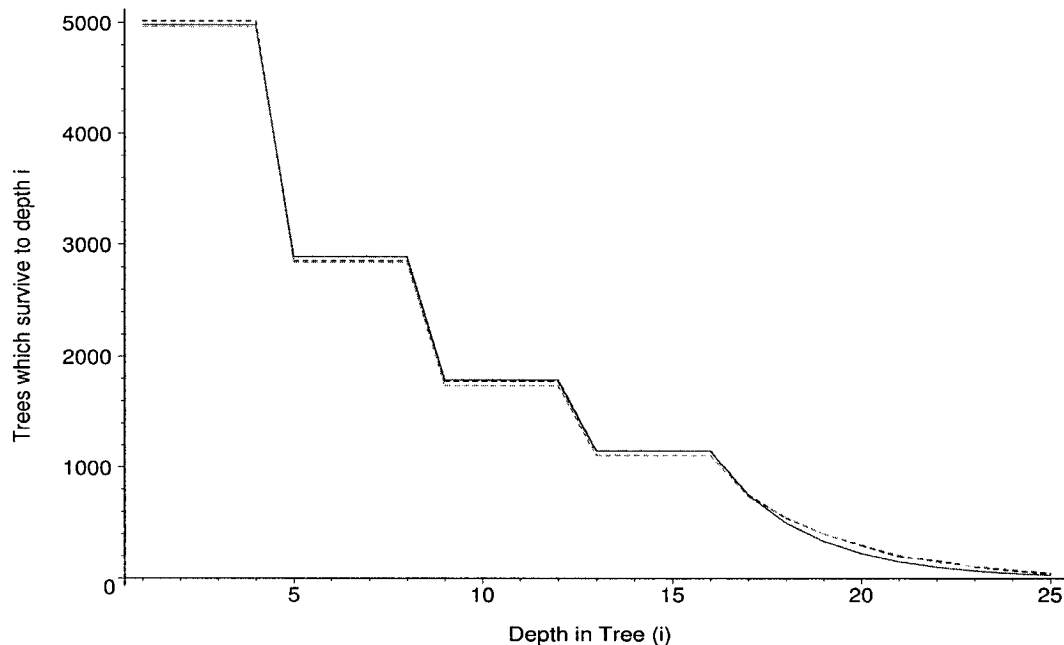


Figure 3.15: Predicted vs. observed tree depth for (4,4)-direct probes at  $m = 12000$ . The solid line shows the predicted value, and the two dotted lines show the observed values for two sequencing attempts.

In practice, we sum over depths  $i = 1 \dots \frac{3}{2}\lambda$ ; the probability that a tree reaches a depth beyond that is negligible. The following table gives the results of this estimate of tree size with direct and reverse (4,4)- probes for  $m = 12000$  and  $m = 15000$ , along with the average tree size over several sequencing attempts using the same probing pattern and  $m$ .

Probe	$m$	Predicted	Observed
(4,4)-direct	12000	543	589
(4,4)-reverse	12000	31.1	35.2
(4,4)-direct	15000	2072	2153
(4,4)-reverse	15000	77.4	72.8

Now that we can predict the amount of work required to resolve a single ambiguous extension, we should be able to estimate the total work required to reconstruct a complete sequence. We just need to calculate the expected length of a branching-mode segment. Unfortunately, this process is not simple. Moreover, the strategy of

selecting the longest unambiguous *prefix* of the path tree after resolving a branching introduces a selection effect on the path trees themselves.

If, upon eliminating all of the spurious paths in a tree, only a *single character*—the now unambiguous character at the branching position  $i$ —is appended to the putative sequence, the (possibly empty) path tree produced at position  $i + 1$  is independent of the tree produced at  $i$ . By selecting a sequence whose length is determined by subsequent (secondary) branches in the path tree, we are eliminating some subset of path trees from consideration.

This problem warrants further consideration, but is beyond the current scope of this thesis. For now, we can estimate the work required to reconstruct a sequence when only single-character segments are selected from a path tree. Recall that Equation 3.15 gives the expected number of queries performed to resolve an ambiguous extension. We denoting the probability of a branching event as  $p_b = 1 - (1 - \alpha)^3$ . A little thought reveals that 4 queries are performed for every character produced by the algorithm's *simple* extension mode. Thus, the predicted total number of queries needed to reconstruct an  $m$ -length sequence is

$$m \cdot Q(m) \cdot p_b + 4m \cdot (1 - p_b)$$

The following table shows the predicted vs. observed total number of queries performed during sequencing attempts for 12000- and 15000-character target sequences:

Probe	$m$	Predicted	Observed
(4,4)-direct	12000	$2.77 \times 10^6$	$2.64 \times 10^6$
(4,4)-reverse	12000	$1.86 \times 10^5$	$1.83 \times 10^5$
(4,4)-direct	15000	$1.54 \times 10^7$	$1.56 \times 10^7$
(4,4)-reverse	15000	$6.02 \times 10^5$	$5.87 \times 10^5$

As expected, the analytical and experimental values are in close agreement. However, this constrained case does not accurately predict the amount of work required to reconstruct a sequence when we take the longest prefix of the path tree after resolving a branching. In such cases, although the individual trees tend to be larger, there are fewer of them, so the total amount of work performed appears experimentally to be reduced by about 30%, for both direct and reverse probes and  $m = 12000, 15000$ .

## Chapter 4

# Failure Modes of the Gapped-SBH Algorithm

We know that sequencing failures occur because of fooling probes in the spectrum. The likelihood of finding a fooling probe increases as the length of the target sequence increases, so longer DNA fragments are more likely to result in failure than shorter ones. Beginning with these observations, we develop a means of analytically predicting the likelihood of sequencing failure. This chapter presents a formal analysis of the probability of sequencing failure as a function of the probing pattern, the length of the sequence, and other parameters to the algorithm.

Subsequent sections discuss in greater detail the probability of encountering different types of sequencing failures. The most common sequencing failure, called a Mode 1 failure, occurs when there are  $\kappa$  fooling probes scattered throughout the target sequence which confirm a spurious extension. Alternatively, there are Mode 2 failures, which occur when two similar  $(\lambda - 1)$ -character subsequences occur at different points in the target sequence, and the spectrum contains the fooling probes required to compensate for the differences between them. Within the range of fragment lengths  $m$  where the gapped-SBH can correctly reconstruct a random sequence with at least a 90% chance of success, Mode 2 failures are less likely to occur than Mode 1. However, each of these two Modes results in the depth-bound ( $H$ ) being reached. A different type of sequencing failure occurs when the tree of paths grows wider than the breadth-bound of the algorithm ( $B$ ); this event is discussed in turn.

Finally, some unique problems arise when sequencing *natural* DNA—DNA drawn from a living organism. Since natural DNA is (unfortunately) not generated by a maximum-entropy process, the incidence of Mode 1, Mode 2, and breadth-bound failures all increase, and the relative occurrence of the three types of failures changes significantly. These issues are introduced in the last section of this chapter.

## 4.1 Sequencing Failures

Sequencing failures are caused by a set of fooling probes which initiate and then confirm an incorrect extension at a particular point in the sequence. Different modes of failure require different sets of fooling probes, but in all cases, there is *some* set of probes which serve to create an unresolvable ambiguous extension. We denote this set of probes as follows:

**Definition 4.1.** *The set of probes which confirm an extension character is called the extension set of that character.*

Each possible extension in a Mode 1 failure has a  $\kappa$ -probe extension set, consisting of each of the  $\kappa$  probes which samples the ambiguous extension character. Each possible extension in a Mode 2 failures has up to  $\lambda$  probes in its extension set.

Finally, it is convenient to differentiate the initial fooling probe that starts the branching mode of EXTEND from the other  $(\kappa - 1)$  fooling probes needed to confirm a spurious extension.

**Definition 4.2.** *At any point during the sequencing process, a probe is said to be a feasible-extension probe if its  $(\lambda - 1)$ -character prefix matches the  $(\lambda - 1)$ -character suffix of the putative sequence.*

The EXTEND algorithm attempts to add characters one at a time to the putative sequence. When EXTEND queries the spectrum for feasible extensions to the sequence at position  $i$ , the correct feasible-extension probe is located at position  $i$  of the target sequence; any other feasible-extension probes must be located elsewhere in the target sequence. There are 3 such probes that can produce an ambiguous extension at  $i$ .



**Definition 4.3.** *A feasible-extension probe for position  $i$  which is located at position  $j \neq i$  of the target sequence is not the correct extension at  $i$  is called a spurious extension probe.*

## 4.2 Mode 1 Failures

The most common type of sequencing failure are called *Mode 1* failures. They account for the majority of failures observed for sequences of length 15000 or less, using probes with  $\kappa = 8$ . As mentioned in Chapter 2, these failures occur when there are  $\kappa$  fooling probes, scattered uniformly along the full length of the target sequence that confirm a spurious extension to the putative sequence. This results in two sequences in the path tree which are identical except for the initial ambiguous character. Once a Mode 1 spurious sequence has been extended by  $\lambda$  characters beyond the branch, no further probes sample the spurious character, so that path cannot be eliminated. Hence, it becomes impossible to disambiguate the branching position. In this section, we discuss the probability of this event. We begin with an example:

**Example 4.1.** Consider the situation where the following event occurs during a sequencing attempt. Using a direct (3,2)-probing scheme (NNN . . N . . N) , an ambiguous extension is encountered at the position indicated by the square brackets. The putative sequence lies to the left of the ambiguous position, and the two paths which emanate from the branching position are identical except for the initial ambiguous character:

```

... A A C G G T T A C A [T] T G A T A T G G A ...
                           [G] T G A T A T G G A ...

```

If we assume that the top path is the correct one, we know that the spurious path is supported by the following fooling probes:

<i>Path Tree</i>																			
... A A C G G T T A C A										[T]	T	G	A	T	A	T	G	G	A ...
										[G]	T	G	A	T	A	T	G	G	A ...
<i>Probes</i>																			
1		C	.	.	T	.	.	C	A	[G]									
2			G	.	.	T	.	.	A	[G]	T								
3				G	.	.	A	.	.	[G]	T	G							
4							A	.	.	[G]	.	.	A	T	A				
5										[G]	.	.	A	.	.	T	G	G	

There are  $\kappa = 5$  fooling probes required to confirm the spurious sequence; all of them must be present to cause a Mode 1 failure at this point in the sequence. ■

There are two distinct components to a Mode 1 failure: the initial ambiguous branch (caused by a single spurious feasible-extension probe), and the  $(\kappa - 1)$  addition fooling probes which must confirm the incorrect extension character. Additionally, in [PU00] Preparata & Upfal calculate a small factor to account for the possible overlap between the  $(\kappa - 1)$  fooling probes after the initial one. The total probability of a Mode 1 failure is given by the following theorem.

**Theorem 4.1.** *The probability of a Mode 1 failure occurring while reconstructing an  $m$ -character randomly generated-sequence from  $(s, r)$ -probes is bounded above by the equation*

$$p_1 = m \cdot \left(1 - e^{-\frac{3m}{4^\kappa}}\right) \cdot \left(1 - e^{-\frac{m}{4^\kappa}}\right)^{\kappa-1} \cdot \left(1 + \frac{4^{r+1}}{3m}\right)^r \left(1 + \frac{4^s}{3m}\right)^{(s-1)} \quad (4.1)$$

*Proof.* Each factor in Equation 4.1 will be discussed separately. First, there are  $m$  locations in the sequence at which a Mode 1 failure may occur. The first factor in Equation 4.1 ( $m$ ) accounts for all of the possible failure positions along the sequence. At each position in the sequence, a Mode 1 failure is equally likely.

The second factor  $\left(1 - e^{-\frac{3m}{4^\kappa}}\right)$  accounts for the initial fooling probe which causes an ambiguity in the extension process. There are 3 potential probes which can cause an ambiguous extension, since the spurious character must, of course, be different from the correct extension. Each potential fooling probe is independent of the others, so at least one ambiguous extension to the sequence occurs with probability  $1 - (1 - \alpha)^3$ . However, recall that  $\alpha = 1 - e^{-\frac{m}{4^\kappa}}$ , so  $(1 - \alpha)^3 = (e^{-\frac{m}{4^\kappa}})^3$ .

The third factor in Equation 4.1,  $(1 - e^{-\frac{m}{4\kappa}})^{\kappa-1}$  accounts for the  $\kappa - 1$  additional fooling probes needed to confirm the ambiguous character. Each of these fooling probes is present (independently) in the spectrum with probability  $\alpha = 1 - e^{-\frac{m}{4\kappa}}$ , and all  $\kappa - 1$  must be present to confirm the initial ambiguous character.

The final two factors correct for the possibility of overlapping probes occurring at the same position in the target sequence. A detailed explanation of this correction can be found in [PU00], and will not be explored in detail here. Briefly, the likelihood of overlapping probes is determined by the autocorrelation function of the probing pattern, and since  $(s, r)$ -probes have a reasonably low autocorrelation the correction factor is quite small. The correction for overlap adds only about 2-3% to the total probability of failure. Other (irregular) patterns exist with better or worse autocorrelation, some of which will be discussed in later chapters in this proposal.  $\square$

**Example 4.2.** A sequencing trial with direct (3,3)-probes has produced the 12-character putative sequence GCTGAGCGAGTA. Assume that the next character in the sequence is A. The next extension query is AGC...G...?; the response to this query will contain the correct extension character (A) and up to three spurious extension characters. Each of the three possible spurious extensions is produced with probability  $\alpha$ , if and only if the corresponding spurious-extension probe is found in the spectrum. The spurious extension G is contained in the query result if the spurious-extension probe AGC...G...**G** is present. Similarly, the spurious extensions T and C are contained in the query result if the spurious-extension probes AGC...G...**T** and AGC...G...**C** are present. Any one of the spurious-extension probes is present with probability  $\alpha$ , so the chance that any one of the three is found is  $1 - (1 - \alpha)^3 = 1 - e^{-\frac{3m}{4\kappa}}$ .

The presence of any one of the three spurious-extension probes initiates the branching mode of EXTEND. However, sequencing failure only occurs if a full set of  $\kappa$  probes fooling probes is present for *one* of the spurious extensions. We'll assume that the extension set for the correct extension (A) contains the six probes AGC...G...**A**, GAG...**A**...G, TA**A**...G...T, A**A**A...C...T, and **A**AG...T...T. The extension set for the spurious extension G contains the probes AGC...G...**G**, GAG...**G**...G, TA**G**...G...T, A**G**A...C...T, and **G**AG...T...T. All of the probes in the extension set must be present in the spectrum, or the EXTEND algorithm will resolve the initial ambiguity as it performs the breadth-first expansion

of the path tree. Each of the  $(\kappa - 1)$  fooling probes in the extension set beyond the initial spurious-extension probe are each present in the spectrum with probability  $\alpha$ , so the total probability of finding them is  $\alpha^{\kappa-1}$ . ■

It may appear that the probes in the extension set only need to match the correct probes in positions which precede the ambiguous character. However, if there is more than a single-character disagreement between the correct probe and the fooling probe, then additional fooling probes will be required to support the extra mismatched characters. If any of the added fooling probes introduced yet more mismatched characters the extension set quickly grows very large, and the probability of finding all of the required fooling probes decreases rapidly. To a reasonable approximation, this type of failure requires a fooling probe for every character added to the spurious path. When  $\alpha < 1/4$ , the extension of such a ‘totally spurious’ path to the depth bound  $H$  is exceedingly unlikely. However, when  $\alpha \approx 1/4$ , the probabilistic extension of paths which do not correspond to any subsequence of the target DNA fragment becomes possible. Thus,  $\alpha = 1/4$  is effectively a hard limit on the length of fragments which may be reconstructed using gapped-SBH. The fragment length ( $m$ ) at which  $\alpha \approx 1/4$  is given for  $m = 6 \dots 10$  in the following table:

$\kappa$	Maximum $m$
6	1180
7	4710
8	18900
9	75400
10	302000

For a more detailed discussion of the information-theoretic bounds on SBH, see [PFU99]. However, long before the information-theoretic limit, or the above maximum  $m$  is reached, other failures reduce the effectiveness of gapped-SBH too low for it to be useful.

Mode 1 failures describe a very specific occurrence, wherein all of the fooling probes in the extension set for a spurious extension agree with the correct sequence, except for the single spurious position. This specific case accounts (perhaps counter-intuitively) for the majority of all sequencing failures.

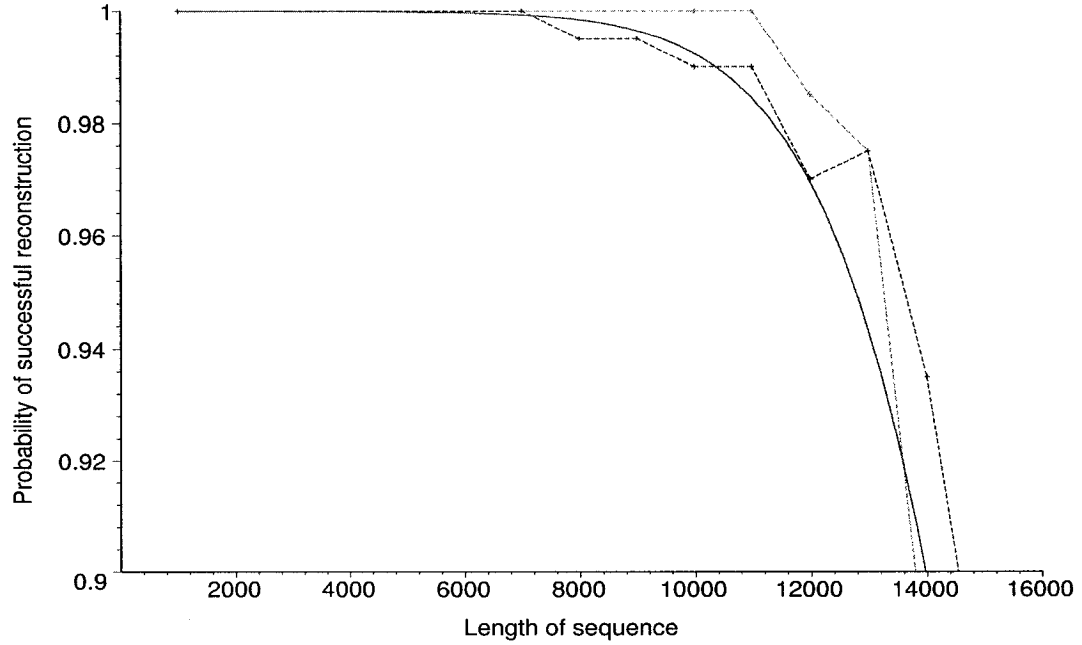


Figure 4.1: Predicted and observed incidence of Mode 1 Failures for (4,4)-direct and reverse probing patterns. The solid line represents the predicted values. The upper dotted line shows the observed values for reverse probes, and the lower dotted line the observed values for direct probes.

Equation 4.1 provides a very accurate estimate of the probability Mode 1 failures, which are the most likely sequencing failures for the SBH algorithm. Figure 4.1 displays the number of Mode 1 failures predicted by Equation 4.1 vs. the observed number of failures over several hundred sequencing trials at each  $m$ . At times, it can be useful to have a simpler equation. If we ignore the correction factor for overlapping probes—which amounts to only a few percent or less—the whole equation may be reasonably approximated by:

$$p'_1 = 1 - e^{3m\alpha^\kappa} \quad (4.2)$$

There is one important point to note regarding this approximation. Although it would appear that it might apply to an arbitrary probing pattern with  $\kappa = 8$ , it is valid only for (4,4) and (5,3) probes. The correction factor for overlapping probes has only been defined for  $(s,r)$  probes, and can quite large when  $r$  is substantially different from  $s$ .

Now we move on to another failure mode, which accounts for an increasing proportion of sequencing failures as  $m$  approaches the feasible limit.

### 4.3 Introduction to Mode 2 Failures

While Mode 1 failures account for the majority of observed sequencing failures, there is another class of failures that are similar to the failures observed in ungapped-SBH. This second class will be referred to as *Mode 2* failures, and they are qualitatively very different from Mode 1. While the correct and spurious paths in a Mode 1 failure differ only in the initial branching position, the two paths in a Mode 2 failure appear to diverge completely, with only about one fourth of the characters agreeing. Since the probabilistic extension of a completely spurious path up to the depth-limit  $H$  is extremely unlikely, we assume that Mode 2 failures have some other cause.

Mode 2 failures appear qualitatively similar to the failures which occur using the ungapped-SBH algorithm. In ungapped-SBH, a failure occurs when there is a  $(\lambda - 1)$ -character<sup>1</sup> string which occurs at two locations  $a \neq b$ , in the target sequence. However, the likelihood of encountering an exact  $(\lambda - 1)$ -character duplicate string using (4,4)- or (5,3)-probes is vanishingly small. Even with a target sequence length  $m$  near the feasible limit (i. e.  $m = 16000$ ), the probability of finding a duplicate 19-character string is only  $\frac{m}{4^{19}} \approx 5.8 \times 10^{-6}$ . Mode 2 failures are observed much more frequently than this, so there must be another cause.

First we consider the simplest case, where the target sequence *does* contain a sufficiently long duplicate string. When there is a sequence with length  $\geq \lambda - 1$  that occurs at two different locations in the target sequence, a Mode 2 failure will occur when the first of the two segments is encountered during sequencing. We can easily calculate the probability of this event. Let the two identical segments occur at positions  $a$  and  $b$ , and let them be denoted  $s(a)$  and  $s(b)$  respectively. Without loss of generality, assume that  $a < b$ . The character immediately following  $s(a)$  may be called  $e(a)$ , and the character following  $s(b)$ ,  $s(b)$ . Upon encountering the first occurrence of the duplicated segment,  $s(a)$ , the sequencing algorithm is unable to select between  $e(a)$  and  $s(b)$ , and sequencing fails.

---

<sup>1</sup>When ungapped probes are used,  $\lambda = \kappa$ .



be extended indefinitely. It is nearly identical to the aligned 8-character subsequence of the top (correct) sequence; other than the initial ambiguous character, there is one other mismatch between the correct and spurious paths, indicated in braces:  $\{G\}\{A\}$ . ■

Mode 2 failures generally occur when there are two homologous strings containing only a few mismatched positions that require only a few fooling probes to compensate for the mismatches. We want to predict the likelihood of this occurrence.

## 4.4 Probability of Mode 2 Failures

A Mode 2 failure may occur at any point in the sequence. For any given  $(\lambda - 1)$ -character substring  $s(a)$  occurring at location  $a$ , we must consider all of the possible  $(\lambda - 1)$ -character substrings of the target sequence as potential sites for the self-sustaining segment  $s(b)$ . For every value of  $b = 1 \dots m$  where  $b \neq a$ , there exists some set of fooling probes which compensate for the differences between  $s(a)$  and  $s(b)$ . Since a different set of fooling probes are required, the locations of the homologous strings are not interchangeable, and there are  $\binom{m}{2} \approx m^2$  ways of selecting  $s(a)$  and  $s(b)$  from the target sequence. The probability that any particular pair of strings  $s(a)$  and  $s(b)$  will cause a Mode 2 failure is affected by three factors:

1. The probing pattern used.
2. The number of disagreements between the two  $(\lambda - 1)$ -character homologous strings  $s(a)$  and  $s(b)$ .
3. The location of disagreements within the homologous strings.

For any particular pair of strings, it is fairly straightforward to calculate the probability that they will lead to a Mode 2 failure, simply by counting the disagreements between them and the fooling probes required to compensate.

**Example 4.4.** Referring to the same event as Example 4.3 above, we can count the number of fooling probes required by this particular pair of homologous strings, given a (3,2)-direct probing pattern. In this case, there are three fooling probes which compensate for the disagreements between the strings:



Path Tree																				
										0	1	2	3	4	5	6	7	8	9	10...
...	A	C	G	A	G	T	C	(C	T	[G]	A	G	T	{G}	A	T	A	T	A	T...
										[T]	A	G	T	{A}	A)	T	C	T	G	G...
Probes																				
1	C	G	A	.	.	C	.	.	[T]											
2					G	T	C	.	[T]	.	.	T								
3					T	C	C	.	.	A	.	.	{A}							

There are two disagreements between the correct path and the spurious path over the 8-character homologous subsequence. In the spectrum, the initial branching [G-T] is compensated for by probes 1 and 2, and the second disagreement {G-A} by probe 3. No other fooling probes are required, because the remaining probes are guaranteed in the spectrum by the sequence CT[T]AGT{A}A. ■

While it is straightforward to determine the number of fooling probes required to compensate for the differences between a particular pair of homologous strings, the probability of a Mode 2 failure is extremely complex to calculate precisely. There are  $\binom{\lambda-1}{i}$  ways of selecting  $i$  positions in a  $(\lambda-1)$ -character string, so for two  $(\lambda-1)$ -character strings which contain  $i$  mismatches, there are  $\binom{\lambda-1}{i}$  possible arrangements of the locations of disagreement between them. Moreover, there is no simple way of determining (or estimating) the average number of fooling probes required to compensate for the  $i$  disagreements. One permutation might require only a few fooling probes, while another might require the maximum  $(\lambda-1)$ .

In [HPY02], we present an exhaustive estimate for the probability of a Mode 2 failure using  $(s, r)$ -reverse probes. Here we present another, simpler approximation for the event, for both direct and reverse probing patterns. A different estimate is used for the two classes of probing patterns, but the fundamental concepts are the same, and we explore those first.

Consider a  $(\lambda-1)$ -symbol self-sustaining segment  $s(b)$  which, when aligned with the target sequence at position  $a$ , matches the target in all positions except  $s_k$ ,  $1 \leq k < \lambda$ . (This corresponds to the case where there is only one disagreement between  $s(a)$  and  $s(b)$ .) Since  $s_k$  is the site of the only disagreement between the correct sequence  $s(a)$  and the aligned self-sustaining segment  $s(b)$ , it must coincide with the branching position. In general, no matter how many mismatches there are between

the correct sequence and the self-sustaining segment, the first (left-most) mismatch must be aligned with the branching position in a Mode 2 failure: it is precisely this mismatch which causes the initial branching to occur.

**Example 4.5.** An attempt is made to sequence a target sequence  $S$  using (3,3)-reverse probes (for which  $\lambda = 12$ ). If  $S$  contains the two homologous 11-character strings  $s(a) = \text{CGAG T TGA C GC}$  and  $s(b) = \text{CGAG G TGA T GC}$ ,  $a \neq b$ , then a Mode 2 failure may occur when the sequencing algorithm encounters either  $s(a)$  or  $s(b)$ . In either case, since  $s_5$  is the first position of disagreement between  $s(a)$  and  $s(b)$ ,  $s_5$  is aligned with the initial branching position.

If the sequence before  $s(a)$  is  $\dots\text{GTTAAC TT}$ , then the following event may occur at  $s(a)$ . With the initial branching denoted  $[\text{T}][\text{G}]$ , the second disagreement denoted  $\{\text{C}\}\{\text{T}\}$ , and the self-sustaining segment enclosed in parentheses  $()$ , the event has the following appearance:

$$\begin{array}{c} \dots\text{G T T A A C T T (C G A G [T] T G A \{C\} G C A G A} \dots \\ \qquad\qquad\qquad [\text{G}] \text{ T G A \{T\} G C) C T G} \dots \end{array}$$

It is impossible for any other position  $s_i$  of the self-sustaining segment to be aligned with the branching position. Since fooling probes cannot compensate for differences in the self-sustaining segment which occur to the left of the branch, we can eliminate  $s_6 \dots s_{11}$  from consideration. And because  $s(a)$  and  $s(b)$  agree at all positions  $s_1 \dots s_4$ , there will not be a branch at any of those locations. ■

If we denote the branching as position 0, then the character immediately to the right of the self-sustaining segment may be called the *offset*, and denoted  $J$ . By this definition  $J \geq 0$ . The sustaining segment consists of  $\nu = \lambda - 1 - J$  characters to the left of the branching position, the branching position itself, and  $J - 1$  characters to the right of the branching. Each of the characters to the left of the branching is fully constrained: when  $s(a)$  is aligned with  $s(b)$ , the first  $\nu$  characters of the two strings must all agree. The character aligned with the branching position ( $s_k$ ) must be supported by fooling probes. And the characters to the right of the branching position must either match (with probability  $1/4$ ) or be supported fooling probes. These rules allow us to calculate the probability of Mode 2 failures as a function of the offset  $J$ .

We return again to the simplest Mode 2 failure, occurring when  $J = 0$ . In this case, there are two identical strings  $s(a)$  and  $s(b)$  with length  $\geq \lambda - 1$  at two locations  $a \neq b$  in the target sequence. The branching position is the first site of a disagreement after the self-sustaining segment. The probability of this event is approximately

$$\frac{m^2}{2} \left( \frac{1}{4^{\lambda-1}} \right) \frac{3}{4} = \frac{3m^2}{2 \cdot 4^\lambda} \quad (4.3)$$

The factor  $\frac{m^2}{2}$  is derived from the fact that there are  $\binom{m}{2} \approx m^2$  possible locations for the identical strings, and their positions are interchangeable. The self-sustaining segment  $s(b)$  consists of  $\lambda - 1$  characters which agree with the  $s(a)$ , and each of these  $\lambda - 1$  symbols match with probability  $1/4$ . Finally, the branching position itself is selectable in 3 out of 4 ways, adding another factor of  $3/4$ .

When  $J > 0$ , the analysis becomes significantly more complex. The positions of the homologous strings  $s(a)$  and  $s(b)$  are not interchangeable since the sequences *preceding*  $s(a)$  and  $s(b)$  determine the specific fooling probes required. Thus, there are about  $\binom{m}{2} \approx m^2$  ways of selecting the positions of the two strings. The probability of a Mode 2 failure with an offset of  $J$  can be expressed as  $m^2 \pi_J$ , for some coefficient  $\pi_J$ , so the total probability of encountering a Mode 2 failure using any probing pattern is given by the following expression, where  $\pi_J$  is a function of the probing pattern used.

$$P_2 = m^2 \sum_{J=0}^{\lambda-1} \pi_J$$

A slight modification to Equation 4.3—the factor of  $m^2$  must be removed—yields the first coefficient,

$$\pi_0 = \frac{3}{2 \cdot 4^\lambda} \quad (4.4)$$

For the remaining coefficients, a little more thought is required. The  $\nu = \lambda - 1 - J$  positions prior to the branching are *fully constrained* positions:  $s(a)$  and  $s(b)$  must match at each of those locations, with probability  $1/4$ . The branching position and the  $J - 1$  characters after the branching position must either match or be compensated for by fooling probes. For a particular offset  $J$ , we must consider whether or not a fooling probe is required (with its right-most symbol aligned) for each position  $0 \leq i < J$ .

The branching position is position 0, and always requires a fooling probe, since it is (by definition) the site of a disagreement between  $s(a)$  and  $s(b)$ . Unfortunately,

for positions  $i > 0$ , the we must treat direct and reverse probes independently. In general, a fooling probe is required at  $i$  if the probe terminating at  $i$  samples any site of disagreement between  $s(a)$  and  $s(b)$ , including the branching position. However, the different structure of direct and reverse probing attens requires that the calculation of the coefficients  $\pi_J, J > 0$  must be performed separately; we will examine reverse probes first.

#### 4.4.1 Reverse Probes

The calculation of the first few coefficients  $\pi_J, 1 \leq J \leq s$  is quite simple. In these cases, there are  $\nu = \lambda - 1 - J$  constrained positions prior to the branching, and each position agrees with probability  $1/4$ . Furthermore, since each of the first  $s$  shifts after the ambiguous extension samples the branching position, a fooling probe is required at each position  $0 \leq i < J$ , and so  $J$  fooling probes are required when  $J \leq s$ . The branching position itself may be selected in 3 out of 4 ways, so when  $1 < J \leq s$

$$\begin{aligned}\pi_J &= \frac{1}{4^\nu} \cdot 3\alpha \cdot \alpha^{J-1} \\ &= \frac{3\alpha^J}{4^\nu}\end{aligned}\tag{4.5}$$

Unfortunately, when  $J > s$ , such a simple analysis is not possible. Since a fooling probe is required at position  $i$  if any sampled position  $0 \leq j \leq i$  is the site of a mismatch between  $s(a)$  and  $s(b)$ , we must consider the likelihood that *all* sampled positions  $i > 0$  match.

**Example 4.6.** Using (3,2)-direct probes, the algorithm encounters the following situation, where  $s(a) = \text{AA[C]TG(G)AC}$  and  $s(b) = \text{AA[A]TG(C)AC}$ . There are two disagreements between the two homologous strings; the first at  $s_3$ , corresponding to the initial branching, and the second at  $s_6$ . Note that in this example,  $\lambda = 9$ ,  $J = 6$  and  $\nu = 2$ .

The following figure shows that although there are only two positions of disagreement between  $s(a)$  and  $s(b)$ ,  $J = 6$  fooling probes are required to compensate for those mismatches:

Path Tree																			
									0	1	2	3	4	5	6	7	8	9...	
...	G	A	C	T	T	A	C	(A	A	[C]	T	G	{G}	A	C	C	A	G	A...
										[A]	T	G	{A}	A	C)	T	A	T	G...
Probes																			
1	A	.	.	T	.	.	A	A	[A]										
2		C	.	.	A	.	.	A	[A]	T									
3			T	.	.	C	.	.	[A]	T	G								
4				T	.	.	A	.	.	T	G	{A}							
5					A	.	.	A	.	.	G	{A}	A						
6						C	.	.	[A]	.	.	{A}	A	C					

positions  $1 \leq i \leq J$  require fooling probes. For the vast majority of such convolutions, we find that  $J$  fooling probes are required to compensate for the sites of disagreement. This seems to imply that the value of  $\pi_J$  for  $J > s$ ,

$$\pi_J \approx \frac{3\alpha^J}{4^\nu}$$

However, this expression does not give a very good estimate for the probability of Mode 2 failure. We observe that the few convolutions which require fewer than  $J$  fooling probes have a much higher relative probability of causing failure. A permutation of  $\rho^J$  which requires  $i$  fewer fooling probes is  $(1/\alpha)^i$  times as likely to result in a Mode 2 failure. The permutations of  $\rho^J$  with the least required fooling probes contribute most of the weight to the multiplier  $\pi_J$ . In order to arrive at a reasonable approximation for the probability of Mode 2 failures, we make the assumption that we can calculate  $\pi_J$  as if  $(\theta \cdot J)$  fooling probes are required for a given offset  $J$ , where  $\theta \leq 1$ . This gives us the expression

$$\pi_J = \frac{3\alpha^{\theta J}}{4^\nu} \quad (4.6)$$

The value of coefficient  $\theta$  is not constant. We expect  $\theta$  to increase as  $\alpha$  increases, since smaller values of  $\alpha$  make it less likely that any fooling probe will be found in the spectrum, and the permutations of  $\rho^J$  that require fewer fooling probes contribute even more disproportionately to the multiplier  $\pi_J$ . Furthermore, we expect  $\theta$  to decrease as  $J$  increases. This is due to the relative proportion of natural to universal bases which fall to the right of the branching position. At the outset, when  $J \leq s$ ,  $\theta = 1$ . As  $J$  increases, more and more universal bases fall to the right of the branching, and we expect that fewer fooling probes will be required. Computer analysis suggests that as a first approximation,  $\theta = 0.85$  yields a satisfactory match between our analysis and observed results for the (4,4) and (5,3) probing patterns which are most commonly used.

When  $J > \lambda$ , the coefficients  $\pi_J$  are negligible, but we include them here for completeness. Using the same approximation as Equation 4.6,  $\theta \cdot (\lambda - 1)$  are required to support the self-sustaining segment in such a case. Moreover, a fooling probe is definitely required at each position  $i = 0 \dots J - \lambda$  (each position from the branching position to the position just before the start of the self-sustaining segment). Thus,

$$\pi_J = 3 \cdot \alpha^{J-\lambda+1} \cdot \alpha^{\theta(\lambda-1)}$$

In order to arrive at a readable estimate for the probability of Mode 2 failure, we use Equation 4.6 for all positions  $1 < J \leq s$ , sacrificing only a minor degree of precision. Furthermore, we ignore the contribution of the coefficients  $J > \lambda$ , so the probability of a Mode 2 failure using an  $(s, r)$ -reverse probing pattern calculated using Equation 4.4 for  $\pi_0$  and 4.6 for  $\pi_1 \dots \pi_J$ :

$$\begin{aligned}
 P_2 &= m^2 \sum_{J=0}^{\lambda} \pi_J \\
 &= m^2 \left( \frac{3}{2 \cdot 4^\lambda} + \frac{3\alpha}{4^{\lambda-2}} \cdot \frac{1 - (4\alpha^\theta)^\lambda}{1 - 4\alpha^\theta} \right) \\
 &= \frac{3m^2}{4^{\lambda-2}} \left( \frac{1}{32} + \alpha \frac{1 - (4\alpha^\theta)^\lambda}{1 - 4\alpha^\theta} \right)
 \end{aligned} \tag{4.7}$$

Figure 4.2 displays a comparison between the predicted and observed rates of Mode 2 failure for (4,4)- and (5,3)-direct probing patterns. The analytical failure estimates show are in quite close agreement with the observed values.

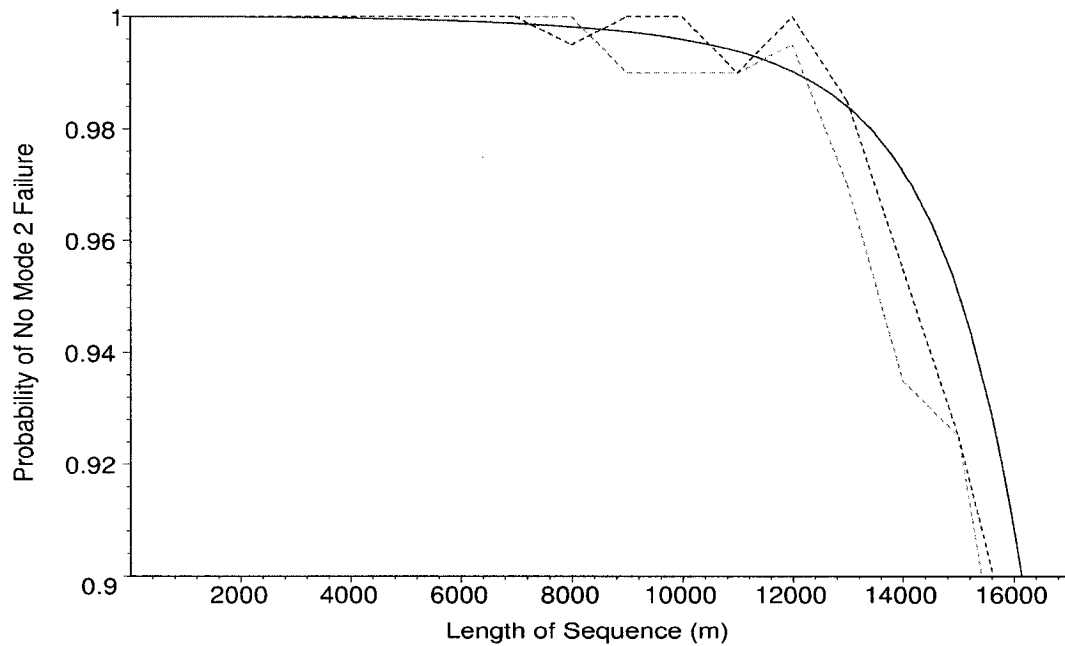


Figure 4.2: Predicted and observed rates of Mode 2 failure for (4,4)- and (5,3)- reverse probes. Note that the predicted probability of failure is the same for both probing patterns. The solid line represents the predicted values. The dotted lines show the observed values for (4,4)-probes (upper line) and (5,3)-probes (lower line).

#### 4.4.2 Direct Probes

As for reverse probing patterns, the calculation of the first  $s$  coefficients  $\pi_J, 1 \leq J \leq s$  is straightforward. There are still  $\nu$  constrained positions to the left of the branching position, and the branching position itself may be chosen in three of four ways. However, a fooling probe is required at position  $1 < i < J$  if and only if  $i$  is the site of a disagreement between  $s(a)$  and  $s(b)$ . An agreement occurs with probability  $1/4$ , and any disagreement must be compensated for by a fooling probe. Since these two events are independent, each position  $1 < i < J$  contributes a factor of

$$\frac{1}{4} + \frac{3}{4}\alpha = \frac{1+3\alpha}{4}$$

and the coefficients  $\pi_J$ , for  $1 \leq J \leq s$  is just

$$\begin{aligned} \pi_J &= \frac{1}{4^\nu} \cdot 3\alpha \cdot \left( \frac{1+3\alpha}{4} \right) \alpha^{J-1} \\ &= \frac{3\alpha^J}{4^\nu} \cdot \left( \frac{1+3\alpha}{4} \right) \end{aligned} \tag{4.8}$$

For  $J \geq s$ , the we could simply use the same method of estimating the total number of fooling probes required as we used for reverse probes. However, we can arrive at a more accurate estimate of the values of  $\pi_J$  by adopting a different approach.

When we consider whether or not a fooling probe is required at position  $i \leq J$ , we are actually asking whether or not the query performed at depth  $i$  in the path tree requires a fooling probe. However, this question assumes that the path tree has, in fact, been extended to at least depth  $i - 1$ . This means that at all depths  $0 \dots i - 2$ , either no fooling probe was required (because  $s(a)$  matched  $s(b)$  at all of the sampled positions) or a fooling probe was found (if there was at least one mismatch between  $s(a)$  and  $s(b)$ ).

In explanation, we will consider a detailed example. Assume that while attempting to reconstruct a target sequence using (3,2)-direct probes, the sequencing algorithm encounters a *potential* Mode 2 failure. We know that there is a string  $s(b)$  which is homologous to the string  $s(a)$  with an offset of  $J = 6$ , but we do not yet know the exact identity of the string  $s(b)$ , nor whether or not the required fooling probes are present in the spectrum. We assume that the spurious path has been extended to depth 3 already. If we restrict our example to only a pair of paths in the tree,



corresponding to the correct path  $s(a)$  and the spurious path containing the self-sustaining segment  $s(b)$ , then there are two possible situations that can be encountered when the sequencing algorithm attempts to extend the tree to depth 4, as shown in the following table. The two situations are labeled *Case a* and *b*. In the following table, the character ‘V’ is used to represent a character *other* than ‘T’ (i. e. ‘A,’ ‘C,’ or ‘G’)<sup>3</sup>, while ‘X’ represents an unknown nucleotide:

										0	1	2	3	4	5	6	7	8	9...	
	...	G	A	C	T	T	A	C	(A	A	[C]	T	T	G	A	C	C	A	G	A...
<i>Case a</i>											[A]	T	T	A	X	X)				
<i>Case b</i>											[A]	V	V	A	X	X)				
Probe 1							A	C	A	.	.	T	.	.	X					
Probe 2							A	C	A	.	.	V	.	.	X					

In *Case a*, position 1 corresponds to the location of an agreement between  $s(a)$  and  $s(b)$ . The probe required to extend the spurious path at this depth has the form of Probe 1 in the above table. If  $s(a)$  and  $s(b)$  also agree at position 4—an event which has probability  $1/4$ —then no fooling probe is needed at depth 4. However, in *Case b*, position 1 is the site of a mismatch between  $s(a)$  and  $s(b)$ . In this situation, a fooling probe of the form of Probe 2 is needed at depth 4 whether or not position 4 is the site of a disagreement between the two homologous strings.

Now we calculate the probability of *Case a* and *b*. At first glance, it appears that position 1 is the location of an agreement with probability  $1/4$ . However, at depth 1, there *must* either have been a match or a fooling probe, or the spurious path would not have been extended to depth 3. The total probability that either a match or a fooling probe occurred is  $\frac{1+3\alpha}{4}$ . Thus, the probability that *Case a* is true is just the likelihood that position 1 is the site of an agreement between  $s(a)$  and  $s(b)$ , given that the spurious path *was* extended at depth 1,

$$\frac{\frac{1}{4}}{\frac{1+3\alpha}{4}} = \frac{1}{1+3\alpha}$$

<sup>3</sup>In actuality, ‘V’ is the IUPAC-IUB-GCG ambiguity code for ‘A, C or G’; meaning that ‘V’ is in a sense the *official* character code to use.

For convenience, we will let  $\gamma = \frac{1}{1+3\alpha}$ . Now, since the right-most position sampled by the query matches the correct sequence with probability  $1/4$ , we know that at depth 4 the probability that no fooling probe is required is  $\gamma \cdot \frac{1}{4}$ . It follows that the probability that the spurious path gets extended at depth 4 is

$$\frac{1}{4}\gamma + \left(1 - \frac{1}{4}\gamma\right)\alpha = \frac{\gamma + \alpha(4 - \gamma)}{4}$$

Assuming that the spurious path *does* get extended to depth 4, an identical argument can be made for the probability that it will be extended to depth 5:

											0	1	2	3	4	5	6	7	8	9...
	...	G	A	C	T	T	A	C	(A	A	[C]	T	T	G	A	C	C	A	G	A...
Case a											[A]	T	T	A	X	X)				
Case b											[A]	V	V	A	X	X)				
Probe 1								C	A	A	.	.	T	.	.	X				
Probe 2								C	A	A	.	.	V	.	.	X				

At depth 5, we consider only whether there position 2 is the location of an agreement between  $s(a)$  and  $s(b)$ , and the probability that the spurious path is extended to depth 5 is again  $\frac{\gamma + \alpha(4 - \gamma)}{4}$ .

If a query samples more than one position to the right of the branching position, then each sampled position agrees with the correct sequence with probability  $\gamma$ . Thus, a query using a (3,3)-direct probing pattern at depth 7 samples all paths in the shaded positions: ‘NNN...N...N...?’. Since there are two sampled positions in the query, the probability that a spurious path is extended by a query of this form is

$$\frac{1}{4}\gamma^2 + \left(1 - \frac{\gamma^2}{4}\right)\alpha = \frac{\gamma^2 + \alpha(4 - \gamma^2)}{4}$$

Recall that in Section 4.4.2, we analyzed the direct-probe queries in terms of  $r$  regions. The queries in each region sample a constant number of positions to the right of the branching. The 4 regions in a direct (4-4)-probing pattern are shown here. This table has been modified slightly from the original in Section 4.4.2 so that the first row shows the *offset* at which the character in that column is aligned with the branching position, and the third line shows the regions themselves.

Depth	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	X	X	X	X	.	.	.	X	.	.	.	X	.	.	.	X	.	.	.	X
Region							4	-		3	-		2	-			1			-

For ease of notation, we denote by  $\delta_i$  the probability that a spurious path is extended by a query which samples  $i - 1$  positions to the right of the branching, so

$$\begin{aligned}\delta_i &= \frac{1}{4}\gamma^{i-1} + \left(1 - \frac{\gamma^{i-1}}{4}\right)\alpha \\ &= \frac{\gamma^{i-1} + \alpha(4 - \gamma^{i-1})}{4}\end{aligned}$$

Queries in region 1 do not sample any positions to the right of the branching, and so they agree with the correct sequence with probability  $\frac{1}{4}\gamma^0 = \frac{1}{4}$ , and the spurious path is extended with probability  $\delta_1$ . Queries in region 2 sample one position to the right of the branching, so they extend a spurious sequence with probability  $\delta_2$ . Generally, queries in region  $i$  sample the post-branch sequence at  $i - 1$  locations, so they will extend a spurious path with probability  $\delta_i$ . Now we may calculate the coefficients  $\pi_J$ .

To recap, there are  $\nu - 1$  characters to the left of the branch that agree with probability  $1/4$ . The branching position may be chosen in three of four possible ways. There are  $\lfloor \frac{J}{s} \rfloor$  complete regions to the right of the branch, each of which consists of  $s - 1$  queries that extend the spurious sequence with probability  $\delta_{i-1}$ . There is one query adjacent to each complete region which samples the branching position, and therefore requires a fooling probe. And finally, the last  $(J \bmod s)$  positions fall into region  $\lfloor \frac{J}{s} \rfloor + 1 = \lceil \frac{J}{s} \rceil$ . The coefficients  $\pi_J$  for  $J = 1 \dots rs$  are

$$\begin{aligned}\pi_J &= \frac{3}{4} \cdot \frac{1}{4^{\nu-1}} \cdot \delta_{J \bmod s}^{\lceil \frac{J}{s} \rceil} \cdot \prod_{i=1}^{\lfloor \frac{J}{s} \rfloor} \alpha \delta_i^{s-1} \\ &= \frac{3}{4^\nu} \cdot \delta_{J \bmod s}^{\lceil \frac{J}{s} \rceil} \cdot \prod_{i=1}^{\lfloor \frac{J}{s} \rfloor} \alpha \delta_i^{s-1}\end{aligned}\tag{4.9}$$

When  $J \geq rs$ , the coefficients are a little simpler, since all  $r$  regions are complete,

and we just have an additional  $J \bmod s$  positions which all require a fooling probe.

$$\begin{aligned}\pi_J &= \frac{3}{4^\nu} \cdot \alpha^{J \bmod s} \cdot \prod_{i=1}^r \alpha \delta^i \\ &= \frac{3}{4^\nu} \cdot \alpha^{(J \bmod s) + r} \cdot \prod_{i=1}^r \delta^i\end{aligned}\tag{4.10}$$

While the total probability of a Mode 2 failure using direct probes is easily calculated computationally, it is rather more complex to perform by hand. If we want a closed-form estimate for the probability of Mode 2 failure using direct probes, we can fall back on the same technique we used for reverse probes. Figure 4.3 displays the results of the two analytical estimates compared with the observed results for direct (4,4)-probes.

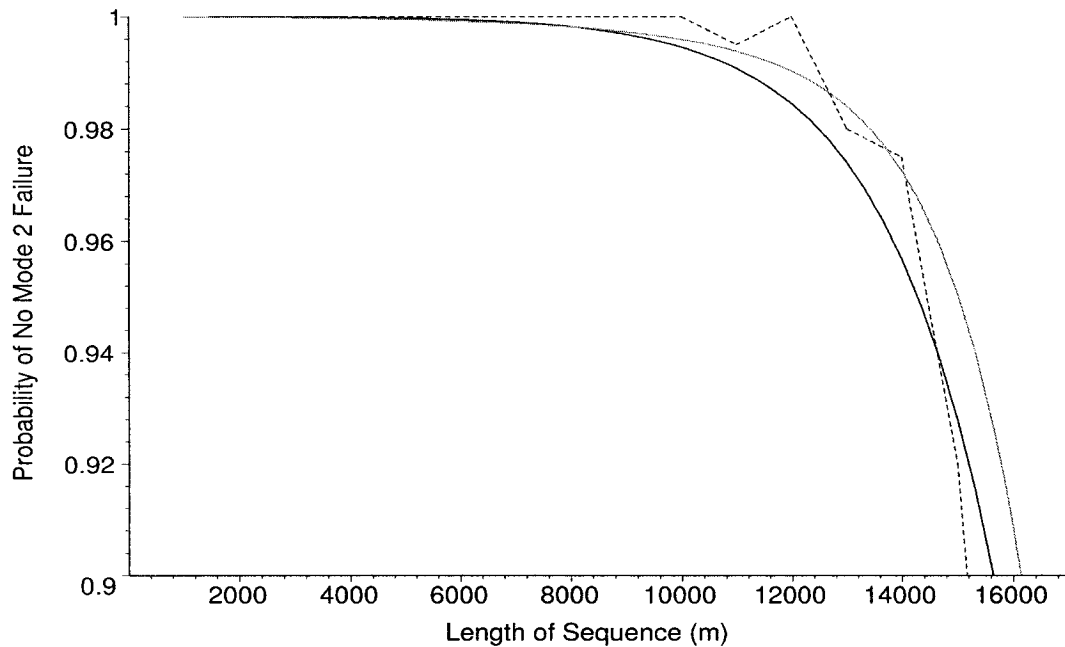


Figure 4.3: Predicted and observed rates of Mode 2 failure for (4,4)- direct probes. Note that there are two analytical estimates for the probability of failure; one using the simpler method used for reverse probes (the upper solid line), and one using the probability calculated by Equations 4.9 and 4.10 (the lower solid line). The dotted line represents the observed values.

## 4.5 Predicted Probability of Success

After spending so much time discussing in detail the events which cause the gapped-SBH algorithm to fail, it is worth noting that it is actually successful most of the time, when using (4,4)- or (5,3)- probing patterns on target sequences with  $m \approx 14000$  or less.

The total probability of failure is simple to calculate. We assume (based on extensive simulations) that Mode 1 and Mode 2 failures are independent, so the total probability of successfully sequencing an  $m$ -character target sequence is

$$e^{-(P_1+P_2)} \approx \left(1 - \frac{P_1 + P_2}{m}\right)^m$$

However, in the interesting range of values, where the the probability of success is close to 1, this can be accurately approximated for the by summing  $P_f = P_1 + P_2$ . Figure 4.4 shows a the predicted probability of sequencing success for all  $(s, r)$ -probes with  $\kappa = 8$ , over a range of  $m$ . The specific predicted and observed performance of (4,4)-probes is shown in Figure 4.5. The predictions and observed sequencing trials both yield a maximum target sequence length of about  $m = 13800$  with  $\epsilon = 0.9$ .

We can also see how the proportion of failures caused by Mode 1 and Mode 2 changes as a function of target sequence length. Initially, Mode 1 failures are dominant, but as  $m$  increases, Mode 2 failures contribute more and more to the total probability of failure. Figure 4.6 illustrates this phenomenon.

Another interesting observation can be made concerning the relative incidence of Mode 1 and Mode 2 failures for different probing patterns. If we plot the proportion of all failures caused by Mode 1, for  $\kappa = 8$  and a range of values of  $s = 4 \dots 8$ , we find that as  $s \rightarrow 8$ , Mode 2 failures dominate. In fact, for the special case  $s = 8$ , which corresponds to an ungapped 8-mer, all sequencing failures are expected to fall into Mode 2. This is to be expected, since a single duplicated  $(\kappa - 1)$ -character subsequence is sufficient to cause a Mode 2 failure when ungapped  $\kappa$ -mers are used as probes. Such a duplicate subsequence is about 4 times as likely to be found as a *single* fooling probe. Figure 4.7 shows a graph of the proportion of failures caused by Mode 1 using different  $(s, r)$ -probing patterns.

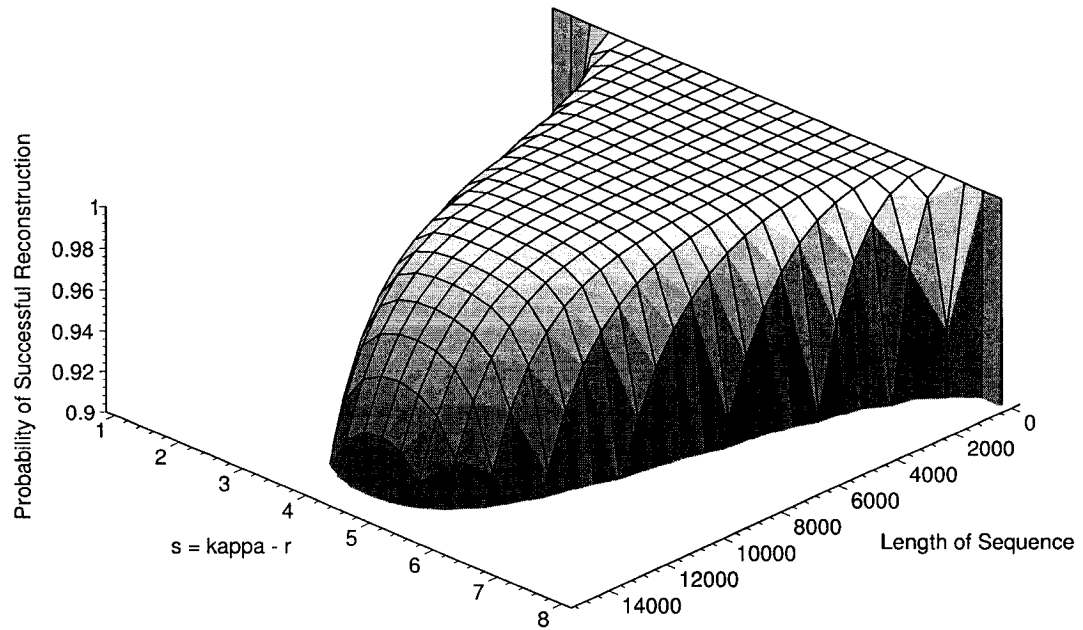


Figure 4.4: Predicted probability of success for all  $(s, r)$ -probes with  $\kappa = 8$ . The best performance is achieved with  $s \approx r$ .

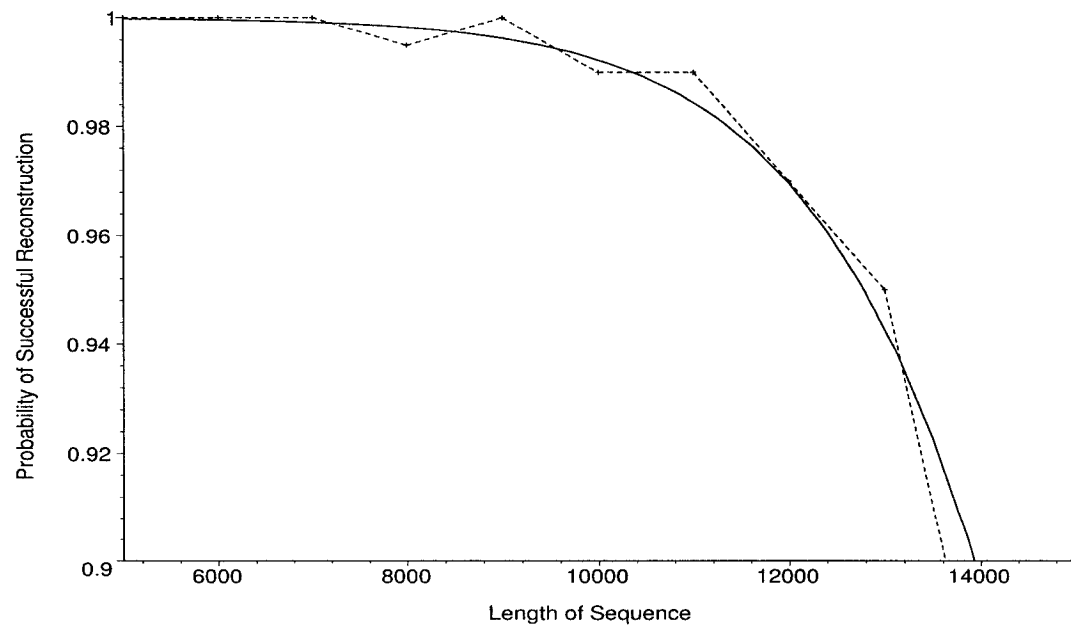


Figure 4.5: Predicted and observed probability of success for  $(4,4)$ -probes. The solid line shows the predicted value; the dotted line the observed behaviour.

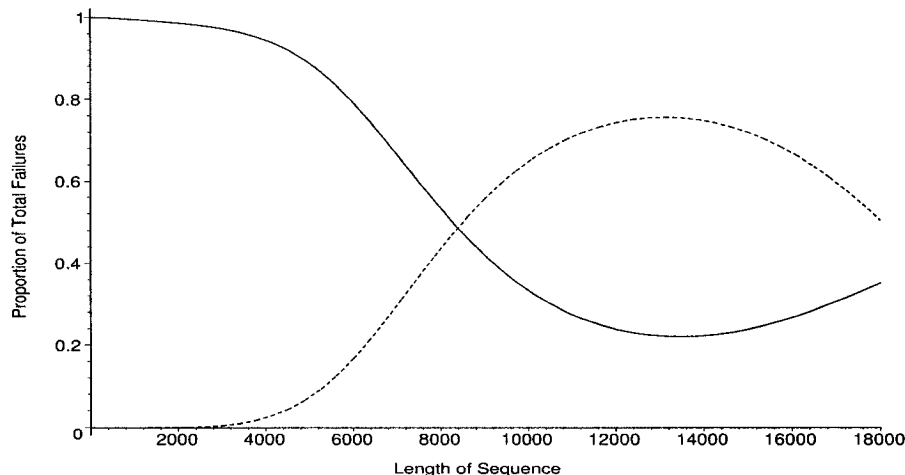


Figure 4.6: Relative proportion of Mode 1 (dotted line) and Mode 2 (solid line) failures using (4,4)-reverse probes.

## 4.6 Measuring Performance on Natural DNA

At the end of the Chapter 2, Figure 2.2 showed the performance of the SBH algorithm on random DNA, and several sources of natural DNA. Here we present a more thorough exploration of the behaviour of the SBH algorithm on a larger variety of natural sources of DNA.

### 4.6.1 Selection of Natural DNA Sources

We wanted to compare the performance of the gapped-SBH algorithm on a variety of organisms. The source of the natural DNA sequences used as sample data in this dissertation were drawn from the public GenBank database of complete genomes. Appendix A contains a complete list of the accession numbers for the sequences used. The organisms selected were chosen from all three major domains of life: eukaryota, eubacteria and archaea. The only real requirement for inclusion was that the genome was long enough to conduct several hundred sequencing attempts with non-overlapping fragments of length  $m = 5000$  (about 2Mbp).

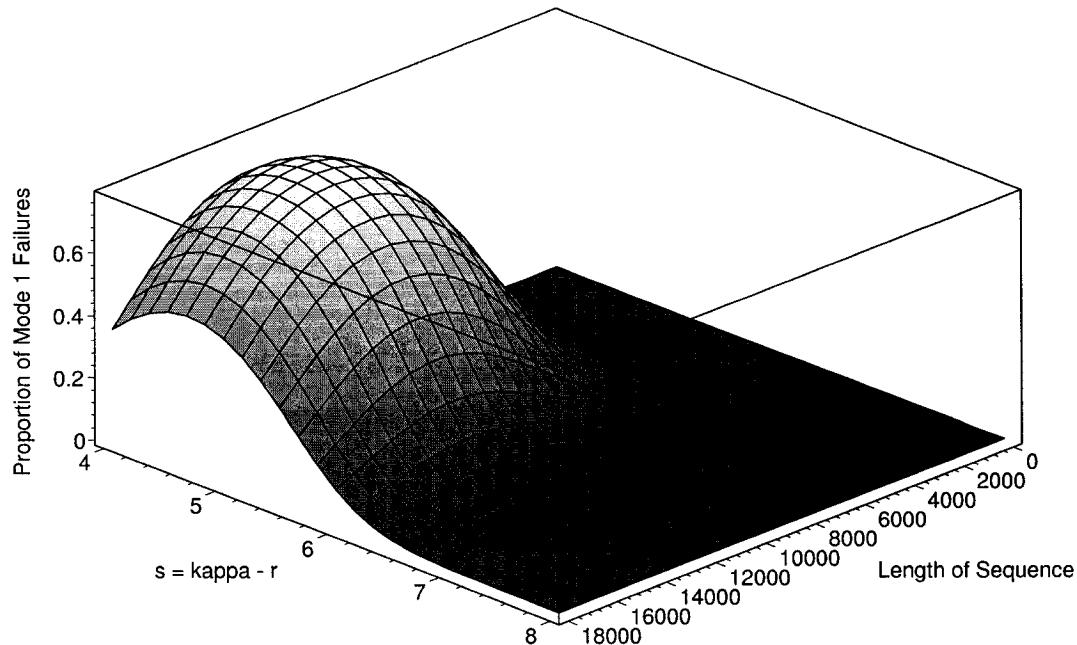


Figure 4.7: Relative proportion of Mode 1 and Mode 2 failures for the family of  $(s, r)$ -probes,  $s = 4 \dots 8$ . Note that 8-nucleotide ungapped probes (8-mers) correspond to the  $s = 8$ , where *all* failures are expected to be caused by Mode 2.

Several different eukaryotes were included, including humans and yeast. In eukaryotic organisms, there is a qualitative and quantitative difference between the *coding*—the sections of DNA that code for proteins—and *non-coding* regions. The SBH algorithm can be expected to perform differently when reconstructing coding vs. non-coding regions. While the coding regions are probably of the most interest to researchers, it may not be possible to extract only those segments of DNA, if they are even known. To cover all bases, we conducted sequencing trials using both the full genome, and using only the extracted coding regions themselves. Note that coding sequences are actually only the subset of mRNA that is actually translated into proteins; they exclude mRNA that is excised by splicing as well as all intergenic DNA which is never transcribed into mRNA in the first place.

Several different bacterial (prokaryotic) genomes were included as well. Unlike eukaryotes, prokaryotes have a single, circular chromosome. These organisms tend to have a very low degree of intergenomic, or ‘junk’ DNA, and so we expect little difference in performance when using the complete genome, or just coding regions.



For this reason, prokaryote genomes were sequenced *as is*, without attempting to extract only the genes themselves.

Finally, we have the archaea, the most alien form of life. When there were enough annotated coding sequences to produce sufficient data, we extracted coding sequences from the included archaea. This was possible for one of the two archaea that were used.

A brief description of each included organism follows:

### **Eukaryota**

- *Arabidopsis thaliana* is widely used as a model organism in plant biology. It is a small flowering plant that is a member of the mustard family. Chromosomes I and III were both included, and each chromosome was processed to extract the coding sequences.
- *Homo sapiens* should be familiar to all readers. Two chromosomes (3 and 11) of the human genome were included. Both were processed to extract the coding sequences.
- *Plasmodium falciparum* is the parasite that causes malaria. It is known for having an unusually high ‘AT’ content. We included chromosome 3, which was also processed to extract the coding sequences.
- *Saccharomyces cerevisiae* is common baker’s yeast, and the only fungus that was included. Chromosome IV, and the coding sequences extracted therefrom, were both used.

### **Eubacteria**

- *Escherichia coli* is one of the most common types of bacteria, and has been extensively used as a model organism for genetics research. (“Once we understand the biology of *Escherichia coli*, we will understand the biology of an elephant.” —Jacques Monod.) It was the second organism to be completely sequenced, and was completed in 1996.
- *Haemophilus influenzae* was mistakenly identified in 1890 as the cause of the disease influenza, and named accordingly. It was the first living organism to have its full genome narrowly beating out *e. coli*.

- *Salmonella typhimurium* causes salmonellosis in humans, which is contracted mainly through the consumption of undercooked, contaminated meat, poultry and eggs. Its genome was not sequenced until 2001.
- *Streptococcus pneumoniae* completes our selection of bacterial DNA sources. It is the most common cause of meningitis, bacterial pneumonia, and ear infections.

## Archaea

- *Methanosarcina acetivorans* is an archaea which produces methane from (among other things) acetate. The complete genome for strain C2A was made available in March 2002, and we include both the complete genome and the extracted coding sequences.
- *Sulfolobus solfataricus* is a hyperthermophilic archaea found in sulfur-rich acidic hot springs. It grows optimally at temperatures ranging from 70 to 90 degrees Celsius and pH values from 2 to 4. The included genome for this organism was completed in October 2001.

Most of the listed organisms will be referred to by the initial of their genus and full species name (i. e. *e. coli*). However, *salmonella* and *streptococcus* may be used for the specific examples of these genii included in this thesis, and human DNA may simply be called that. A specific chromosome from an organism is labeled ‘chr.  $i$ ’, where  $i$  is the number of the chromosome. Finally, we use the shorthand annotation ‘CDS’ to denote coding sequences, as used in the GenBank files.

### 4.6.2 Observed SBH Results for Natural DNA

To derive estimates of SBH performance on natural DNA, simulations were conducted using data drawn from the public GenBank database. For each fragment length, 400 unique, non-overlapping fragments were drawn from the complete genome. Using (4,4)-reverse probes, an attempt was made to reconstruct each fragment from its spectrum. The approximate maximum length  $m$  of sequences for which  $\epsilon = 0.9$  of the sample fragments could be successfully reconstructed using reverse (4,4)-probes

is presented in the following table. Recall that the maximum  $m$  for random sequences is 13800.

Sequence Source	Maximum $m$
<b>Eukaryotes</b>	
<i>a. thaliana</i> chr. I	400
<i>a. thaliana</i> chr. I CDS	1600
<i>a. thaliana</i> chr. III	900
<i>a. thaliana</i> chr. III CDS	1700
<i>h. sapiens</i> chr. 3	700
<i>h. sapiens</i> chr. 3 CDS	1400
<i>h. sapiens</i> chr. 11	800
<i>h. sapiens</i> chr. 11 CDS	1600
<i>p. falciparum</i> chr. 3	100
<i>p. falciparum</i> chr. 3 CDS	400
<i>s. cerevisiae</i> chr. IV	1300
<i>s. cerevisiae</i> chr. IV CDS	2300
<b>Eubacteria</b>	
<i>e. coli</i>	5400
<i>h. influenzae</i>	2800
<i>s. typhimurium</i>	6000
<i>s. pneumoniae</i>	2400
<b>Archaea</b>	
<i>m. acetivorans</i>	400
<i>m. acetivorans</i> CDS	1700
<i>s. solfataricus</i>	3000

Complete performance graphs for almost all of the above data sources are presented on the following pages. However, we omit *a. thaliana* chromosome I, and

human chromosome 3, for the simple reason that these chromosomes exhibit nearly identical performance curves as the other included chromosome from the same organism. We also omit *p. falciparum* and the full genome (non-CDS) *m. acetivorans* DNA, since performance on these sequences is so poor that it is irrelevant.

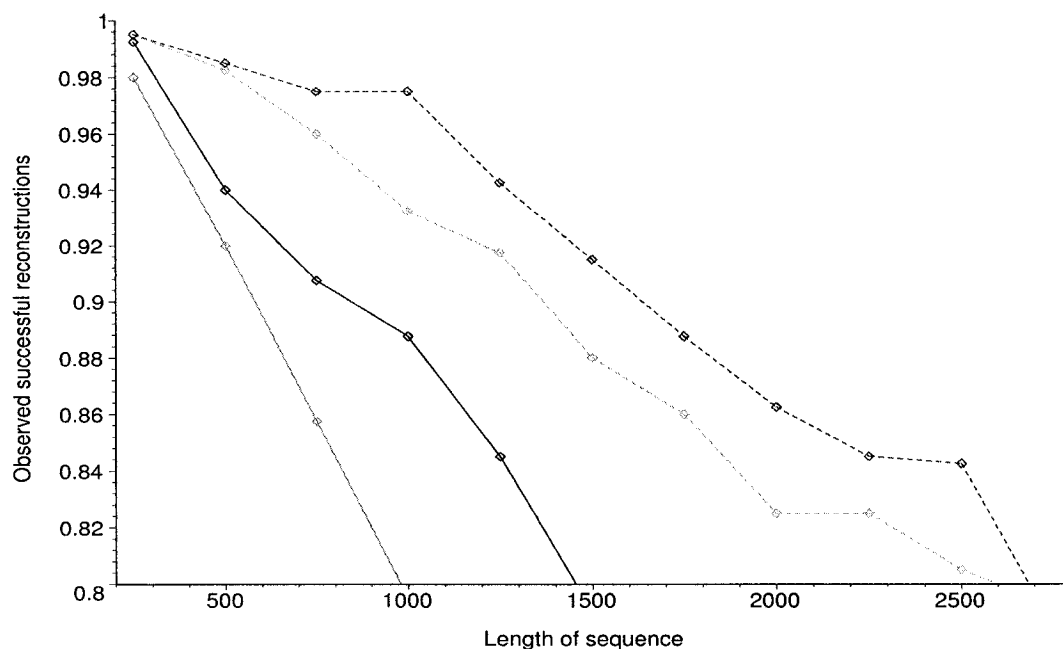


Figure 4.8: Observed proportion of sequences successfully reconstructed by the basic SBH algorithm using reverse (4,4) probes. The four curves correspond to *a. thaliana* chromosome III coding sequences, human chromosome 3 coding sequences, the full *a. thaliana* chromosome III, and the full human chromosome 3, from right to left respectively.

### 4.6.3 Some Comments on SBH with Natural DNA

It is not surprising that the performance of our method degrades significantly when the target sequences are drawn from natural DNA. Both Mode 1 and Mode 2 failures are more common, although Mode 2 failures show a much higher relative increase in frequency. But some types of failure appear to occur only when sequencing natural DNA.

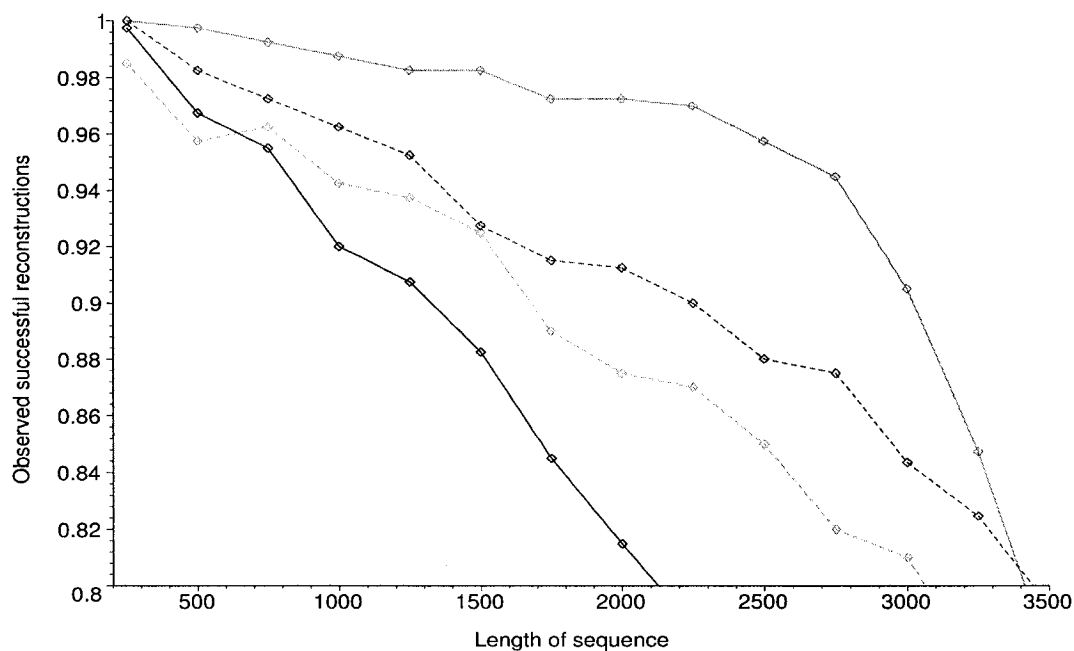


Figure 4.9: Observed proportion of sequences successfully reconstructed by the basic SBH algorithm using reverse (4,4) probes. The four curves correspond to *s. solfataricus*, *s. cerevisiae* chromosome IV coding sequences, the full *s. cerevisiae* chromosome IV, and *m. acetivorans* coding sequences, from right to left respectively.

In contrast to the failures observed for random data, when attempting to reconstruct natural DNA sequences, the algorithm frequently fails because the breadth-bound  $B$  is reached. However, increasing  $B$  from 256 to 2048 generally results in a negligible increase in performance: typically less than 2%. If nothing else, this indicates that the types of failures which account for much of the performance difference are qualitatively different from the two modes observed in random DNA.

A great deal of work (see for example, [F+94, LLY00]) has gone into calculating the information content, or entropy of DNA. While the general results indicate that DNA appears to be nearly random, most of the techniques for estimating information content assume that the data source being observed is stationary. This assumption is inherently flawed. Introns are qualitatively different from exons: one type of DNA is used to code for proteins, and the other is not. Coding regions differ from intergenic regions even more drastically. Any measure of information or entropy which assumes a memoryless model is bound to misrepresent the data.

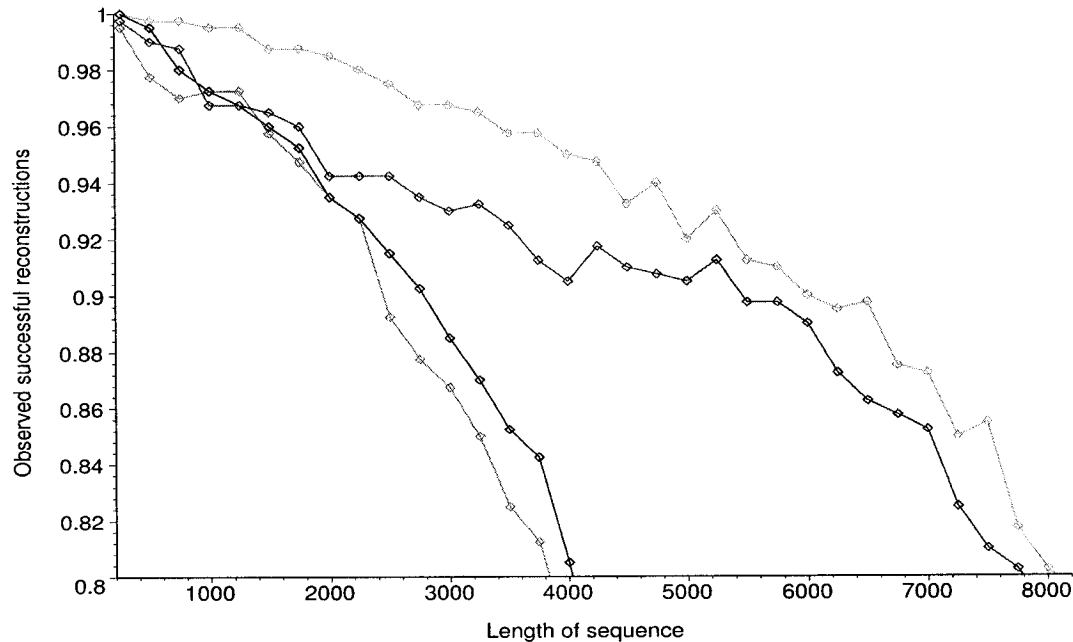


Figure 4.10: Observed proportion of sequences successfully reconstructed by the basic SBH algorithm using reverse (4,4) probes. The four curves correspond to *salmonella*, *e. coli*, *h. influenzae*, and *s. pneumoniae*, from right to left respectively.

Furthermore, exons themselves cannot be random sequences: the proteins into which they are translated have specific functions which must be encoded in their sequences. There are also higher-order patterns in DNA caused by natural processes of infection by viruses, recombination of DNA molecules, ‘jumping genes’ called transposons, and the non-random effect of natural selection.

A simple measure of the ‘randomness’ of DNA can be obtained by estimating the entropy of the DNA sequence over some window of characters. By examining the nearest 100 or 1000 bases, and assuming (falsely) that they issue from a stationary source, we can calculate the information content of the sequence, in (*bits/base*). Any deviation from the 2 bits/base required for a uniformly random sequence indicates that some bases are more likely to occur within that segment of DNA than others. Since any deviation from a uniform random generating process increases the chance of finding fooling probes, lower estimated entropy values indicate that sequencing failures are more likely. Unfortunately, in terms of predicting the occurrence of fooling probes within lower-entropy sequences, we cannot be any more precise.

One organism has a startlingly high incidence of A and T bases, with correspondingly low entropy: *plasmodium falciparum*. Over 80% of all of the nucleotides in the genome of this parasite are adenosine and thymine. In this case, SBH is nearly useless, at least in its present form. Experimentally, the longest fragments which can be sequencing using gapped-SBH are only about 30bp in length. SBH fares much better on the DNA of other organisms, but the case of *p. falciparum* serves to illustrate the worst-case scenario.

Certain specific features of natural DNA which cause sequencing failures can be identified, recognized and targeted. For instance, microsatellites and tandem repeats can cause problems for the SBH algorithm. In both cases, a short sequence (typically 2-10 bases) of DNA is repeated up to several hundred times in succession. These locally repeating regions produce sequencing failure, due to explosive branching in the path tree. A more detailed explanation of these repeating segments, along with a method for recognizing them and continuing the sequencing process, is contained in Chapter 6.

Appendix C contains the beginnings of a fairly detailed model of the repeat structure of natural DNA sources, and Chapter 5 describes a method for recovering from some the failures common to both random and natural DNA sources. However, simply by applying the standard gapped-SBH algorithm, we can sequence DNA fragments of certain organisms which are more than an order of magnitude longer than that achievable by traditional sequencing methods. In particular, the maximum achievable length of sequences for *e. coli* and *salmonella* is very encouraging.

## Chapter 5

# Resolving Ambiguous Extensions by Polling

When attempting to reconstruct maximum entropy random sequences, all sequencing failures can be grouped into two classes: Mode 1 and Mode 2 failures (refer to Sections 4.2 and 4.3 for a detailed description of each type of failure, respectively). In failure Mode 1, a single ambiguous character is confirmed by  $\kappa$  independent probes scattered throughout the target sequence. In Mode 2, the presence of a  $(\lambda - 1)$ -character transition sequence and a set of fooling probes provokes a real branch in the reconstruction, where both the correct and spurious paths correspond to an actual substring of the target sequence. In both cases, the spectrum is insufficient to select between the correct and spurious paths. Fortunately, the spectrum itself is not the only source of information available to the reconstruction algorithm.

When either type of sequencing failure occurs, it is possible to use the putative sequence—the  $\mu$ -character fragment of the target sequence which has already been reconstructed—to gain additional information. Specifically, there are at least  $\kappa$  fooling probes in the spectrum which confirm a spurious extension (more in the case of a Mode 2 failure); by counting the number of probes in each extension set which can be found in the  $\mu$ -character putative sequence, we can select the correct extension with a high degree of confidence. This process is called *polling*, since it chooses between two alternative extensions by counting the number of *used probes* which confirm either extension. Of course, we must now discuss what it means for a probe to be *used*.



## 5.1 Definition of Used Probes

By the time the branching mode of EXTEND has reached the depth-bound  $H$ , we know that all of the fooling probes necessary to confirm a spurious extension are present in the spectrum: if any of them were not present, then reconstruction would not have failed. Assuming that the fooling probes are scattered randomly along the full length of the complete length- $m$  sequence, a proportion  $\mu/m$  of the fooling probes confirming the spurious extension should be present in the sequence completed thus far. These probes have been *used* in the reconstruction already performed; they are formally defined as follows:

**Definition 5.1.** *A used probe is a probe which is located in the  $\mu$ -character putative sequence.*

When sequencing is commenced with a  $\lambda$ -character seed segment, there can be only a single used probe: the probe which matches the seed segment. Ignoring the possibility of repetitions, each character added to the putative sequence also *uses* one additional probe. Thus, if the putative sequence is  $\mu$  characters long, there are at most  $(\mu - \lambda + 1)$  used probes in the spectrum.

**Example 5.1.** The 22-character putative sequence AGGGTCTAGGTATTGGGAT-TAA has been reconstructed from a spectrum of direct (4,4) probes. At this point there are  $\mu - \lambda + 1 = 22 - 20 + 1 = 2$  used probes: AGGG...A...A...G...T, GGGT...G...T...G...A, and GGTC...G...T...A...A. ■

We expect a fraction  $\mu/m$  of the probes confirming a spurious extension to be *used*. However, due to repetitions of probes within the target sequence, some of the probes which confirm the *correct* extension may be present in the previously completed portion of the sequence (and thus *used*). However, this event is uncommon enough that it is possible to adopt a strategy of choosing between the two extensions by comparing the number of previously used probes which confirm each of them. The extension which is confirmed by the greatest number of used probes can be rejected as spurious. This process is referred to as *polling*.<sup>1</sup>

---

<sup>1</sup>Unlike most polling processes where the candidate with the most votes is the winner, here the candidate with the most votes is the loser.

A probabilistic analysis of the *polling* method for Failure Mode 1 is straightforward:  $\kappa$  probes are required to confirm the single ambiguous character, and so we consider only the  $\kappa$  probes in the extension set for each branch. The analysis for Failure Mode 2 is more subtle, but fundamentally the same. We consider each case in turn.

## 5.2 Polling and Mode 1 Failures

Mode 1 failures, as discussed in Section 4.2, have a very simple characterization. When an unresolvable ambiguous extension occurs and two paths are extended up to the depth bound  $H$ , there is only a single character difference (the initial ambiguous character) between the two paths. After the initial ambiguity, the paths match exactly.

During sequencing runs,  $H$  is typically set to  $\geq 128$ . This contrasts with the length of the probes, which is virtually always  $\lambda \simeq 20$ . ( $\lambda = 20$  occurs for (4,4)- and (5,3)-  $(s, r)$ -probing patterns). The difference between  $H$  and  $\lambda$  is especially important when using the polling method to resolve an ambiguous extension. Recall from the previous chapter that  $H$  must be chosen sufficiently large to prevent failures due to completely spurious sequences supported entirely by fooling probes. Since each character in such a path must be supported by fooling probes, the probability of extending a path to  $n$  characters is an exponentially decreasing function of  $n$ . The value of  $H$  determines the minimum length which must be achieved by such a false path to cause sequencing failure, and so  $H$  is chosen to be sufficiently large to make false paths of this type vanishingly unlikely.

Additional branches may occur subsequent to the initial ambiguous character. To eliminate these additional branches, all paths in the tree are pruned to  $\lambda$  characters (including the initial branching position). The pruning process eliminates all branches which occur at  $t > \lambda$  characters beyond the initial ambiguous character by simply excising them from the tree. It also tends to greatly reduce the total number of paths; usually, there are only 2 paths which remain. By definition, these paths must originate at the root of the tree. Moreover, secondary branches which occur within the first  $\lambda$  characters are subject to a depth bound of  $(H' > H - \lambda)$  characters. Since  $H \geq 128$ ,  $H - \lambda \geq 108$ , which is still large enough to make the probabilistic extension

of totally spurious paths extremely unlikely.

After pruning, each of the  $\lambda$ -character paths in the shortened tree is compared to all of the others. If there are only two paths, then they should differ only in their initial character—the branching position. If there are more than 2 paths in the set—due to additional secondary branches which survived the pruning—then for *some* path  $p_i$ , there should be another path  $p_j, i \neq j$  which is identical to  $p_i$ , save for the initial position. This is the criterion used to recognize a Mode 1 failure during sequencing. Virtually all other cases can be treated as Mode 2 failures, but this will be discussed thoroughly in the next session.

**Definition 5.2.** *A sequencing failure is a Mode 1 Failure if and only if there are at least two paths in the pruned tree, which differ in only the initial character.*

Once a Mode 1 failure has been recognized, it remains to count the number of used probes confirming each path. There are  $\kappa$  probes in the spectrum which sample the ambiguous position. The putative sequence contains  $\mu$  characters, and the length of the target sequence is  $m$  characters, so  $(\frac{\mu}{m} \cdot \kappa)$  used probes confirming the spurious path are expected. If probes were unique within the target sequence, there would be no used probes confirming the correct path, but due to repetitions of probes within the sequence, each probe confirming the correct path may appear in the  $\mu$ -character putative sequence with probability approximately  $(\frac{\mu}{m} \cdot \alpha)$ . The polling algorithm selects the extension with the fewest used probes in the extension set. In the event of a tie (an equal number of fooling probes in the extension set for both paths), no decision can be made, and the sequencing process halts.

To predict the likelihood that the polling mechanism selects the correct extension, it is necessary to predict the probability of finding more used probes confirming the spurious path than the correct path. We will let  $p_j^{(1)}(\mu)$  denote the probability of finding  $j$  used probes confirming a spurious Mode 1 extension after  $\mu$  characters have been sequenced. There are  $\kappa$  probes which sample the ambiguous position, and are thus involved in polling. The probability that a specific fooling probe has already been used is  $\mu/m$ . Therefore, the probability of finding  $j$  used probes for the spurious branch of a Mode  $i$  failure is equal to

$$p_j^{(1)}(\mu) = \binom{\kappa}{j} \left(\frac{\mu}{m}\right)^j \left(1 - \frac{\mu}{m}\right)^{\kappa-j} \quad (5.1)$$

Now we need to consider the probability of finding used probes which confirm the *correct* extension, due to multiplicity of probes. Let  $q_j^{(1)}(\mu)$  be the probability of finding  $j$  used probes confirming the correct extension of a Mode 1 failure. The probability that any one particular probe has already been used is  $\delta = 1 - e^{-\mu/4\kappa}$ , so the probability of finding at least  $j$  used probes for the correct path is

$$q_j^{(1)}(\mu) = \sum_{h=j}^{\kappa} \binom{\kappa}{h} \delta^h (1 - \delta)^{\kappa-h} \quad (5.2)$$

Note that the likelihood of encountering a Mode 1 failure remains constant as  $\mu \rightarrow m$ ; they can occur with equal probability at any point along the target sequence. The likelihood of finding used probes confirming either the correct or spurious extension increases as  $\mu \rightarrow m$ , but the likelihood of selecting the correct extension also increases with  $\mu$ . To calculate the overall probability of success for the polling method, an average over all possible failure points must be taken. Recall that the polling algorithm selects the *incorrect* alternative when more used probes are counted for the correct than for the spurious path, and is unable to choose between the two paths when there are an equal number of used probes for each. In either case, the polling algorithm fails to select the correct path after an ambiguous branch, but sequencing halts only if the number of used probes for each extension is equal. When there are more used probes confirming the correct extension than the spurious one, the algorithm selects the incorrect path and continues. This event leads to *false positive* sequences—sequences which appear to be correct but contain one or more errors—and will be discussed in detail later in this chapter.

The probability that the polling algorithm cannot correctly resolve a Mode 1 failure is:

$$P(\text{poll failure} | \text{Mode 1}) = \frac{1}{m} \sum_{\mu=1}^m \sum_{j=0}^{\kappa} p_j(\mu) \cdot q_j(\mu) \quad (5.3)$$

### 5.3 Polling and Mode 2 Failures

Mode 2 failures, described in detail in Section 4.3, are more difficult to recognize than Mode 1 failures, due to the difficulty of precisely identifying the sustaining segment. However, since there are only two types of sequencing failure, any unresolved ambiguity which cannot be identified as a Mode 1 failure is assumed to be Mode 2. This assumption necessarily groups together standard Mode 2 failures with more complex failure cases. For example, it is possible (though unlikely) that a Mode 2 failure falls within  $\lambda$  positions of a Mode 1 failure. By Definition 5.2, any sequencing failure not specifically recognized as a Mode 1 failure, is treated as a Mode 2 failure.

If it were possible to determine the exact nature of the sustaining segment that prompts the occurrence of a Mode 2 failure, it would also be possible to identify the fooling probes necessary to complete the failure. The probes which are produced by the sustaining segment itself could be ignored, and only the fooling probes would be polled. Since we cannot identify sustaining segment exactly, a different strategy must be used. Sustaining segments which terminate more than  $\lambda$  characters beyond the initial branching are highly improbable, so the polling algorithm uses the set of all probes which contain the initial ambiguous character within their span (and thus have a rightmost character at most  $(\lambda - 1)$  characters beyond branching point). Any probes which are superfluous to the failure—which are not needed to confirm the spurious extension—will be present in both paths, and thus either contribute 1 vote to both poll counts, or contribute to neither count.

The set of probes polled for Mode 2 failures contains  $\lambda$  members: all probes which contain the initial ambiguous position within their span. Other than that minor difference (there are only  $\kappa$  probes polled for Mode 1 failures), the equations for the number of used probes confirming the spurious and correct paths are identical to Mode 1. The probability  $(p_j^{(2)}(\mu))$  of finding  $j$  used probes confirming the spurious branch of a Mode 2 failure is

$$p_j^{(2)}(\mu) = \binom{\lambda}{j} \left(\frac{\mu}{m}\right)^j \left(1 - \frac{\mu}{m}\right)^{\lambda-j} \quad (5.4)$$

And the probability  $(q_j^{(2)}(\mu))$  of finding at least  $j$  used probes confirming the *correct* branch of a Mode 2 failure is

$$q_j^{(2)}(\mu) = \sum_{h=j}^{\lambda} \binom{\lambda}{h} \delta^h (1-\delta)^{\lambda-h} \quad (5.5)$$

With the exception of failures caused by duplicated  $(\lambda-1)$ -character strings, Mode 2 failures are equally likely to occur at any point along the sequence (recall from Section 4.3 that this event corresponds to self-sustaining segment offset of  $J = 0$ ). When there is a  $(\lambda-1)$ -character duplicated subsequence in the target sequence, a Mode 2 failure will occur when the sequencing process reaches *first* of the two identical strings. In such a case, the point of failure is the smaller of two values chosen randomly on  $[1, m]$ . However, in all other cases (when the offset  $J > 0$ ), the positions of the two homologous strings are not interchangeable, so a failure—as for Mode 1—is equally likely at any point along the sequence. Since the event  $J = 0$  accounts for such a small proportion of all Mode 2 failures, we may reasonably ignore the possibility of such an event in our calculations. Thus, the probability of an incorrect polling result for Mode 2 can be expressed as

$$P(\text{poll failure} | \text{Mode 2}) = \frac{1}{m} \sum_{\mu=1}^m \sum_{j=0}^{\lambda} p_j^{(1)}(\mu) \cdot q_j^{(1)}(\mu) \quad (5.6)$$

## 5.4 Overall Performance Improvement

Equations 5.3 and 5.6 can be quite easily combined. First, we define a parameter  $\sigma_i$  to represent the number of probes polled for each failure type;  $\sigma_1 = \kappa$ , and  $\sigma_2 = \lambda$ . Then a combined equation for a Mode  $i$  polling failure can be calculated by the following equation:

$$P(\text{poll failure} | \text{Mode } i) = \frac{1}{m} \sum_{\mu=1}^m \sum_{j=0}^{\sigma_i} p_j^{(i)}(\mu) \cdot q_j^{(i)}(\mu) \quad (5.7)$$

$$= \frac{1}{m} \sum_{\mu=1}^m \sum_{j=0}^{\sigma_i} \binom{\sigma_i}{j} \left(\frac{\mu}{m}\right)^j \left(1 - \frac{\mu}{m}\right)^{\sigma_i-j} \sum_{h=j}^{\sigma_i} \binom{\sigma_i}{h} \delta^h (1-\delta)^{\sigma_i-h} \quad (5.8)$$

Thus, we can conclude that the probability of failure  $\phi(m)$  as a function of sequence length is

$$\phi(m) = P(\text{poll failure}|\text{Mode 1}) \cdot P_1 + P(\text{poll failure}|\text{Mode 2}) \cdot P_2 < P_1 + P_2$$

Solving  $\phi(m) = \epsilon$  would give a quantitative measure of the increase in performance achieved by the polling mechanism. As it has not been possible to find a closed-form solution to the expansion of  $P(\text{poll failure}|\text{Mode } i)$ , a numerical solution must suffice. By averaging over the parameter  $\mu$  for both Mode 1 and Mode 2 failures, and calculating the respective likelihood of each type of failure, it is possible to predict the performance of the algorithm.

The predicted results indicate the proportion of *correct* reconstructions expected. While the standard sequencing method never produces incorrect sequences (sequences which contain one or more errors), the polling mechanism introduces the possibility of selecting an incorrect extension at some point in the sequence. In the case of Mode 1 failures, the result of an incorrect poll result would be limited to a single incorrect character. Polling failures for Mode 2 are more problematic, since they can result in a completely incorrect sequence after the first incorrectly chosen character. On the other hand, incorrect polling results for Mode 2 failures also tend to produce sequences which differ substantially in length from the actual target sequence, allowing such errors to be fairly easily identified.

Figure 5.1 shows the predicted proportion of successful sequencing attempts using the polling method, compared with the observed results over 200 trials for each  $m$ . Using (4,4)-probes on random data, a 20% improvement is observed, which is in excellent agreement with the numerical prediction. Note, however, that the experimental results do not account for *false positive* sequences (sequences which are reported to be correct, but which contain one or more errors). In the laboratory, it would be impossible to differentiate between correct and incorrect sequences, and thus it is unreasonable to subtract the false positive trials from the results observed in simulation. We can, however estimate the number of errors which are expected to occur.

The polling algorithm only fails to produce an answer when there are an equal number of used probes in the extension set for both ambiguous branches. If either

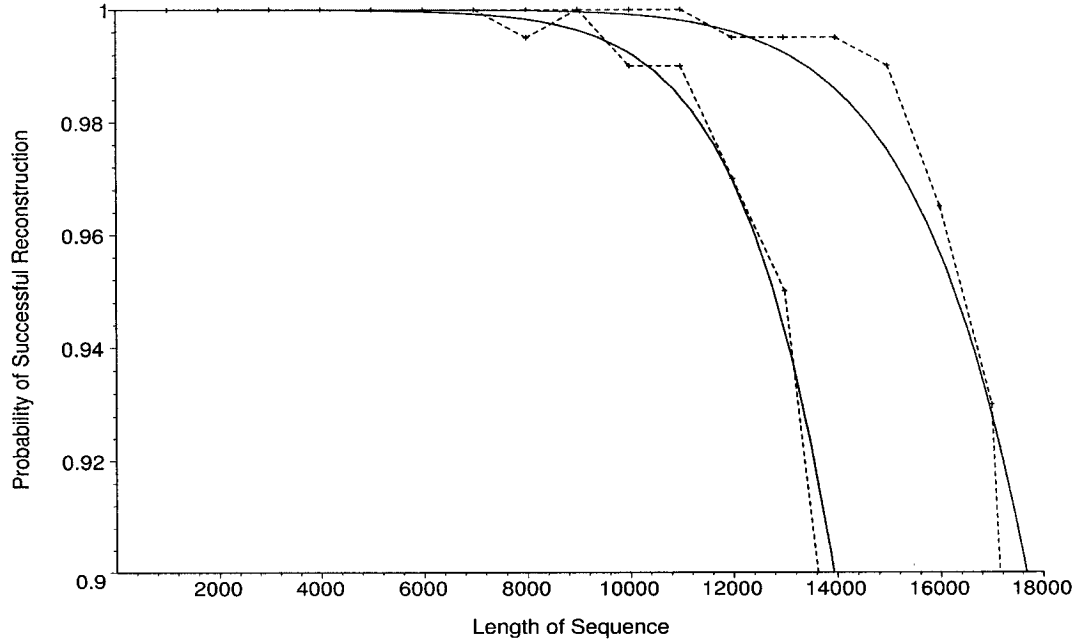


Figure 5.1: The solid lines show the predicted performance of the sequencing algorithm with and without the polling provision; the dotted lines show the observed behaviour. Both the predicted and observed data assume the use of (4,4)-probes.

branch has a numerical advantage, then the branch with the fewest used probes is selected. Thus, errors only occur when the spurious branch is confirmed by *fewer* used probes than the correct branch. We can estimate the probability of this event with a minor modification of Equation 5.7:

$$P(\text{poll incorrect} | \text{Mode } i) = \frac{1}{m} \sum_{\mu=1}^m \sum_{j=0}^{\sigma_i} p_j^{(i)}(\mu) \cdot \sum_{k=j+1}^{\sigma_i} q_k^{(i)}(\mu) \quad (5.9)$$

Note that the last factor in Equation 5.9 is the probability of finding  $(j + 1)$  or more used probes in the extension set for the correct path. Figure 5.2 displays the probability of an incorrect polling result for Mode 1 and Mode 2 failures for a range of target sequence lengths. The probability of selecting an incorrect path is never greater than 4%. However, the probability of an incorrect result masks the fact that polling errors are far more likely when failures are encountered early in the sequence. Also, note that for long sequences, the expected number of failure points in the sequence also rises: although there may only be about a 4% chance of choosing



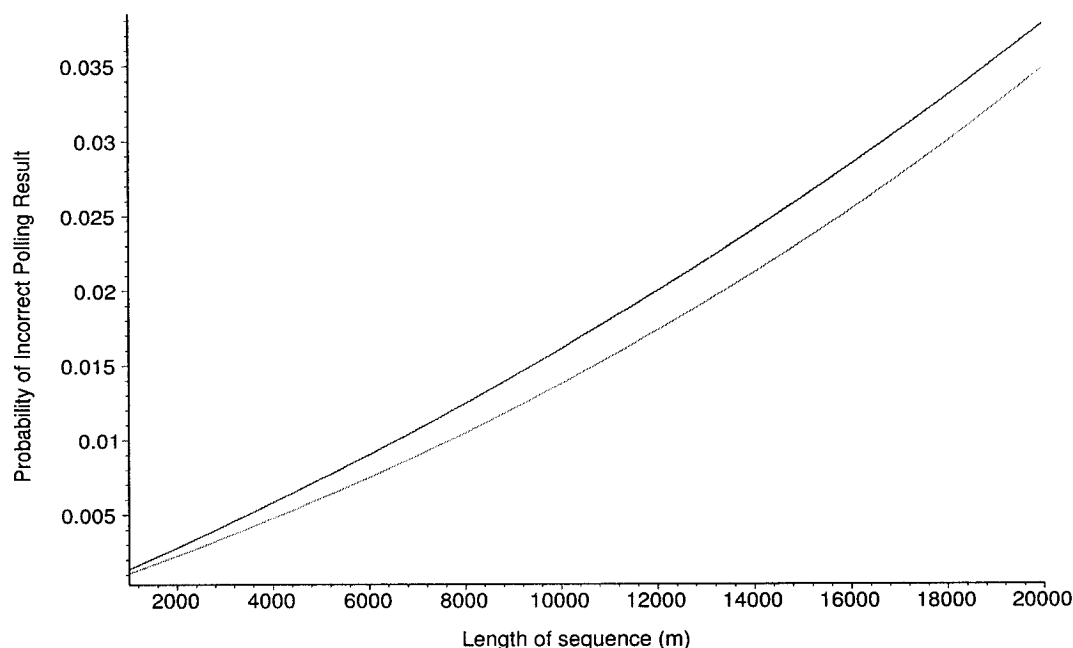


Figure 5.2: The probability that the polling algorithm selects the incorrect path after a Mode 1 (upper curve) or Mode 2 (lower curve) failure. The probability is weighted by the likelihood of each type of failure at all positions  $1 \dots \mu \dots m$ .

the incorrect path at a particular failure point in a 18,000-character target sequence, there are likely to be several failure points within the sequence.

## 5.5 Using PCR Primers

The performance benefit derived from the polling algorithm is dependent on the *correct* resolution of failures. Averaged over all  $\mu$ , the probability of choosing the correct extension for both Mode 1 and Mode 2 failures is very high, but for  $\mu < m/4$ , the probability of an incorrect polling result is approximately 0.5. Figure 4.3 shows the probability that the polling algorithm will produce an incorrect result for  $0 < \mu \leq \frac{3}{5}m$ .

It is evident from this figure that when a failure occurs very early in the sequencing process, the polling algorithm cannot be expected to produce reliable results. In fact, when  $\mu = m/20$ , the polling algorithm will either fail to produce an answer (due to a tie in the polling) or produce an incorrect result about 2/3 of the time, whereas for all  $\mu > m/2$ , the likelihood of a polling failure is very close to 0. It is possible to take

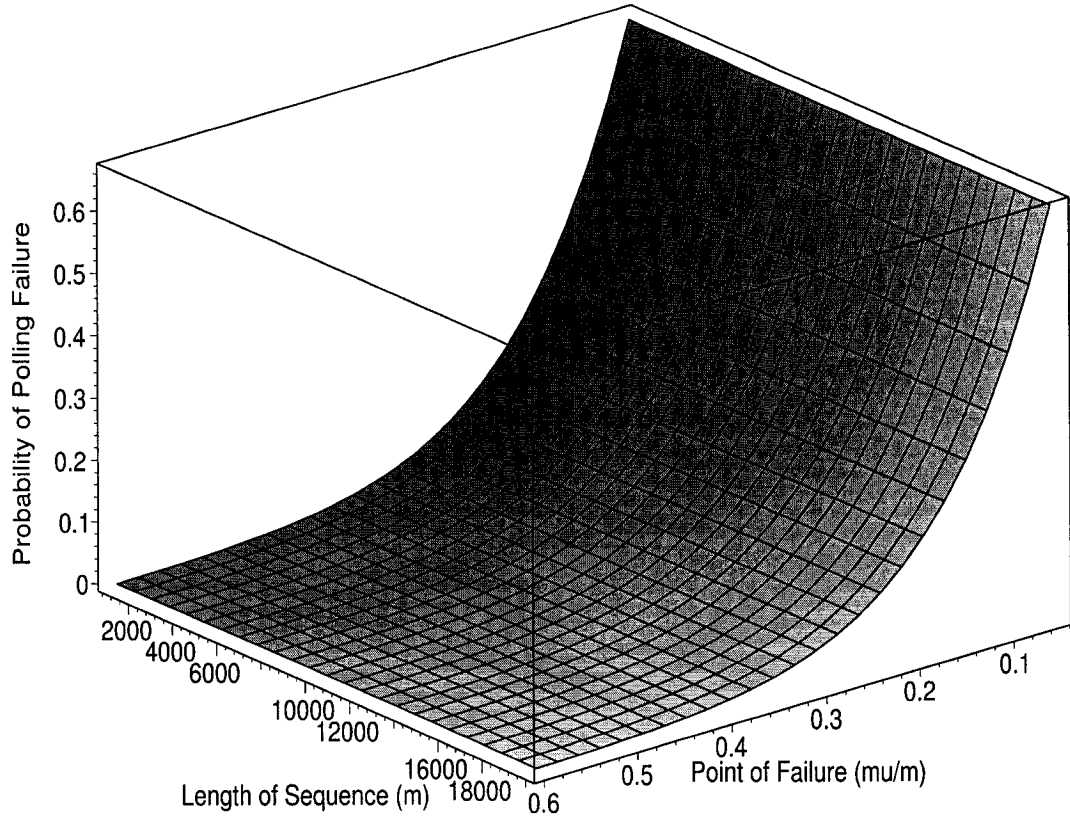


Figure 5.3: Probability of an incorrect polling result as a function of target sequence length ( $m$ ) and failure point ( $\mu/m$ ).

advantage of this phenomenon if we assume that both PCR primers are available.

Consider first the case of a Mode 1 failure which occurs at position  $\mu \approx \frac{m}{10}$  of a 12,000-character sequence. The probability that the polling algorithm will successfully resolve the failure is only about 51%. On the other hand, a Mode 1 failure which occurs at  $\mu \approx 9m/10$  of a 12,000-character sequence can be successfully resolved with over 99.9% confidence. By using both PCR primers, and performing two separate sequencing attempts—one beginning at each end of the sequence—it is possible to deal with the first event ( $\mu \approx \frac{m}{10}$ ) as an event of the second type ( $\mu \approx \frac{9m}{10}$ ). The same Mode 1 failure will be encountered during both sequencing attempts. The polling algorithm will be called upon twice to attempt to resolve the failure, and the result with higher probability of success can be chosen.

By choosing to deal with a Mode 1 error in a way that maximizes success, the

probability that each of the fooling probes confirming the spurious path is changed to  $\frac{\max(\mu, 1-\mu)}{m}$ . Also, the likelihood of finding a used probe confirming the correct path is  $\delta = 1 - e^{-\frac{\max(\mu, 1-\mu)}{4\kappa}}$ . Effectively,  $\mu$  is never less than  $m/2$ . Figure 5.4 shows the increase in confidence derived from selecting the more favorable direction from which to use the polling algorithm.

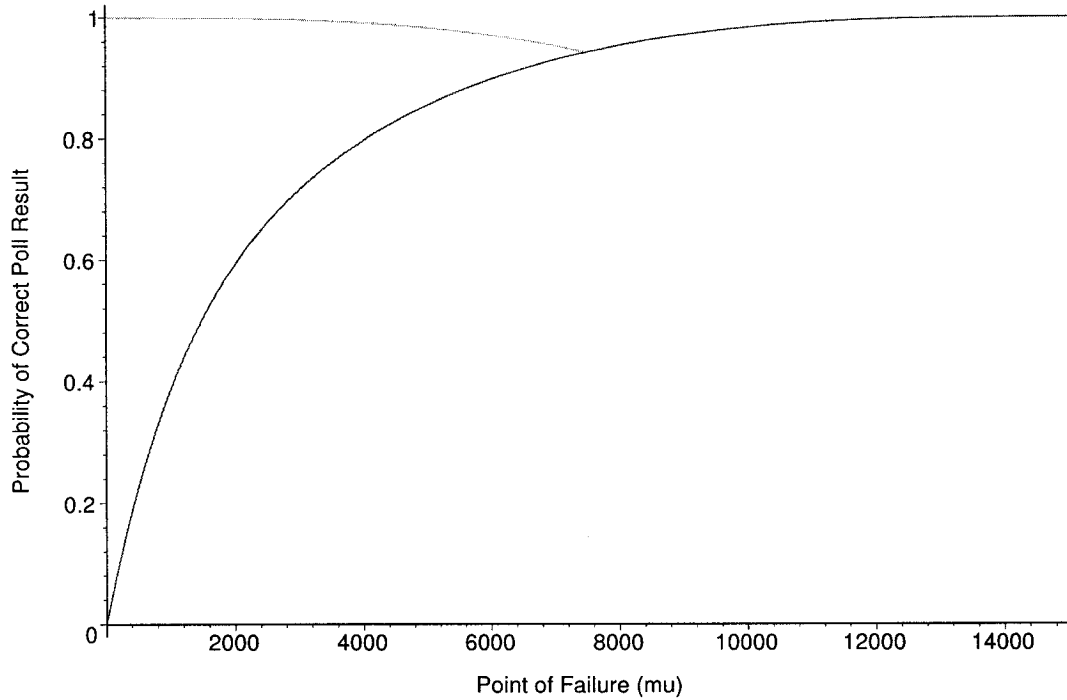


Figure 5.4: Probability of polling failure given a Mode 1 failure at the point  $\mu$  in a  $m = 15,000$ -character sequence. The lower curve displays the single-direction case; the upper curve the dual-direction (PCR seed) case. Note that the upper and lower curves are identical for  $\mu \geq m/2$ .

The analysis for Mode 2 failures is somewhat more complex, since the set of fooling probes which are required for a Mode 2 failure is dependent on the two homologous strings, the probing pattern, and even the relative position of the two homologous strings. When sequencing from right-to-left instead of left-to-right, the probing pattern is effectively reversed. Thus, a pair of homologous strings at positions  $i$  and  $j$  which lead to a Mode 2 failure at position  $i$  during forward sequencing may not cause a Mode 2 failure at  $i$  during reverse sequencing. Nor does it appear to be likely that

a Mode 2 failure *will* be observed at position  $j$  during forward *or* reverse sequencing. Furthermore, there may be a Mode 2 failure at some position  $l \neq i, j$  which occurs during reverse sequencing, entirely unrelated to the Mode 2 failure observed in forward sequencing.

Although it is difficult to prove, we may reasonably assume that a the forward and reverse sequencing attempts made using PCR primers are independent attempts to reconstruct the string: the occurrence of a Mode 2 failure in one direction does not affect the probability of finding such a failure in the other direction. Thus, by performing two sequencing attempts, using both PCR primers, we may be able to eliminate a Mode 2 failure completely. This assumption appears to be justified by the experimental results. If a Mode 2 failure is encountered only during one of the two sequencing attempts, then the sequencing attempt which does not contain a failure may be used; we only need to resort to the polling algorithm when there is a Mode 2 failure in both the forward and reverse sequencing attempts. If the probability of encountering a Mode 2 failure while reconstructing an  $m$ -character DNA fragment is  $P_2(m)$ , then the likelihood of encountering a Mode 2 failure in *both* the forward and reverse sequencing attempts is  $P_2(m)^2$ .

On the other hand, even if a Mode 2 failure *does* occur in both the forward and reverse sequencing attempts, we can select between the two events based on the likelihood of a correct polling result. In this case, the improvement in polling success is somewhat lower than for Mode 1 failures, but the bi-directional approach still improves the confidence of the polling method significantly. In this case, each failure occurs at point chosen uniformly on  $[1, m]$ . Let  $\pi(\mu)$  by probability that the larger of the two values occurs at the point  $\mu$ . Then  $\pi(\mu) = 2 \cdot \frac{\mu}{m^2}$ .

The following table shows the predicted vs. expected proportion of Mode 1 and Mode 2 failures which are resolved incorrectly (producing a false positive sequencing results) by the polling algorithm for  $m = 10000, 13000, 16000$ : this is the error rate of the polling algorithm. (For lengths  $m < 10000$ , failures of either type are too rare to be relevant.)

Length $m$	<i>Mode 1</i>		<i>Mode 2</i>	
	Predicted	Observed	Predicted	Observed
10000	1.6%	1.5%	0.8%	1.0%
13000	2.2%	2.1%	1.1%	1.2%
16000	2.8%	2.8%	1.5%	1.5%

As we can see, the number of observed successful polling attempts matches very well the predicted values, and we can generally trust the polling algorithm to produce the correct result.

## 5.6 Effectiveness of Polling on Natural DNA

The performance of the branching EXTEND algorithm shows moderate to severe degradation on natural DNA sequences, due to the non-uniform distribution of bases, and the presence of long repeats within target sequences. Figures 5.5 and 5.6 show the improvement in achievable target sequence length which is gained by applying the polling algorithm to natural DNA. The maximum length which can be sequenced with an expected 90% success rate ( $\epsilon = 0.9$ ) is also shown in the following table, for 4 different natural DNA sources:

	Gapped Algorithm	w/ Polling	Verified Polling
Random DNA	13800	17700	17300
<i>a. thaliana</i> chr. III CDS	1700	2600	2200
human chr. 3 CDS	1400	1500	1400
<i>s. cerevisiae</i> chr. IV CDS	2300	3300	2900
<i>e. coli</i>	5400	7600	6300
<i>h. influenzae</i>	2800	3800	3500
<i>salmonella</i>	6000	7700	7400
<i>s. solfataricus</i>	3000	3200	3200

The results in the second column of the table above present include *false positives*—sequences which appear to be correct, but contain at least one incorrect nucleotide at a position  $i$ , where  $H < i < m - H$ . The third columns displays only verified *correct* sequences: these numbers are derived by subtracting the number of false positive

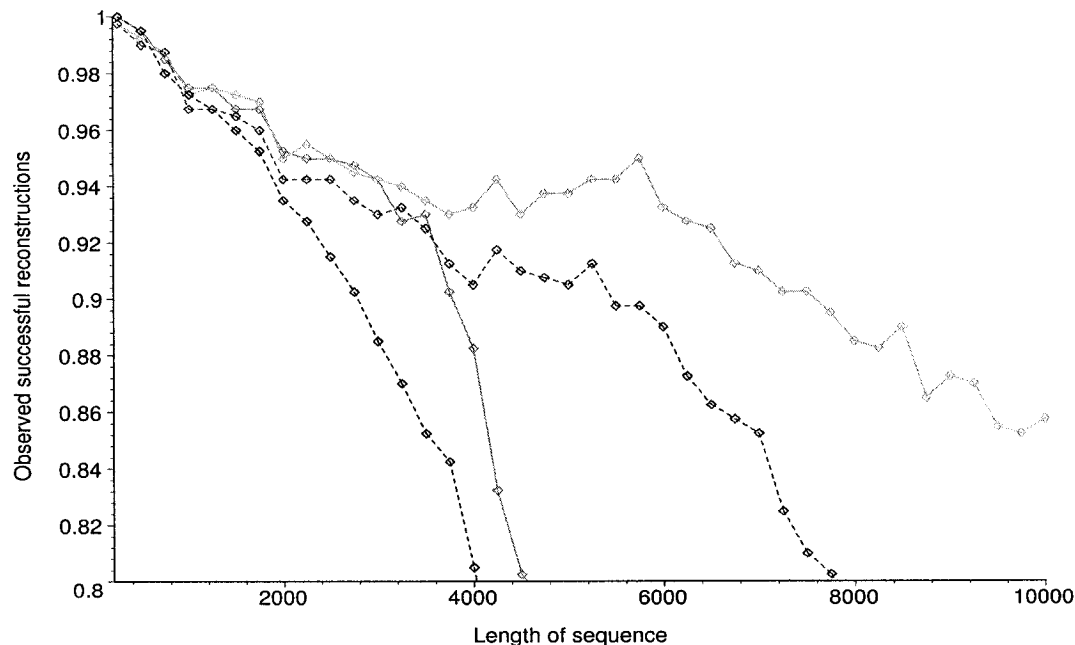


Figure 5.5: Observed performance of polling provision on *h. influenzae* (the two left-most lines) and *e. coli* (the two right-most lines). The solid grey lines show performance with the polling provision; the dotted black lines without.

reconstructions from the total number of presumed successes. While false positive results are simple to detect in simulation, they are impossible to discover in the real world. However, since a precise model of natural DNA is not available, it can be helpful to observe experimentally the overall number of polling errors. We note that the percentage of false positive sequences, where  $\epsilon = 0.9$  is less than about 4%. Where there appears to be a large difference between the maximum fragment length in the *Polling* and *Verified Polling* conditions, it is because for some natural DNA sources, the performance of the sequencing algorithm remains almost constant for a large range of  $m$  (as can be seen for *e. coli* in Figure 5.5). In the range  $m = 5000 \dots 7500$ , the *Polling* condition shows a success rate of approximately 90-91%, and so a very small proportion of false positive results can have a large impact on the data.

Overall, the polling provision is a fairly simple and effective way of extracting a performance improvement of about 20% on random DNA at no additional cost in terms of microarray size, and very little computational work. Furthermore, the method appears to work extremely well on some natural DNA sequences. With the

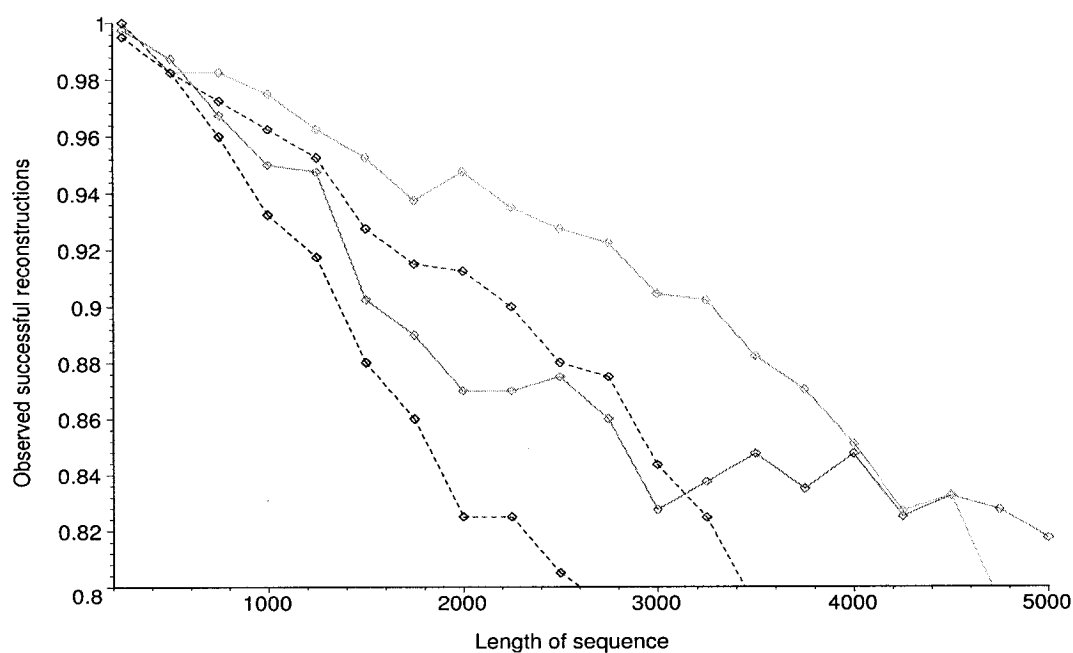


Figure 5.6: Observed performance of polling provision on human chromosome 3 coding sequences (the two left-most lines) and *s. cerevisiae* (the two right-most lines). The solid grey lines show performance with the polling provision; the dotted black lines without.

polling provision, SBH can reconstruct human coding sequences about 50% longer than without, and the performance on *salmonella* approaches 8Kb, even after subtracting false positive results.

## Chapter 6

# Identifying and Escaping from Repeating Segments

To the great benefit of biological diversity and the disadvantage of computational biologists trying to perfect SBH, the DNA of living organisms is not uniformly random. The non-random nature of DNA is evident in many ways. Individual nucleotides do not always occur with equal probability, thus altering the occurrence probabilities of the four bases. Transposons produce long (several hundred bases or more) exactly repeated subsequences, which automatically cause the gapped-SBH algorithm to fail. Finally, shorter transcription errors can lead to tandem repeats or microsatellites: periodic sequences composed of a very short segment which repeats over and over in succession. This chapter is concerned with the effect that these repeating segments have on the gapped-SBH sequencing algorithm.

### 6.1 Recognizing Repeating Segments

While tandem repeats may theoretically occur in randomly-generated DNA, in practice they are encountered almost exclusively in natural DNA sequences. They have the effect of triggering explosive branching in the path tree, resulting in the relatively common occurrence of failures due to violating the breadth-bound  $B^1$ . When they occur, the size of the path tree originating at a single ambiguous character can exceed

---

<sup>1</sup>Compared with the incidence of such breadth-bound failures while sequencing random DNA.



$B$  at a depth of only 50-100. A quick glance at the complete set of paths generated in such failure cases reveals the unique nature of these *repeat failures*.

Based on experimental observation, there appear to be three basic types of these failures, which we will refer to as *Class 1* . . . 3. The following example shows a typical occurrence of a Class 1 failure. In each path, the leftmost character corresponds to the branching position, so the initial ambiguous character is a choice between the two options A and G):

**Class 1**

```

... ATC ATC ATC ATC ATC ATC ATC ATC ATC ATC
... ATC ATC ATC ATC ATC ATC ATC ATC ATC ATC GTG
... ATC ATC ATC ATC ATC ATC ATC ATC ATC GTG AAC
... ATC ATC ATC ATC ATC ATC ATC ATC GTG AAC GGG
... ATC ATC ATC ATC ATC ATC ATC GTG AAC GGG ACC
      ⋮
... ATC ATC GTG AAC GGG ACC CGA TGT CTA GCT
... ATC GTG AAC GGG ACC CGA TGT CTA GCT CCC
...GTG AAC GGG ACC CGA TGT CTA GCT CCC CAA

```

In this example, the repeating segment in each path is highlighted in grey. In each path, there are a variable number of copies of the triplet ATC followed by a path GTGA . . . , which is identical (to the extent that it can be observed) in all paths. Not all failures that contain a repeating segment are as simple as this one.

In other cases, there can be two (or more) sequences which continue from the end of the repeating segment, but the failure is of the same basic type. For instance, the following example shows an example where there are two different paths extending the sequence after a GA repeating segment. These types of repeat failure will be called *Class 2*.

**Class 2**

---

```

... GA GA GA GA GA GA GA G
... GA GA GA GA GA GA GA T
... GA GA GA GA GA GA GA C C
... GA GA GA GA GA GA GA TA A
... GA GA GA GA GA GA G C CA T
... GA GA GA GA GA TA AG C
... GA GA GA GA G C CA TG A
... GA GA GA GA TA AG CC T
      ⋮
... GA CC AT GA CG GC GA T
... G T AA GC CT TT TA TG G
... CC AT GA CG GC GA TT C

```

Finally, there is a third class of failures involving two or more *different* short repeated segments. If the two repeated segments are GA and ACC, then the corresponding set of paths might look like:

**Class 3**

---

```

      ⋮
... A C C A C C A C C A C C A C C A C
... G A G A G A G A G A G A G A G A G
... A C C A C C A C C A C C A C C A G
... G A G A G A G A G A G A G A G T T
... G A G A G A G A G A G A G T T A C
... A C C A C C A C C A C C A G G T C
... G A G A G A G A G A G T T A C G G

```

These three classes of repeat-segment failure appear to account for up to 20% of the sequencing failures for natural DNA. Since the incidence of such events is dependent on natural biological processes, it seems to be impossible to predict exactly how often they occur. However, it is possible in many cases for the EXTEND algorithm to recognise a repeating segment when it is encountered, excise it, and continue sequencing.

## 6.2 Causes of Repeat Failures

In natural DNA, regions consisting of consecutive occurrences of a very short subsequence are called *tandem repeats*, or *microsatellites*. Such repeating segments will be referred to here more precisely as *periodic segments*. When these periodic segments reach a sufficient length, they confound the operation of the sequencing algorithm, and produce the repeat failures described above. Notationally, we define periodic segments as follows:

**Definition 6.1.** Let  $a$  denote a string of  $k \geq 1$  characters, and  $a'$  denote a  $j$ -character prefix of  $a$ , for  $j \geq 0$ . Using the notation  $a^n$  to represent  $n$  consecutive copies of the sequence  $a$ ,  $a^n a'$  denotes a periodic segment with total length  $\phi = k \cdot n + j$ , and a period of length  $k$ .

Within a periodic segment, the short sequence  $a$  is called the *repeated segment*. Periodic segments which are shorter than the probe length  $\lambda$  do not produce sequencing failures. In other words, when EXTEND encounters a periodic segment  $p = a^n a'$  during sequence reconstruction, a failure occurs only when sequencing cannot proceed past the segment. If a target sequence  $S$  contains a periodic segment the range  $S_{(i,j)}$ , then there is of course a probe in the spectrum corresponding to the  $(j+1)^{\text{th}}$  character of  $S$ . If there are no spurious-extension probes at position  $j+1$ , then obviously there can be no difficulty in continuing the sequencing process.

However, if there is a spurious-extension probe for  $j+1$  which is contained entirely within  $S_{(i,j)}$  (i. e. the rightmost character of the fooling probe falls within the repeating segment), then a branch occurs in the sequencing process. There are two feasible-extension probes at the branching position: one feasible-extension probe produces the correct post-repeat extension of the sequence, while the other (spurious) probe allows the repeating segment  $p$  to continue with another repetition of  $a$ . Such a fooling probe must be contained entirely within the periodic segment, and will be found at some position  $j+1 - (i \cdot k)$ ,  $i > 0$  of the sequence. Consider the following example:

**Example 6.1.** Using a (3,2)-reverse probing pattern (N . . N . . NNN), a repeating segment  $p = a^4 a'$  with period  $a = \text{GGC}$  and  $a' = \text{G}$  is encountered.

	...	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
actual	...	A	[(G	G	C)	G	G	C	G	G	C	G	G	C	G]	A	T	T	T	...
escape							C	.	.	C	.	.	C	G	A					
spurious					C	.	.	C	.	.	C	G	G							

The span of the periodic segment in this example is  $S_{2,13}$ . The fooling probe for position 14 is located at position  $14 - k = 11$ , and  $|a| = k = 3$ . ■

There must be a spurious-extension probe at all positions  $j + 1 \dots j + k$ ,  $|a| = k$ . If *any* of these fooling probes is missing, then the spurious path begun at  $j + 1$  will be eliminated at depth  $k$  at the  $k^{\text{th}}$  level in the path tree. In order to produce the explosive branching characteristic of repeating segment failures, there must be a complete set of  $k$  fooling probes that fall entirely within the span of the periodic segment  $S_{(i,j)}$ .

**Example 6.2.** Consider the probing (3,3)-direct probing pattern  $NNN \dots N \dots N \dots N$ , and a target sequence containing the repeating segment  $a^3 a'$  with  $a = \text{GGCA}$  and  $a' = \text{GG}$ . The complete periodic segment is shown in square brackets [], and the period within parentheses (), on the top line.

...	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
...	A	[(G	G	C	A)	G	G	C	A	G	G	C	A	G	G]	T	T	T	...
1		G	G	C	.	.	G	.	.	G	.	.	A						
2			G	C	A	.	.	C	.	.	G	.	.	G					
3				C	A	G	.	.	A	.	.	C	.	.	G				
4					A	G	G	.	.	G	.	.	A	.	.	T			

The periodic segment spans  $4 \cdot 3 + 2 = 14$  symbols in the sequence. It does not contain a fooling probe for position 16 of the sequence, so this segment does not cause a sequencing failure. Although there are 3 probes which fall entirely within the periodic segment, probe 4 is the only feasible-extension probe for the 16<sup>th</sup> character in the sequence, so extension of the sequence beyond the repeating segment is unambiguous.

On the other hand, if the periodic segment is extended by only a single character, a failure does occur. Consider the following, with  $\text{GGCA}$  and  $a' = \text{GGC}$ .

...	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...	
...	A	[(G G C A) G G C A G G C A G G C]																T	T	A	C	...
1'		G	G	C	.	.	G	.	.	G	.	.	A									
2'			G	C	A	.	.	C	.	.	G	.	.	G								
3'				C	A	G	.	.	A	.	.	C	.	.	G							
4'					A	G	G	.	.	G	.	.	A	.	.	C						
5'						G	G	C	.	.	G	.	.	G	.	.	T					
6'							G	C	A	.	.	C	.	.	G	.	.	T				
7'								C	A	G	.	.	A	.	.	C	.	.	A			
8'									A	G	G	.	.	G	.	.	T	.	.	C		

With the longer periodic segment ( $\phi = 15$ ), when EXTEND attempts to produce the 17<sup>th</sup> character in the sequence, there are two feasible-extension probes: 5' (which provides the correct extension) and 1' (the spurious extension). Furthermore, as the two paths are extended further, probes 2', 3' and 4' ensure that the false path continues to be extended, while probes 6', 7' and 8' extend the correct path. At depth 5 in the tree, the spurious path produces a secondary branching, since both probes 1' and 5' match the corresponding extension query. The path tree at this point contains the following three paths, with the initial branching position shown in square brackets, the secondary branching in parentheses, and the putative sequence falling to the left of the branching position:

							1	2	3	4	5	
...	G	G	C	A	G	G	C	[T]	T	T	A	G
								[A]	G	G	C	(T)
												(A)

The newly-spawned spurious path will be extended to depth 9, at which point the cycle will continue. In fact, at depth  $4i + 1, i \geq 0$ , a new spurious path is spawned in the tree which extends the repeating segment by an additional  $k = 4$  characters. ■

In the above example, the first (shorter) periodic segment had length 14, which was one character shorter than the minimum length required to cause a repeat-failure with (3,3)-probes and a 4-character repeated segment (GGCA). In general, the minimum length required for a periodic segment to cause a failure, which will be denoted  $\rho$ , is given by the following theorem:

**Theorem 6.1.** *A periodic segment  $p$  has the form  $a^n a'$ , where  $a'$  is a prefix of  $a$  with length  $j \geq 0$ . If the total length of the periodic segment  $\phi = k \cdot n + j$  is at least  $\rho = \lambda + k - 1$ , then the sequencing algorithm fails when it encounters  $p$ .*

*Proof.* If the length of the periodic segment  $\phi \geq \lambda + k - 1$  characters, then there is a complete set of  $k$  fooling probes contained within the span of the periodic segment. These fooling probes are sufficient to allow the indefinite extension of the periodic segment, since for every possible alignment of the probing pattern with the period  $k$ , there exists a probe in the spectrum that is contained entirely within  $p$ .  $\square$

Note that in the event of a repeat-failure, the path tree *does* contain the correct extension beyond the repeated segment. However, the algorithm has no way of differentiating between the correct and spurious paths—without some modification. We will describe such a modification later in this chapter, but first the types of repeats encountered should be discussed a little further.

Example 6.2 is an example of what we will call a *well-formed* repeat, where a periodic segment containing at least  $\rho$  characters actually occurs in the target sequence. In the case where the repeating segment is only a single character too short to provoke the infinite branching characteristic of this type of failure, a single fooling probe at another point in the sequence can complete the cycle. For instance, in Example 6.2 above, when the repeating segment had length  $14 = \rho - 1$ , if the probe  $4' = \text{AGG} \dots \text{G} \dots \text{A} \dots \text{C}$  had been present in the spectrum as a fooling probe, a repeat failure would have occurred. A single fooling probe occurs with probability  $\alpha$  in a random sequence. In natural DNA, a particular fooling probe may be more or less likely to be present in the spectrum, for various reasons.

Any deviation from a uniform random model of DNA makes some DNA sequences more likely than others. For instance, the DNA of *p. falciparum* is composed predominantly of adenine (A) and thymine (T); these two nucleotides account for well over 80% of this organism's genome. Thus, a fooling probe such as  $\text{AAT} \dots \text{A} \dots \text{T} \dots \text{T}$  would be significantly more likely to be found in the spectrum of a target sequence drawn from *p. falciparum* than, for instance,  $\text{GGA} \dots \text{C} \dots \text{C} \dots \text{T}$ . Moreover, even if each of the four bases accounts for exactly one fourth of an organism's DNA, there are other factors which affect the frequency with which individual subsequences occur. Appendix C has a more detailed discussion of these effects.

Repeat failures that are caused by periodic segments shorter than  $\rho$  characters (and one or more fooling probes) are called *fooling-repeats*. They produce a sequencing failure which cannot be differentiated from that produced by a proper periodic repeat in the target sequence, even when such a sequence does not exist. It has not yet been determined what proportion of the observed repeat failures are due to well-formed repeats, and what proportion are due to fooling-repeats completed by fooling probes.

### 6.3 Variations on Repeat Failure

Class 1 failures are the most well-behaved and well-understood type of repeat failure. In such cases, there is only a single repeated segment  $a$ , and a single path out of the sequence after the periodic sequence  $a^n a'$ . After the initial branch, an additional spurious path is spawned every  $|a| = k$  characters. All of the spurious paths are identical, modulo the number of times the segment  $a$  occurs in each. Class 2 and 3 repeat failures are more complex, with more than one potential path issuing from the periodic segment, or more than one repeating segment. Before discussing the effects of these variations in detail, we need to define two new concepts, *continuations* and *exit points*.

**Definition 6.2.** A continuation is a sequence path which leads out of the periodic segment.

**Example 6.3.** In the following three paths, there is a single continuation past the periodic segment with period ‘GA’. The continuation consists of the sequence beginning with CCAT.

```
... GA GA GA GA GA G CCAT
... GA GA GA GA G C C A TGA
... GA GA GA G C C A T G AAC
```

■

**Definition 6.3.** A periodic repeat has the form  $a^n a'$ . The segment  $a$  consists of  $k$  characters  $a_1, a_2, \dots, a_k$ , and the segment  $a'$  has length  $|a'| < k$ . The exit point from the periodic repeat is said to occur at a particular character  $a_i, i \geq 0, i = |a'|$ .

**Example 6.4.** Using the same three paths as Example 6.3, the exit point to the sequence occurs at  $a_1$ . The repeated segment is ‘GA’, and  $a' = a_1 = G$ , which has length 1. In all three paths, the continuation ‘CCAT...’ begins after character  $a_1$  (G).

```
... GA GA GA GA GA G CCAT
... GA GA GA GA G C CA TGA
... GA GA GA G C CA TG AAC
```

■

**Example 6.5.** The following set of six paths contain 2 different continuations, each occurring at a different exit point. The repeated segment is ‘GAT’, the two continuations begin with ‘CCT’ and ‘ATAC’.

```
... GAT GAT GAT GAT CCT
... GAT GAT GAT GA ATAC
... GAT GAT GAT CCT AAC
... GAT GAT GA A TAC AAA
... GAT GAT CCT AAC GCC
... GAT GA A TAC AAA CCA
```

Each of the two continuations has a different exit point from the periodic sequence: ‘CCT’ exits after  $a' = \text{''}$ , or at  $a_0$ , and ‘ATAC’ exits after  $a' = GA$ , or  $a_2$ . ■

Following any periodic segment, there can be only one *correct* continuation, and thus only one correct exit point. There are two reasons why spurious continuations might appear. First, an appropriate collection of fooling probes can create the appearance of other continuations to the sequence after the periodic repeat. Second, the periodic segment may not be unique in the target sequence. For instance, a single periodic segment  $a^n a'$  may be followed by the subsequence  $b$ . Another periodic segment  $a^n a''$  with an identical period  $a$  may occur later in the target sequence, where it is followed by the subsequence  $c$ . The whole local segment has the form  $\dots a^n a' b \dots a^n a'' c \dots$ , where  $a' b$  and  $a'' c$  will create two alternate continuations to the first occurrence of the periodic segment. If  $a' = a''$ , then both continuations will occur at the same exit point, but if  $a'$  and  $a''$  have different lengths, then each continuation will occur at a different exit point from the periodic segment.

Finally, there can be a combination of multiple continuations and multiple exit points to a sequence. Here is an example of a periodic segment with three exit points:



**Example 6.6.** The repeating segment  $a$  is AGA, two exit points occur at  $a_0$  (continuations GAGT and TAGT) and a third continuation occurs at exit point  $a_1$  (ACGG).

```

... AGA AGA AGA AGA AGA AGA A G...
... AGA AGA AGA AGA AGA AGA A A...
... AGA AGA AGA AGA AGA AGA GA...
... AGA AGA AGA AGA AGA AGA TA...
... AGA AGA A AC GGG CGT GGG AT...
... AGA AGA GAG TTT TAA TCG GA...
... AGA AGA TAG TTT TAA TCG GA...
... AGA A AC GGG CGT GGG ATT AA...
... AGA GAG TTT TAA TCG GAT TG...
... AGA TAG TTT TAA TCG GAT TG...
... A AA CGG GCG TGG GAT TAG GG...
... G AG TTT TAA TCG GAT TGG GA...
... T AG TTT TAA TCG GAT TGG GA...

```

Note that the two exit points occurring at  $a_0$  are identical to each other after the first character:

```

... 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ...
... (A G A) [G] A G T T T T A A T C G G ...
... (A G A) [C] A G T T T T A A T C G G ...

```

Such matching continuations are indicative of a Mode 1 failure that immediately follows the periodic segment. On the other hand, the third exit point is an entirely different path, similar to a Mode 2 failure. Similar combinations of multiple exit points are not uncommon in practice. ■

The interaction between periodic segments and the standard failure modes creates some difficulty in designing an algorithm to recover from repeating segments. In some cases, it should be possible to use the branching EXTEND algorithm to choose between alternate continuations after handling the periodic segment. It may also be possible to combine repeat-detection with the *polling* provision to handle situations which cannot be resolved by EXTEND.

Finally, there is a phenomenon related to periodic segments, which we will call *pseudo-repeats*. These segments appear to be abetted by the incidence of low entropy segments of DNA that do not take the form of a strict periodic repeat, but have the

form  $(a^n b^m)^k$ , where  $n$  and  $m$  are not constant values, but can vary as  $k$  increases. Many of the observed examples of this event appear to have  $m = 1$  and  $|b| = 1$ . It may be possible to recover from these pseudo-repeats as well, but more work on this is required.

For now, we want to develop a strategy for detecting and recovering from a well-formed periodic segment in the target sequence.

## 6.4 The REPEATRECOVERY Algorithm

When the branching EXTEND algorithm encounters a periodic repeat, each of the possible exit points from the repeat will spawn a path every  $|a| = k$  characters while EXTEND attempts to resolve the initial ambiguity. As the EXTEND algorithm extends the path tree deeper, the path tree grows wider. In fact, the width of the path tree at depth  $i$  is proportional to  $i \cdot k$ .

We are only interested in periodic repeats which are long enough to produce a sequencing failure. Therefore, we only need to look for periodic repeats after the EXTEND algorithm has failed. The algorithm that searches for and attempts to recover from periodic sequences is REPEATRECOVERY; the pseudo-code for this algorithm will be listed later in this section. First, we describe several sub-steps which are required by REPEATRECOVERY.

We expect that the  $\lambda - 1$  characters preceding the initial branching position consist of the pattern  $a^n a'$ . At the branching position, there should be two alternative extensions: one extension begins another cycle of the periodic segment  $a$ ; the other which corresponds to the first character of the continuation after the repeat. This is true in most cases. However, Class 3 repeat failures differ from Class 1 and 2 in an important characteristic.

If there is only a single periodic segment present in a repeat failure, then the sequencing algorithm will not encounter a branch until the first exit point to the periodic segment. In the case of Class 3 failures, since the repeated segments themselves are not identical, the sequences must diverge at the first occurrence of a mismatch between the repeated segments.

For now, we handle Class 3 failures by ignoring them, and consider only repeat

failures in which there is a single repeating segment shared among all paths. This allows the construction of the following algorithm for detecting repeats. We look for a periodic sequence with period  $i = 1 \dots \lambda - 1$ , in the  $\lambda + i - 2$ -suffix of the putative sequence (including the branching position).

Note that periodic segments with short periods are preferred over longer periods, for the simple reason that a repeating segment that consists of  $(AT)^n$  could also be viewed as a repeating segment consisting of  $(ATAT)^n$ . The only difference between the two periods is that the critical length  $\rho$  of a periodic repeat depends on the length of the repeated segment ( $k$ ). In this case, an additional two symbols would be required to cause sequencing failure if the larger repeated segment were used.

```

REPEATFINDER( $s, \lambda$ )
1  for  $i \leftarrow 1$  to  $\lambda$ 
2      do if  $s_1 = s_{1+i}$ 
3          then  $a \leftarrow s_{1\dots i}$ 
4               $j \leftarrow 1$ 
5               $\text{repeat} \leftarrow \text{true}$ 
6              while  $j < \lambda + i$  AND  $\text{repeat} = \text{true}$ 
7                  do  $b \leftarrow s_{j\dots j+i}$ 
8                      if  $a \neq b$ 
9                          then  $\text{repeat} \leftarrow \text{false}$ 
10                      $j \leftarrow j + i$ 
11              if  $\text{repeat} = \text{true}$ 
12                  then return  $a$ 
13 return failure

```

The REPEATFINDER algorithm is run on every path in the tree. Once a periodic segment  $a$  has been found in one path, a similar algorithm, REPEATVERIFIER<sup>2</sup> is executed, to check that the periodic segment is present in all (or all but one) of the paths. If there is more than one path which lacks any occurrence of the periodic segment, then the REPEATRECOVERY algorithm fails.

---

<sup>2</sup>REPEATVERIFIER is similar enough to REPEATFINDER that the pseudocode is not included here.

Recall that for a probe pattern of length  $\lambda$ , and a periodic repeat consisting of the pattern  $a^n a'$ ,  $|a| = k$ , a repeating segment must have total length  $\rho = \lambda + k - 1$ . Any periodic repeat that is at least  $\rho$  characters in length will provoke infinite branching. Thus, it is impossible for the algorithm to determine the exact length of any periodic repeat. All repeating segments of sufficient length are identical in terms of the path tree they produce.

Furthermore, there is no way for the REPEATRECOVERY algorithm to distinguish between fooling-repeats and true periodic segments. If a periodic segment in the target sequence contains fewer than  $\rho$  symbols, a repeat failure occurs as long as fooling probes are found that compensate for the length shortfall.

Once a periodic segment has been verified in all paths, we can attempt to recover from the failure. We present a simple algorithm that removes the periodic segment from all paths so that we can reduce the tree to a set of only a few possible *continuations*. The REPEATREMOVER algorithm removes all copies of the string  $a$  from every path, along with the shortest  $a'$  string which was found in all paths. This has the effect of treating all continuations as though they share the same exit point. In explanation, consider the two sequences:

```
...AGAAGA A ACGGGCGTGGGAT...
...AGAAGA G AGTTTTAATCGGA...
```

The repeated segment is 'AGA'. In the first sequence,  $a' = A$ , and in the second sequence,  $a'$  is empty. Instead of viewing the continuation in the top path as 'ACGG...', occurring at  $a_2$ , we can treat it as though it contains the continuation 'A ACGG...', occurring at  $a_1$ . Then both continuations share the same exit point  $a_1$ , and we avoid introducing an offset between paths.

Once the periodic segment has been removed from all paths, we are still left with a (potentially) large set of paths. Each path *should* correspond to a prefix of the post-repeat sequence. The following table shows such a set of paths before and after the periodic segment has been removed. We will denote the set of paths containing the repeating segment as  $S_1$  and the set of paths with the repeating segment removed as  $S_0$ .

With Periodic Segment ( $S_1$ )	→ Without Periodic Segment ( $S_0$ )
ATC ATC ATC ATC ATC ATC GTG	→ GTG
ATC ATC ATC ATC ATC GTG AAC	→ GTGAAC
ATC ATC ATC ATC GTG AAC GGG	→ GTGAACGGG
ATC ATC ATC GTG AAC GGG ACC	→ GTGAACGGGACC
ATC ATC GTG AAC GGG ACC CGA	→ GTGAACGGGACCCGA
ATC GTG AAC GGG ACC CGA TGT	→ GTGAACGGGACCCGATGT
GTG AAC GGG ACC CGA TGT CTA	→ GTGAACGGGACCCGATGT CTA

Note that all of the paths in  $S_0$  appear to be prefixes—of varying length—of the same sequence. In fact, each path in  $S_0$  is a prefix of the longest path in the set. This indicates that there is only a single continuation to the sequence after the repeating segment. In such a case, we can select the longest path in  $S_0$  as the correct extension to the sequence. However, there may be mismatches between the paths in  $S_0$ . There are two reasons that this may occur:

1. There is more than one continuation to the periodic segment.
2. There is an ambiguous character (and consequent branch) during the sequencing of a continuation.

In either of these cases, we cannot simply choose the longest path in  $S_0$ , and must perform additional work to try to decide between the alternate continuations. However, we still want to eliminate those paths in  $S_0$  which are exact prefixes of other paths in  $S_0$ . This can be performed quite simply: for every path  $p$ , if there is another longer path  $q$  which contains  $p$  as a prefix,  $p$  may be removed from the set. Each path in the above example is a prefix of the longest path in the set, so after removing all of the prefixes, only a single path is left: GTGAACGGGACCCGATGTCTA. This will be true for all Class 1 repeat failures.

The pseudo-code for the REPEATREMOVER algorithm itself is:

REPEATREMOVER( $P, a, a'$ )

- 1  $j \leftarrow |a|$
- 2  $k \leftarrow |a'|$
- 3  $Q \leftarrow \text{NIL}$
- 4 **for each** path  $p$  in  $P$

```

5      do while  $p_{1...j} = a$ 
6          do  $p \leftarrow p_{j+1...|p|}$ 
7           $p \leftarrow p_{k+1...|p|}$ 
8           $Q \leftarrow Q \cup p$ 
9  for each path  $p$  in  $Q$ 
10     do if  $\exists q \in Q$  s.t.  $q_{1...|p|} = p$ 
11         then remove  $p$  from  $Q$ 
12 return  $Q$ 

```

REPEATREMOVER produces the set of all continuations to a periodic segment. At this point, the standard branching EXTEND algorithm with the polling extension may be used to select between them. In the most well-formed cases (Class 1), there will be only a single continuation in the collapsed set of paths, and extension of the sequence may immediately proceed.

## 6.5 Results and Observations

Some minor bookkeeping needs to be done to indicate the presence of a repeated segment in the final reconstructed sequence. We are just running simulations, so we would like to verify whether the reconstructed sequence matches the target sequence, or whether it contains errors. Since the exact number of repeats is impossible to determine, a repeated segment  $a$  in the reconstructed sequence must be allowed to match an arbitrary number of copies of  $a$  in the target sequence; a base-by-base verification of the sequence is otherwise impossible. This imprecision also means that the precise length of the reconstructed sequence is no longer known with certainty. The uncertainty in length introduced by this method will typically be fewer than approximately 10 bases out of  $2000 < m < 10000$  when it is used.

Initial results from the implementation of the REPEATRECOVERY algorithm indicate that it offers a performance increase somewhat less than the *polling* mechanism on natural sequences. However, although the polling and repeat-recovery techniques may be combined for further performance increases, the benefits offered by each technique are not independent: it appears that sequences that cannot be reconstructed

by the polling technique often cause problems for the repeat-recovery method as well.

Furthermore, it does not appear possible to choose between alternate continuations by performing the breadth-first expansion of path trees, nor does the polling provision offer much benefit in choosing between the post-periodic paths. Periodic segments with multiple continuations are equivalent to Mode 2 failures, and the probes in the extension set for each path often occur in a low-entropy region near the periodic segment, resulting in a high number of *used* probes for each alternative.

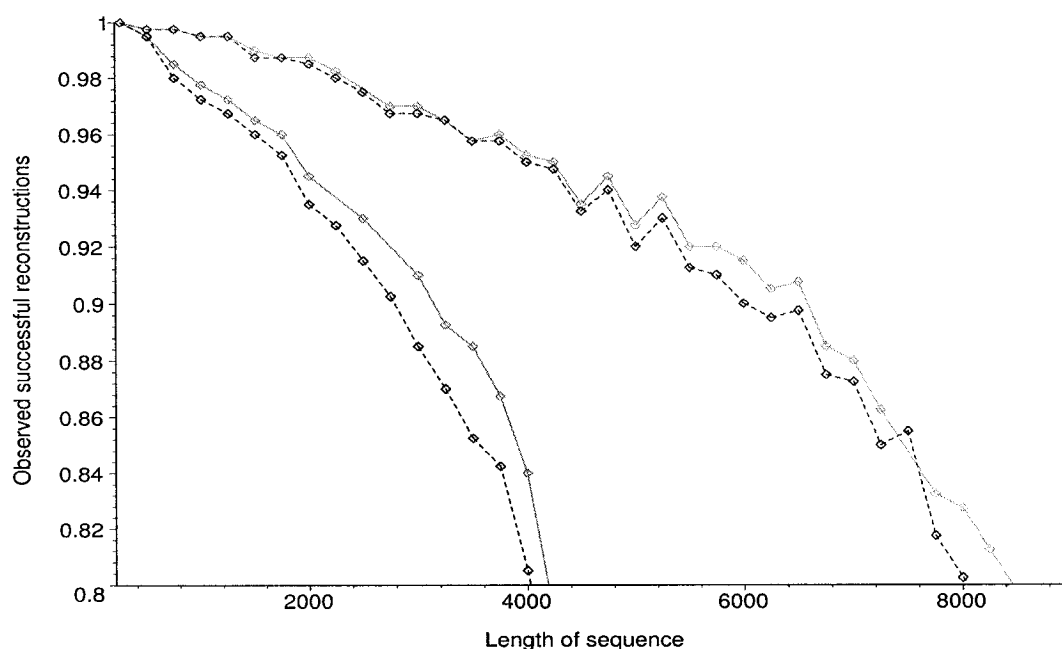


Figure 6.1: Observed performance of repeat recovery on *h. influenzae* (the two left-most lines) and *salmonella* (the two right-most lines). The solid grey lines show performance with the repeat recovery; the dotted black lines without.

Figures 6.1 and 6.2, along with the following table compare the results of the REPEATRECOVERY algorithm to the basic gapped-SBH algorithm, and the basic algorithm with the polling provision enabled. The numerical values in the table represents the longest fragment length for which at least 90% of the 400 non-overlapping fragments were completely and correctly reconstructed.

The data titled ‘Gapped Algorithm’ show the data for the basic sequencing algorithm, with neither the polling provision nor the REPEATRECOVERY algorithm enabled. The ‘Polling’ data illustrate the results of the algorithm with the polling

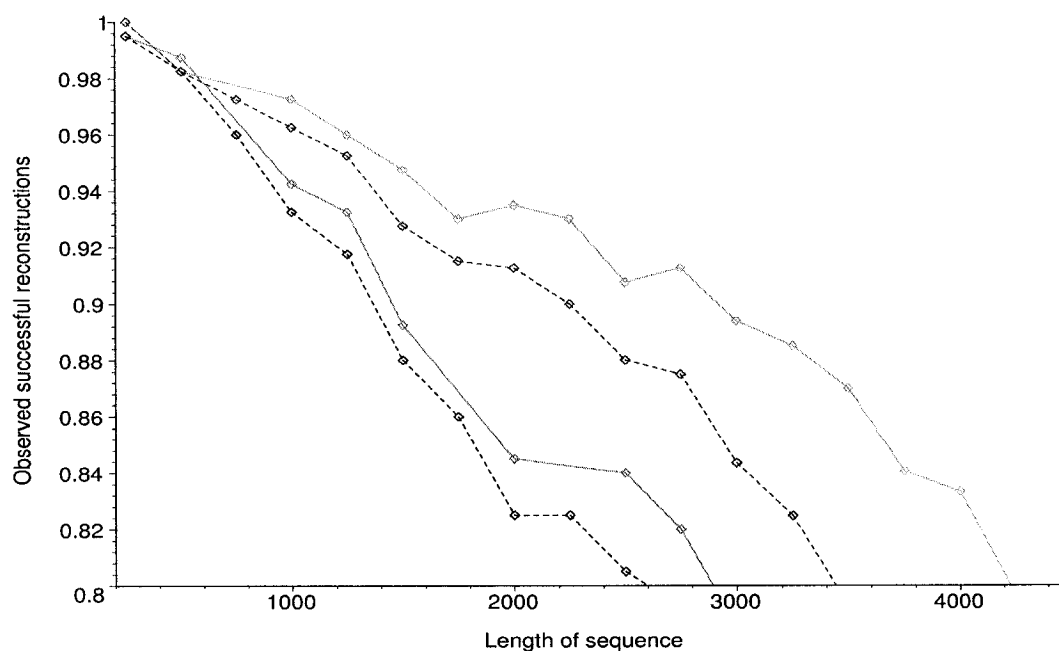


Figure 6.2: Observed performance of repeat recovery on human chromosome 3 coding sequences (the two left-most lines) and *s. cerevisiae* chromosome IV coding sequences (the two right-most lines). The solid grey lines show performance with the polling provision; the dotted black lines without.

provision enabled. Finally, the ‘Repeat’ data show the results of the sequencing algorithm with the REPEATRECOVERY algorithm enabled.

	Gapped Algorithm	w/ Polling	w/ Repeat
<i>a. thaliana</i> chr. III CDS	1700	2600	1900
human chr. 3 CDS	1400	1500	1400
<i>s. cerevisiae</i> chr. IV CDS	2300	3300	2900
<i>e. coli</i>	5400	7600	5800
<i>h. influenzae</i>	2800	3800	3100
<i>salmonella</i>	6000	7700	6600
<i>s. solfataricus</i>	3000	3200	3000

The above data illustrate that the REPEATRECOVERY algorithm does offer a small performance benefit to the basic method, but the current implementation leaves ample room for improvement.



## Chapter 7

# Pooling Information from Multiple Spectra

Until now, we have used only a single spectrum to perform sequencing. If a second spectrum is constructed from the same target sequence but using a different probing pattern, the information available therein might be expected to improve sequencing performance. In order to explore this hypothesis, we need to reformulate our performance metric slightly. Up to this point, we have simply calculated the length  $m$  of the longest target sequence which we can reconstruct with a 90% rate of success ( $\epsilon = 0.9$ ). However, every added spectrum requires the addition of another microarray, with corresponding increase in chip size and cost. If the maximum length of the target sequence does not increase by *at least* as much as the increase in cost, then the extra chip area is not justified by the performance gain. It would be more efficient to simply perform two separate experiments on shorter sequences.

In order to discuss the performance achieved by sequencing algorithms making use of multiple spectra, we will weight the results by the total chip size. More formally, we could adopt the metric of  $\frac{\# \text{ bases}}{\text{chip features}}$ . This level of precision is not needed here, since we will restrict our discussion to spectra of equal size. It is sufficient for our purposes to determine whether, for an  $n$ -fold increase in chip size and cost, we have achieved better than an  $n$ -fold increase in performance, measured in terms of the longest feasible target sequence. In the rest of the chapter, we will refer to the method of gapped-SBH using more than a single spectrum as *multi-spectrum* SBH.

Before continuing, we note that as  $m$  increases, so does  $\alpha$ . In fact, when  $m = 27000$ —about twice the feasible limit for the single-spectrum SBH algorithm—the value of  $\alpha$  for both (5,3)- and (4,4)-probing patterns is 0.338. When  $\alpha$  exceeds 0.25, the probabilistic extension of totally spurious paths using a single spectrum becomes very likely. To be useful, multi-spectrum SBH will have to overcome this limit.

## 7.1 Choice of Probing Patterns

If multiple spectra are to be used, the probing patterns used in the different spectra should have low correlation with one another. In the extreme case, imagine using the *same* probing pattern for each spectrum: the spectra obtained would be identical, and no additional information would be gained. Fortunately,  $(s, r)$ -probing patterns offer a natural and convenient source of probing patterns with low cross-correlation. If we calculate the cross-correlation of a direct  $(s, r)$ -probing pattern with its reverse, we find that there are never more than  $\max(s, r)$  natural bases aligned between the two patterns, at any offset. The correlation polynomial for a (5,3)-direct probe with a (5,3)-reverse probe is shown in Figure 7.1.

We will focus our discussion on a specific type of multi-spectrum SBH which makes use of two spectra: the direct and reverse versions of an  $(s, r)$ -probing pattern. For any given  $(s, r)$ -pattern, the construction of two microarrays of identical size for the direct and reverse patterns results in two spectra which will be called  $S_d$  and  $S_r$  respectively. The direct and reverse spectra of an  $(s, r)$ -pattern are together referred to as *tandem* spectra, and thus SBH using tandem spectra may be called *tandem sequencing*.

## 7.2 The Multi-spectrum SBH Algorithm

The simplest way of exploiting a second spectra is to make two attempts to reconstruct the target sequence, once with each spectrum. For probing patterns with low correlation, extensive computer simulation indicates that the two processes appear to be effectively independent, although a rigorous analysis has not been completed.

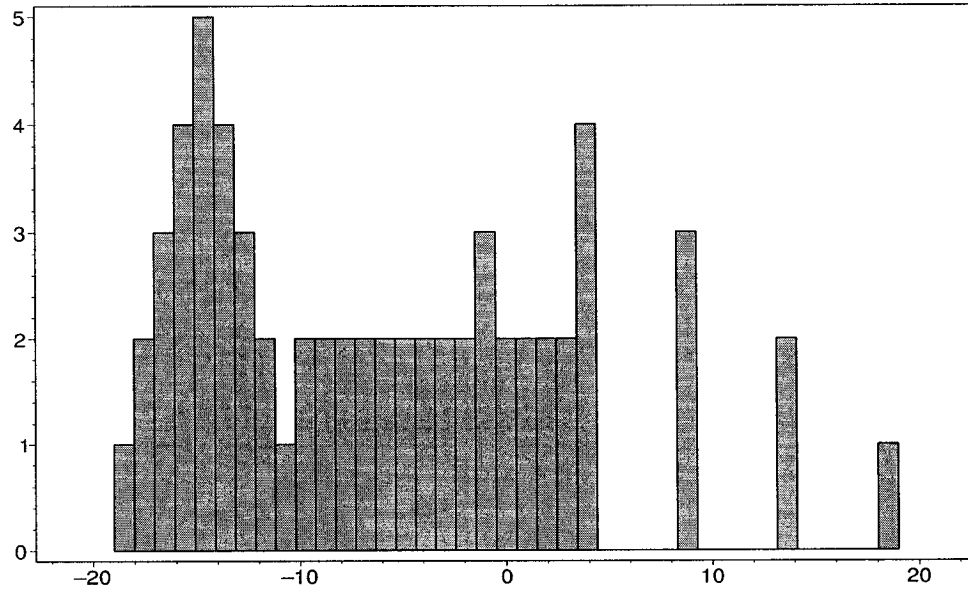


Figure 7.1: The cross-correlation polynomial for a (5,3)-direct probe (NNNNN....N....N....N) and its corresponding reverse (5,3)-probe (N....N....N....NNNNN).

The result is a trivial improvement in performance, with

$$p(\text{fail with } S_d) = p(\text{fail with } S_r) \Rightarrow p(\text{success}) \approx 1 - p(\text{fail with } S_d)^2$$

For fixed  $\epsilon = 0.9$ , the sequence length which can be achieved using two spectra in this way is about 16000, or an improvement about 16% over the single-spectrum case—an insignificant improvement compared with the doubled chip cost.

By contrast, there is another method of exploiting two spectra that yields more than double the performance of the single-spectrum algorithm. This is achieved by using the spectra in a local, or *probe-by-probe* fashion. This technique requires a new QUERY algorithm called MULTIQUERY that queries all spectra independently, and returns only results which are present in every spectrum.

MULTIQUERY is very similar to the QUERY algorithm described in Chapter 2. The only notational change made from the pseudo-code for the single-spectrum QUERY algorithm is that  $\mathcal{S}^*$  is used to indicate a *set* of spectra.  $\mathcal{S}^*$  is passed as a parameter to the multi-spectrum version of QUERY, which are each queried in turn. The pseudo-code for the multi-spectrum query algorithm is presented here:

```

MULTIQUERY( $\mathcal{S}^*, q$ )
1   $q' \leftarrow q_{(|q|-l, |q|-1)}$ 
2   $e_f \leftarrow \{ACGT\}$ 
3  for each  $S$  in  $\mathcal{S}^*$ 
4      do  $e_s \leftarrow \text{QUERY}(S, q)$ 
5           $e_f \leftarrow e_f \cap e_s$ 
6  return  $e_f$ 

```

The extension  $e_f$  is initialized to contain all possible extension characters. As each spectrum is queried in turn, extension characters not present in  $e_s$  are eliminated from  $e_f$ . A character will remain in  $e_f$  after all spectra are queried if and only if it was in the extension  $e_s$  of all spectra  $S$ . In this way, we ensure that a spurious extension can be produced only if it is confirmed by a fooling probe in every spectrum.

We would like to calculate how the use of multiple spectra and the MULTIQUERY algorithm affects the execution time of the SBH algorithm. (Recall from Chapter 3 that work can be measured in terms of spectrum CHECKS.) If there are no other effects on the running time size of path trees, then if MULTIQUERY is used in place of all calls to QUERY in the standard single-spectrum algorithms, then for  $n$  spectra in  $\mathcal{S}^*$ , we will perform  $n$  times as many calls to QUERY. In fact, the use of the second spectrum should reduce the size of path trees for a fixed length  $m$ , so in the worst case, multi-spectrum sequencing is only slower by a constant factor than single-spectrum sequencing. We would now like to examine how the additional spectra affect the probability of failure.

### 7.3 Probability of Failure with Tandem Spectra

The query-by-query verification of every possible extension in both the forward and reverse spectra has a significant effect on sequencing performance. The discussion here expands upon a similar analysis performed in [HPY02]. Here, as in that paper, Mode 1 and Mode 2 failures are treated separately.

### 7.3.1 Mode 1 Failures

As we discussed in Chapter 4 (see Theorem 4.1 for details), the probability of Mode 1 failure for the single-spectrum sequencing algorithm is

$$P_1^{(s)} = m \left(1 - e^{-\frac{3m}{4\kappa}}\right) \left(1 - e^{-\frac{m}{4\kappa}}\right)^{\kappa-1} \left[ \left(1 + \frac{4^{r+1}}{3m}\right)^r \left(1 + \frac{4^s}{3m}\right)^{(s-1)} \right] \quad (7.1)$$

The individual factors are briefly explained here. There are  $m$  ways of selecting a failure position. The spurious character at the position of failure may be chosen in 3 possible ways  $\left(1 - e^{-\frac{3m}{4\kappa}}\right)$  and then confirmed by  $\kappa - 1$  additional fooling probes  $\left(1 - e^{-\frac{m}{4\kappa}}\right)^{\kappa-1}$ . The remaining factor (contained within square brackets [ ]) is a correction factor to account for probe overlap.

In the tandem-spectrum case, the most significant change is in the third component of the equation:  $2\kappa - 1$  instead of  $\kappa - 1$  fooling probes are required. This occurs because although an ambiguous extension can take one of three possible values in one spectrum, once a value has been determined, it must be confirmed by a unique fooling probe in the companion spectrum. Without any loss of generality, we designate one spectrum as the *primary* spectrum  $S_1$ , and the other as the *secondary* spectrum.

Every query in the primary spectrum produces one or more spurious responses to a query with probability

$$1 - (1 - \alpha)^3 \approx 1 - e^{-\frac{3m}{4\kappa}}$$

which corresponds to the first factor in Equation 7.1.

If queried by itself, the secondary spectrum produces a spurious response with the same probability. However, there is only *one* possible probe in the secondary spectrum that will confirm a particular spurious character in the primary. Thus, the probability that MULTIQUERY produces a spurious character in response to a query is

$$\alpha(1 - (1 - \alpha)^3) \approx \left(1 - e^{-\frac{3m}{4\kappa}}\right) \left(1 - e^{-\frac{m}{4\kappa}}\right)$$

Once a branch occurs, it must be confirmed by an additional  $\kappa - 1$  fooling probes in each spectra. Thus,  $1 + 2(\kappa - 1) = 2\kappa - 1$  fooling probes are required to cause a Mode 2 failure. Finally, the correction factor for probe overlap (contained within the square brackets [ ] in Equation 7.1) must be raised to the power of two, since it affects

each spectrum independently. Thus, the probability of a Mode 1 failure using tandem spectra is

$$P_1^{(t)} = m \left(1 - e^{-\frac{3m}{4^\kappa}}\right) \left(1 - e^{-\frac{m}{4^\kappa}}\right)^{2\kappa-1} \left[ \left(1 + \frac{4^{r+1}}{3m}\right)^r \left(1 + \frac{4^s}{3m}\right)^{(s-1)} \right]^2 \quad (7.2)$$

We now move on to a discussion of Mode 2 failures.

### 7.3.2 Mode 2 Failures

In Chapter 4, we developed an approximation for the probability of Mode 2 failures, which are caused by so-called *self-sustaining* segments. At each of the  $m$  positions in the target sequence, there are  $m-1$  possible locations for the self-sustaining segment, and so there are  $\binom{m}{2} \approx m^2$  possible pairs of strings in the target sequence which can lead to failure. The probability of Mode 2 failures was calculated in terms of the position, or *offset* of a self-sustaining segment. For each position  $0 \leq J \leq \lambda$ , we calculated a coefficient  $\pi_J$ , such that the total probability of failure for the single-spectrum method is

$$\begin{aligned} P_2^{(s)} &= m^2 \sum_{J=0}^{\lambda} \pi_J \\ &= \frac{3m^2}{4^{\lambda-2}} \left( \frac{1}{32} + \alpha \frac{1 - (4\alpha^\theta)^\lambda}{1 - 4\alpha^\theta} \right) \end{aligned} \quad (7.3)$$

where the parameter  $\theta < 1$  is a scaling factor, accounting for the fact that fewer than  $J$  fooling probes are required at offset  $J$ . For a more detailed explanation of Equation 7.3, refer back to Section 4.3.

In order to calculate the expected performance enhancement derived by using tandem spectra, we need to examine how the  $\pi_J$  coefficients are changed. Recall that the individual coefficients  $\pi_J$  were

$$\pi_0 = \frac{3}{2 \cdot 4^\lambda} \quad (7.4)$$

$$\pi_{1 \dots \lambda} = 3 \frac{\alpha^{\theta J}}{4^\nu} \quad (7.5)$$

There are two components to the  $\pi_J$  multipliers:

1. The number of constrained characters in the sequence, each of which contributes a factor of  $1/4$ .
2. The number of fooling probes required to compensate for differences between the correct sequence and the self-sustaining segment, each of which contributes a factor of  $\alpha$ .

We will use the variant symbol  $\varpi_J$  to denote the coefficients for the offset of the self-sustaining segment in the tandem spectrum case.

When sequencing with tandem spectra, item 1 is not affected: two  $(\lambda-1)$ -character strings chosen from the  $m$ -character target sequence agree at each position with probability  $1/4$ . Thus, since no fooling probes are required for the base case of  $J = 0$ , the multiplier for  $J = 0$  does not change, and  $\varpi_0 = \pi_0$ .

The multipliers  $\varpi_{1\dots J}$  are a bit more complex to calculate, but not unduly so. For any value of  $J > 0$ , there are  $\nu = \lambda - 1 - J$  constrained positions to the left of the initial branching position. The branching position can be chosen in 3 different ways, and must be confirmed by a fooling probe in each spectrum. There are  $J$  characters to the right of the branching positions, for which an average of  $\theta J$  fooling probes are required *in each spectrum* to compensate for disagreements. Thus, we have

$$\varpi_{1\dots\lambda} = 3 \frac{\alpha^{2\theta J}}{4^\nu} \quad (7.6)$$

The total probability of a Mode 2 failure when using tandem spectra can be calculated using Equations 7.4 and 7.6:

$$\begin{aligned} P_2^{(t)} &= m^2 \sum_{J=0}^{\lambda} \varpi_J \\ &= \frac{3m^2}{4^{\lambda-2}} \left( \frac{1}{32} + \alpha^2 \frac{1 - (4\alpha^{2\theta})^\lambda}{1 - 4\alpha^{2\theta}} \right) \end{aligned} \quad (7.7)$$

Here, the primary advantage over the single-spectrum case is again derived from the fact that although the initial branching position may be chosen in three of four ways in the primary spectrum, it requires a fully constrained fooling probe in the secondary spectrum to confirm it. This introduces an additional factor of  $\alpha$ , so that if  $i$  fooling probes are required in the single spectrum case,  $2i + 1$  are required in the tandem spectrum case.

## 7.4 Performance Improvement

Figure 7.2 shows the performance of the single-spectrum and tandem-spectrum SBH algorithms compared. In order to present the two curves on the same graph, the single spectrum curve is displayed with the abscissa dilated by a factor of 2.

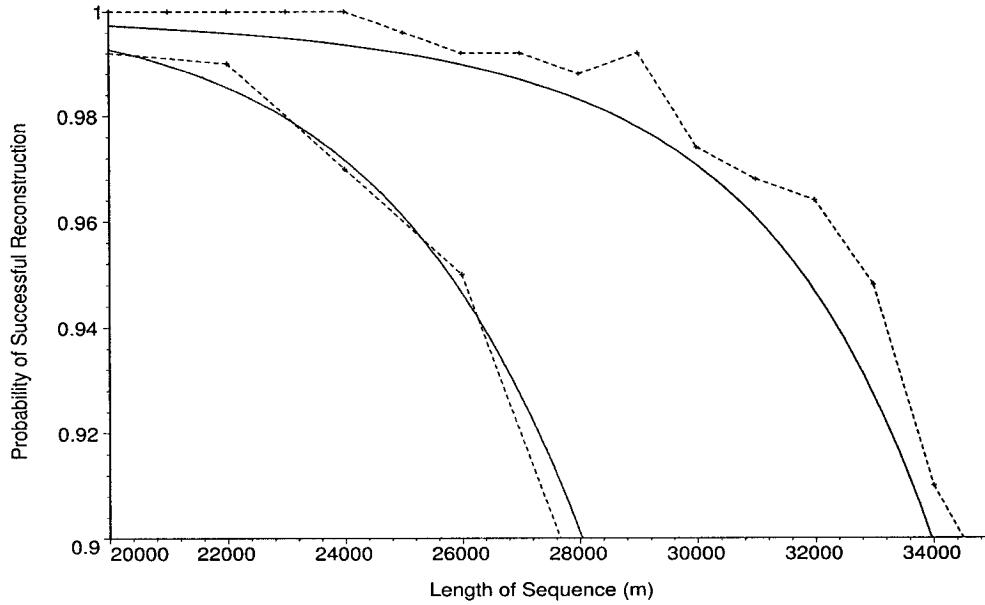


Figure 7.2: Performance of tandem-spectrum sequencing on random DNA using (4,4)-probes. The single-spectrum curve has been scaled by a factor of 2 to permit per-feature comparison with the tandem-spectrum method. Predicted values are shown using solid curves, and observed values with dotted lines.

The total probability of failure for single and tandem spectrum sequencing is simple to calculate. Recall from Chapter 4 that we assume (based on extensive observation) that Mode 1 and Mode 2 failures are independent, so the total probability of successfully sequencing an  $m$ -character target sequence is

$$e^{-(P_1+P_2)} \approx \left(1 - \frac{P_1 + P_2}{m}\right)^m$$

However, in the interesting range of values, where the the probability of success is close to 1, this can be accurately approximated for the single and tandem spectrum cases by summing  $P^{(s)} = P_1^{(s)} + P_2^{(s)}$ , and  $P^{(t)} = P_1^{(t)} + P_2^{(t)}$ .

Since the area and cost of the microarrays in the tandem-spectrum case is twice that for the single-spectrum method, we must take that into account. If we denote as



$m_1$  and  $m_2$  be maximum achievable fragment length using single and tandem spectra for a particular value of  $\epsilon$ , we must compare  $m_2$  to  $2m_1$ . What is desired is a simple factor  $\chi = m_2/2m_1$ . If  $\chi > 1$ , then the extra cost inherent in the tandem-spectrum method is more than justified. However, the expected performance increase realized by utilizing two spectra appears to be very cumbersome to calculate precisely.

Here we reprise the calculation originally made in [HPY02], where we calculate  $m_2/m_1$  analytically for Mode 1 failures, and ignore the correction factor for overlapping probes. If we expand the exponential from Equations 7.1 and 7.2 to second-degree terms, we derive the following approximations, which are accurate when  $m/4^\kappa < 0.6$ :

$$P_1^1 = \frac{3m}{2} \left( \frac{m}{2 \cdot 4^\kappa} - \frac{1}{2} \left( \frac{m}{2 \cdot 4^\kappa} \right)^2 \right)^\kappa \quad (7.8)$$

and

$$P_1^2 = 3m \left( \frac{m}{4^\kappa} - \frac{1}{2} \left( \frac{m}{4^\kappa} \right)^2 \right)^{2\kappa} \quad (7.9)$$

In the single-spectrum case, we introduce the unknown  $x = m/(2 \cdot 4^\kappa)$ . For a chosen confidence level  $\epsilon$ , we must solve Equation 7.8 for  $P_1^{(s)} = 1 - \epsilon$ ,

$$2x - x^2 - \frac{1}{2} \left( \frac{1 - \epsilon}{3x} \right)^{\frac{1}{\kappa}}$$

And for the tandem spectrum case, we let  $y = m/4^\kappa$  and solve  $P_1^{(t)} = 1 - \epsilon$ , as in

$$2y - y^2 - \left( \frac{1 - \epsilon}{3y} \right)^{\frac{1}{2\kappa}}$$

The above equations were solved numerically for  $\epsilon = 0.9$  and different practical values of  $\kappa$ . From these numerical solutions, we may compute the ratio

$$\frac{m_2}{2m_1} = \frac{y}{2x}$$

These values, which provide analytical confirmation of the experimental observations derived from our simulations, are displayed in the following table:

$k$	5	6	7	8	9	10
$x$	.1946	.2057	.2146	.2218	.2278	.2329
$y$	.5112	.5443	.5714	.5941	.6134	.6302
$m_2/2m_1$	1.319	1.323	1.331	1.339	1.346	1.352

When sequencing with two independent spectra performance improvement is always greater than 2, indicating an increase which outweighs the increase in chip cost by at least 30%. Unfortunately, while it is tempting to hope for further performance increases with even more spectra, for 3 or more spectra, the increase in chip cost is barely justified by the increased length of  $m$ , if at all.

## 7.5 Integration with Polling and Repeat-Recovery

The polling and repeat-recovery provisions detailed in Chapters 5 and 6 can be applied to tandem-spectrum sequencing, although only the polling provision has been implemented. We should expect the repeat-recovery algorithm to have the same (small) improvement in performance for the tandem-spectrum method as it does for the single-spectrum method, but since there is no way of predicting its effectiveness, we leave the implementation and discussion of the technique for future work.

To begin, we will briefly recall what the polling provision *is*. The polling provision is a method for resolving Mode 1 and Mode 2 failures. When a failure of either type occurs, there is a set of probes which sample the ambiguous character; these probes form the extension set for that character. The number of used probes (refer to Definition 5.1) in the extension set for each alternative extension are counted, and the extension with the fewest used probes is selected as the correct one.

In the single spectrum case, there are  $\kappa$  probes in the extension set for a Mode 1 failure, and  $\lambda$  for a Mode 2 failure. In the tandem spectrum case, there are an equal number of probes in each spectrum. Thus, there are a total of  $2\kappa$  probes for Mode 1 failures and  $2\lambda$  probes for Mode 2 failures. The polling algorithm may be used in an essentially unchanged manner when two spectra are used; in fact, the extra probes should slightly *increase* the effectiveness of the method. The only caveat is that in the case of Mode 1 failures, the  $\kappa$  probes in each spectrum which form the extension set have different offsets. The following example illustrates this effect.

**Example 7.1.** Using tandem (3,2)-probing patterns (NNN . . N . . N and N . . N . . NNN), the algorithm encounters the following situation:

Path Tree											0	1	2	3	4	5	6	7	8	9...	
...	A	A	C	G	G	T	T	A	C	A	[T]	T	G	A	T	A	T	G	G	A...	
											[G]	T	G	A	T	A	T	G	G	A...	
Probes																					
1			C	.	.	T	.	.	C	A	[G]										
2				G	.	.	T	.	.	A	[G]	T									
3					G	.	.	A	.	.	[G]	T	G								
4								A	.	.	[G]	.	.	A	T	A					
5											[G]	.	.	A	.	.	T	G	G		
6			C	G	G	.	.	A	.	.	[G]										
7						T	T	A	.	.	[G]	.	.	A							
8									C	A	[G]	.	.	A	.	.	T				
9										A	[G]	T	.	.	T	.	.	G			
10											[G]	T	G	.	.	A	.	.	G		

Probes 1-5 belong to the reverse spectrum  $S_R$  and probes 6-10 to the direct spectrum  $S_D$ . The  $\kappa$  probes in extension set from  $S_D$  are located at different offsets relative to the branching position than those from  $S_R$ . The right-most characters of the forward probes fall at positions  $\{0, 1, 2, 5, 9\}$  while the reverse probes fall at  $\{0, 3, 6, 8, 9\}$ . When counting the number of used probes in the extension sets for [T] and [G], all  $2\kappa$  probes are included. ■

The predicted performance of tandem-spectrum SBH with the polling provision is compared with the observed behaviour in Figure 7.3. Recall that in the single-spectrum case, we predicted and observed about a 25% performance improvement. We derive a nearly identical result for the use of the polling provision with tandem-spectrum SBH, indicating that the benefits of the tandem-spectrum sequencing technique and the polling provision are effectively independent, and contribute multiplicatively to the maximum length of a sequenceable fragment. The following table shows, the maximum fragment length which can be sequenced with confidence  $\epsilon = 0.9$ , for both the single-spectrum and tandem-spectrum gapped-SBH methods with and without the polling provision (for random DNA and (4,4)-probes). For the single-spectrum case, the values shown are  $2m_1$ ; twice the actual length achievable, to permit easy comparison with the tandem-spectrum method.

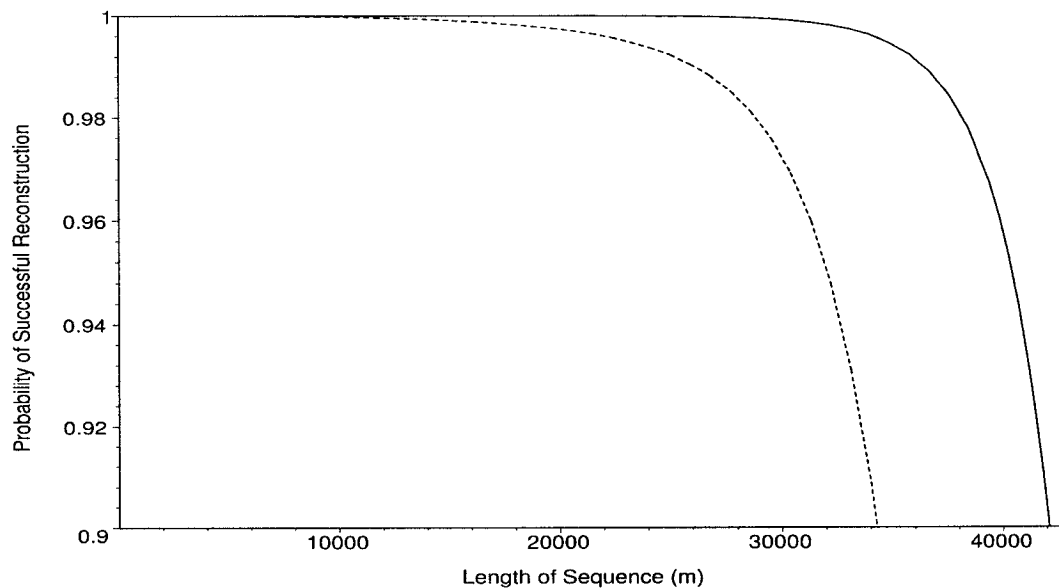


Figure 7.3: Performance of tandem-spectrum sequencing with (solid line) and without (dotted line) the polling provision on random DNA using (4,4)-probes.

Sequencing Method	$m$
Single-spectrum without polling	27800
Single-spectrum with polling	34200
Tandem-spectrum without polling	34600
Tandem-spectrum with polling	42600

The combination of the two algorithmic enhancements yields about a 55% improvement over the standard gapped sequencing technique. Moreover, the sequence length achieved by the tandem-spectrum sequencing algorithm with the polling provision is nearly 2/3 of the information-theoretic bound.

### 7.5.1 Performance on Natural DNA

The results of tandem-spectrum sequencing experiments on natural DNA sequences are shown in Figures 7.4 and 7.5. Note that in these two figures, no abscissa dilation has been performed. Separate experiments on the DNA of various natural organisms using either a single (4,4)-reverse probing pattern or (4,4)-tandem spectra (two spectra realized from the forward and reverse probing form of a (4,4)-gapped probe pattern).

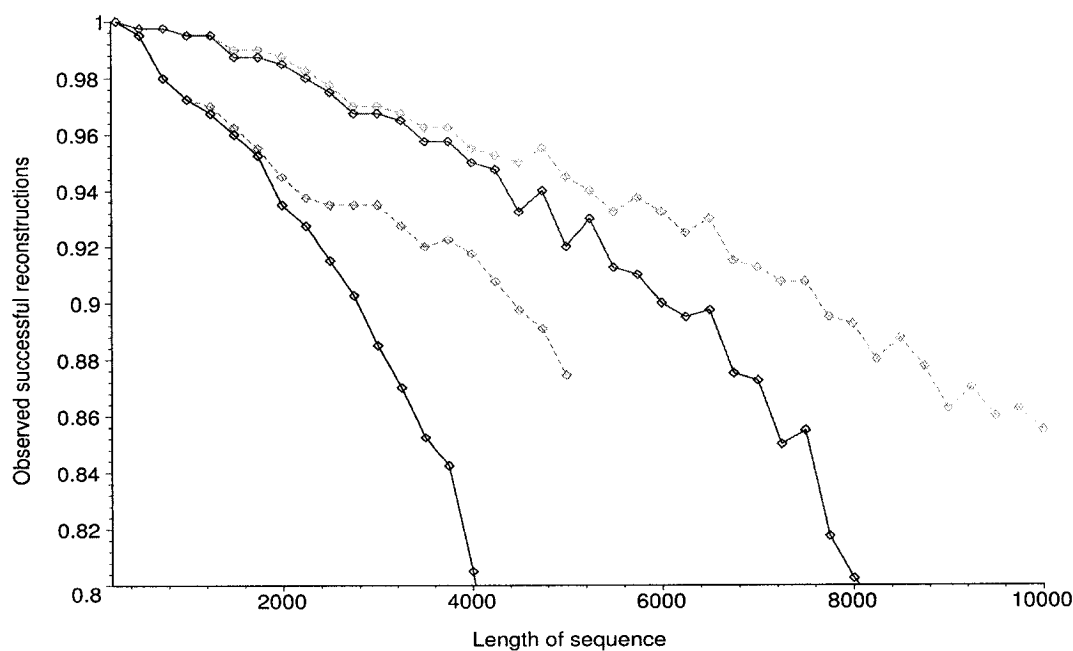


Figure 7.4: Observed performance of repeat recovery on *h. influenzae* (the two left-most curves) and *salmonella* (the two right-most curves). The single-spectrum curves are drawn with solid lines and the tandem-spectrum curves with dotted lines.

The following table shows the longest fragment length achievable with  $\epsilon = 0.9$  for both the single spectrum and tandem spectrum method, using (4,4)-probes.

	(4,4)-Single ( $2m_1$ )	(4,4)-Tandem ( $m_2$ )
Random DNA	27700	34600
<i>a. thaliana</i> chr. III CDS	3400	1800
human chr. 3 CDS	2800	1400
<i>s. cerevisiae</i> chr. IV CDS	4600	2900
<i>e. coli</i>	10800	6100
<i>h. influenzae</i>	5600	4400
<i>salmonella</i>	12000	7600
<i>s. solfataricus</i>	6000	3900

In a disappointing result, for all natural DNA sources,  $m_2 < 2m_1$ . In fact, in certain cases (human chromosome 3, for instance),  $m_1 \approx m_2$ , indicating that no performance benefit *at all* is gained by adding a second spectrum. The most likely

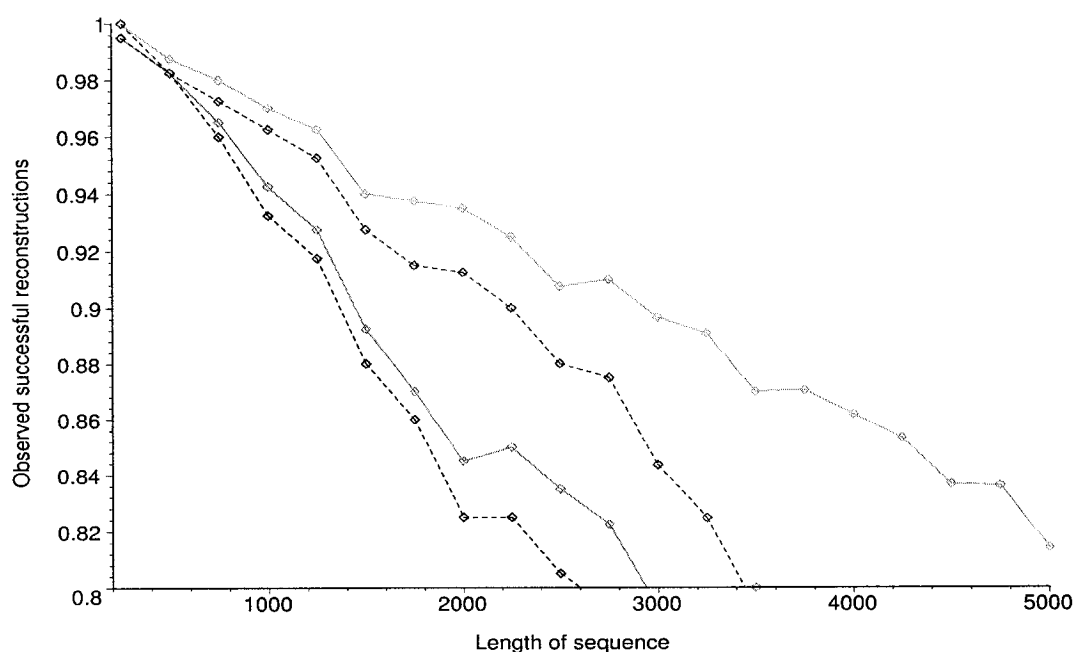


Figure 7.5: Observed performance of repeat recovery on human chromosome 3 coding sequences (the two left-most curves) and *s. cerevisiae* chromosome IV coding sequences (the two right-most curves). The single-spectrum curves are drawn with solid lines and the tandem-spectrum curves with dotted lines.

explanation for this phenomenon is that the DNA of these organisms contains a large number of repeats, either of the  $(\lambda - 1)$ -character strings which cause Mode 2 failures or of short-period local repeats. In either case, since all of the fooling probes necessary to cause a sequencing failure are guaranteed by the target sequence itself to be in the spectrum, then they are guaranteed in *both*  $S_D$  and  $S_R$ . In this case, the independence of the probing patterns does not help.

## Chapter 8

# Constructing a Seed

We have until now assumed the availability of a *seed*—a short segment of known DNA from which we can begin sequencing. This is not unreasonable, since both methods of DNA amplification—cloning and PCR—require that we know the exact sequence of short pieces of DNA (primers) at each end of the sequence. These primers are typically attached biochemically prior to amplification. However, it is possible to construct a seed for a target sequence from only the information available in its spectrum. Seeds thus built will be located at random positions within the target sequence, and must be extended in both directions to complete the reconstruction. Other than this minor modification, constructed seeds are identical to manually affixed primers.

In addition to providing the SBH technique with a degree of algorithmic completeness, the ability to build new seeds also allows us to extract additional information from a spectrum when all of the other algorithms for extending a sequence have failed. Sequencing failure occurs when the information contained in the spectrum is insufficient to produce an unambiguous reconstruction of the target sequence. While there is no way of resolving the failure computationally, we can try to extract as much information as possible from the spectrum when one occurs. Instead of simply abandoning the sequencing effort, we can restart sequencing from a new seed, expressly created in the portion of the sequence which has not yet been reconstructed. In this manner, rather than producing a single sequence fragment, we might produce two or more fragments which cover a correspondingly larger proportion of the target sequence. This strategy is left for future work.

## 8.1 Constructing a Seed

Seed construction begins by selecting a random probe  $p$  from the spectrum. The selected probe has length  $\lambda$  and  $\lambda - \kappa$  “don’t care” positions, where  $\lambda > 2 \cdot \kappa$  for any useful probing pattern. The “don’t care” positions in this initial probe may also be considered to be “unknown” bases; they correspond to nucleotides in the target sequence whose values are not yet determined. To construct a seed, we need to generate a  $\lambda$ -character oligomer from the initially selected probe, so we must somehow fill in all of the initial probe’s unknown positions. Once this process is complete, there is a single valid seed among a large set of invalid seeds; the invalid seeds are strings of  $\lambda$  characters which are *not* substrings of the target sequence, but are produced by random combinations of fooling probes filling the unknown positions in  $p$ . This large set of potential seeds must be processed so that invalid seeds are eliminated from consideration.

Note that while filling the initially unknown bases, *incorrect* values may be assigned to some or all of these positions in the initial probe. To differentiate between completed seeds (which necessarily correspond to actual subsequences of the target sequence) and potential seeds (which may not be subsequences of the target), the potential seeds created during the seed construction process will be called *seedlets*.

**Definition 8.1.** *A seedlet is a sequence of bases (natural and universal) produced by the seed-construction algorithm. It may be either a valid or invalid seed, where valid seeds are sequences consisting only of natural bases, that are actual subsequences of the target. Invalid seeds are sequences that either contain one or more universal bases or do not correspond to any subsequence of the target.*

The set of seedlets grows larger as the initially unspecified positions are filled, so the first stage of this process will be called the *growth* stage. The second stage of the process, whereby we eliminate the invalid seedlets from consideration and reduce the set to a single valid seed will be called the *reduction* stage. There may actually be more than one valid seed in the set at the end of the growth stage, and all valid seeds will remain in the set at the end of the reduction stage. This will be discussed in more detail later in this chapter, but for now we make the simplifying assumption that the growth process produces only a single valid seed.



In Chapter 1, a general description of *spectrum queries* is given, along with a more specific discussion of queries which contain a single free position. In order to perform the growth steps of the seed construction algorithm, spectrum queries which contain more than one free position must be performed. These will be referred to as *multi-position queries*, and are formally defined here.

**Definition 8.2.** A multi-position query is a spectrum query containing more than one free position. The response to a query with  $n$  free positions is the set of all probes in the spectrum which match the query string over all non-free positions.

### 8.1.1 Detailed Example of seed Construction

We now give a detailed example of seed construction for a short DNA sequence. The 70-base DNA sequence GACGTGCCTG ACGCAATAAA GTCATTCCCC GGCTTGATGT CCGGATTTCGG GCGTCGCCGT TCTCTTGTA produces the following (3, 2)-reverse spectrum:

{A . . A . . GTC, A . . A . . TCA, A . . C . . CGG, A . . C . . GCG, A . . C . . TAA, A . . C . . TCC, A . . G . . ATT, A . . T . . CTG, A . . T . . GGA, A . . T . . TTC, C . . A . . AAA, C . . A . . CAA, C . . A . . CGG, C . . C . . CGT, C . . C . . CTT, C . . C . . GAT, C . . C . . TCT, C . . G . . GCA, C . . G . . GTC, C . . G . . TCG, C . . G . . TGA, C . . G . . TTG, C . . T . . AGT, C . . T . . CCG, C . . T . . CTT, C . . T . . TAA, C . . T . . TCT, G . . A . . AAG, G . . A . . CCC, G . . C . . ATT, G . . C . . CGC, G . . C . . GAC, G . . C . . TTG, G . . G . . ATA, G . . G . . CCT, G . . G . . CGG, G . . G . . CTC, G . . G . . GCC, G . . G . . GTT, G . . T . . ATG, G . . T . . CCG, G . . T . . CGC, G . . T . . GGC, G . . T . . GGG, G . . T . . TGT, T . . A . . CAT, T . . A . . TCC, T . . C . . AAT, T . . C . . ACG, T . . C . . GAT, T . . C . . GCT, T . . C . . GGC, T . . C . . GTA, T . . C . . TTC, T . . G . . CGT, T . . G . . GTC, T . . G . . TTC, T . . T . . CCC, T . . T . . CCG, T . . T . . TGT}

The probing pattern has length  $\lambda = 9$  characters. If no probes were duplicated, the spectrum would contain  $70 - 9 + 1 = 62$  probes. In fact, the probes C . . G . . TCG and C . . G . . TGA each occur at two positions in the sequence, and so the spectrum contains only 60 distinct probes.

To begin seed construction, a probe is selected at random from the spectrum. In this example, we assume that the probe C . . A . . AAA has been selected. We say that this probe occurs at position 12 in the sequence (i.e. it was produced by aligning the left-most position of the probing pattern with the eighth character in the sequence).

From this initial probe we need to build a seed: a 9-character sequence containing no unknown characters (universal bases). Four of the characters in the initial probe are universal bases, so we need to assign values to these four unknown positions in the probe. This process can be completed in two growth steps.

The first step in the growth stage will fill in positions 2 and 5 in the initial probe. The query string used to accomplish this is generated by shifting the (3,2)-probe pattern one position to the right with respect to the initial probe, as shown here:

C	.	.	A	.	.	A	A	A
N	.	.	N	.	.	N	N	N

This yields the query string ?.A?.AAA?, containing three free positions. They are represented by the character ?, and correspond to the two positions within the span of the initial probe which will be assigned values as well as one free position outside the span of the initial probe. There are two additional unknown positions in the query string—represented by the ‘.’ character—which coincide with universal bases in the probing pattern and so do not affect the results of the query. In fact, only two of the natural bases in the probing pattern align with natural (and known) bases in the initial probe: positions 8 and 9 of the query. Those two positions are fixed to the values of the natural bases in the the initial probe to which they align (AA). Note that these are the only two positions in the query string which restrict the possible matches in the spectrum; the query string can be rewritten in the form ?.?.?.AA? without affecting the results of the query. By transforming the query string into the same structure as the probes in the spectrum, we highlight the fact that only the five positions in the query string which align with natural bases in the probing pattern affect the query results. All other positions in the string are irrelevant. Thus, any probe which matches the query string in only the seventh and eighth positions is a valid result to the query.

There are three probes in the spectrum which fit the criteria, creating the result set: {C..A..AAA, G..A..AAG, T..C..AAT}. Each of these probes provides us with a different assignment for the three free positions in the query. Each of the three resultant probes provides a potentially correct assignment of values to the free positions in the query string. There is no way of determining which assignment is the correct one, so all three must be considered. Each of the matching probes is aligned

with the initial probe with the same offset that was used to produce the query string (in this case, one position to the right), and a new string with values assigned to free positions in the query is produced from the aligned strings. This is defined as the *combination* of two strings. Two strings are combinable only if there is no aligned position which contains different natural bases in the two strings.

**Definition 8.3.** *The combination of two strings  $s_0$  and  $s_1$  with lengths  $l_0$  and  $l_1$  is calculated for a particular offset  $\kappa \leq \min(l_0, l_1)$  of  $s_1$  to the right of  $s_0$ . Each string is extended to the left or right as needed with universal bases so that each string has length  $l = \max(l_0, l_1) + k$ . We denote the two extended strings  $s'_0$  and  $s'_1$ . If  $s'_0$  and  $s'_1$  are combinable, then the result of the combination is a third string ( $s_c$ ) of length  $l$ . The characters of  $s_c[i], i = 1..l$  take the value of  $(s'_0[i] \& s'_1[i])$ , where the '&' operator is defined according to the following table. In the table,  $N$  denotes any natural base and  $U$  denotes a universal base.*

&	N	U
N	N	N
U	N	U

**Example 8.1.** The two strings  $s_0 = \text{CC} \dots \text{AC} \dots \text{AA}$  and  $s_1 = \text{CATA} \dots \text{AAA}$  are combined with an offset of 1. This corresponds to the following alignment:

C	C	.	.	A	C	.	A	A	
	C	A	T	A	.	.	A	A	A

Each string must be extended with a single universal base.  $s'_0 = \text{CC} \dots \text{AC} \dots \text{AA}$  is produced by appending a universal base to  $s_0$ , and  $s'_1 = \text{.CATA} \dots \text{AAA}$  is produced by prepending a universal base to  $s_1$ .

The first character in the resulting string  $s_c$  is C, since character in string  $s'_0$  aligns with the universal which was added to create  $s'_1$ . The second character in the resulting string is C, since both  $s'_0[2]$  and  $s'_1[2]$  have the value C. The third and fourth characters are AT, since string  $s'_0$  contains universal bases in those positions. The fifth character is A, which is a match between both aligned strings, and so on. The final string  $s_c$  is CCATAC.AAA. ■

Combining the query string C . . A . . AAA with each of the probes in the result set, using an offset of 1, yields the following set of three seedlets: {CC . AA . AAAA, CG . AA . AAAG, CT . AC . AAAT}.

The second step of the growth stage requires three spectrum queries; one for each of the seedlets in the set. The three query strings are generated by shifting the probing pattern another character to the right (two positions to the right with respect to the initial probe), as shown here for the seedlet CT . AC . AAAT:

C	G	.	A	A	.	A	A	A	G
N	.	.	N	.	.	N	N	N	

The query string which is produced by this alignment is ?AA?AAAG?. Again, there are only two characters in the seedlet which align with natural bases in the probing pattern. In this case they are the characters AG, corresponding to the seventh and eighth positions of the pattern. The query may also be written as ? . . ? . . AG?, and matches only one probe in the spectrum: C . . T . . AGT. When combined with the query string, the seedlet CGCAATAAAGT is produced. This is the first seedlet with no unknown positions remaining; the other two queries will produce additional fully-specified seedlets.

The query string corresponding to the seedlet CC . AA . AAAA is ?AA?AAAA?, which matches the probes {C . . A . . AAA, G . . A . . AAG, T . . C . . AAT}. When combined with the query string, the seedlets CCCAAAAAAA, CCGAAAAAAAG and CCTAACAAAAT are generated. Note that this is the same set of probes which matched the first search performed, but the results to this query are used to fill in different positions in the initial probe.

Finally, the query string for the seedlet CT . AC . AAAT is ?AC?AAAT?, which matches the four probes: {A . . G . . ATT, G . . C . . ATT, G . . G . . ATA, G . . T . . ATG}. These four probes yield an additional four fully-specified seedlets: {CTAACGAAATT, CTGACCAAATT, CTGACGAAATA, CTGACTAAATG}.

When the results of all three queries are pooled, we get the following set of eight seedlets: {CCCAAAAAAAA, CCGAAAAAAAG CCTAACAAAAT, CGCAATAAAGT, CTAACGAAATT, CTGACCAAATT, CTGACGAAATA, CTGACTAAATG}.

None of these seedlets contain any universal bases and each of them is long enough to serve as a seed. However, most likely only one of them is an actual substring of

the target sequence. The other seedlets were produced by random combinations of results to the queries performed in the two growth steps. The reduction stage of seed construction aims to eliminate false seeds from the set.

Like spurious branches in the sequencing process, false seeds will only be extended probabilistically. To eliminate them, we simply attempt to extend each of them in parallel. Each extension step produces a single-character query string for each of the seedlets. The query string corresponding to the seedlet CTAACGAAATT is ACGAAATT?. There is only one free position in the query, so any probe in the spectrum which matches the query must match over four positions, compared with only the two positions which were required to match during the growth stage. Fewer matches are expected to each of the extension queries since the query strings contain fewer specified positions, and there are in fact no matches in the spectrum to this query string, so this seedlet is eliminated.

The only seedlet which *can* be extended by a single character is CGCAATAAAGT. (The probe A . . A . . GTC matches the query string AATAAAGT?; the next character is C .) After only a single reduction step, the seed construction process has completed: since there is only one remaining seedlet, it must be a valid seed. Note that this seed is a substring of the target sequence, beginning at the twelfth character of the target. The initial probe also corresponded to the twelfth character of the target sequence: the seed construction process served to fill in the unknown characters in the initially selected probe.

## 8.2 Complexity of the Basic seed Algorithm

We would like to predict the work required to produce a seed; this most easily accomplished by estimating the maximum size of the seed set. The seed construction process always begins with a single probe; the ultimate size of the seed set is dependent on the length of the target string (which determines  $\alpha$ , from Definition 3.3), the probing pattern, and the strategy used to construct the seed. The notion of a *strategy* will be described more fully below.

The growth stage of seed construction is analogous to the branching mode of the *extend* algorithm. The primary difference is that each spectrum query has multiple

unspecified positions, which leads to much larger answer sets. Each free position in the query string increases the number of potential matching probes by a factor of 4.

Queries will be referred to as *n-position* queries, where *n* is the number of free positions in the query. For instance, the A..G..?T? is a two-position query. In general, a query with *f* free positions (an *f-position* query) can potentially yield  $4^f$  results. Every seedlet in the seed set produces a different query string, and every probe which matches a query provides a different potential assignment of values to the unknown positions in the initial probe. Thus, each probe resulting from a query must be combined with the query string in order to fill in additional free positions in the seedlet. This implies that each step in the growth stage increases the size of the seed set by some multiplicative factor. Seed construction always begins with a single probe, so all that is required to estimate the maximum size of the seed set is to calculate the multiplicative factors.

The most obvious method of filling in the unknown positions in a (4,4)-probe is presented here as an example. The process begins with a string of the form of the direct (4,4)-probe pattern NNNN...N...N...N...N. From there, three shifts are performed. Each shift is a single character to the right, and yields a query with five free positions. The initial probe is on the first line with N's standing for natural bases, and '.'s for universal bases. The three query steps are shown in order below. The free positions in each query are shown using the '?' character. The form of the seedlets with the corresponding positions filled is shown directly below each query.

Step	
	<u>    N N N N . . . N . . . N . . . N . . . N</u>
1	N N N ? . . . ? . . . ? . . . ? . . . ? N N N N N . . N N . . N N . . N N . . N N
2	N N N ? . . . ? . . . ? . . . ? . . . ? N N N N N N . N N N . N N N . N N N . N N N
3	N N N ? . . . N . . . ? . . . ? . . . ? <u>    N N N N N N N N N N N N N N N N N N N N</u>

At each step, every match to a spectrum query is combined with the query string and added to the new seed set. Once these three growth steps have been completed,

at least one of the seedlets in the set corresponds to an actual subsequence of the target sequence. Most (or all) of the others are spurious, and will be eliminated during the reduction process.

The expected size of the seed set can be calculated as a function of  $\alpha$ . Recall from Chapter 3 that  $\alpha = 1 - e^{-m/4^\kappa}$  measures the probability of the finding a particular  $\kappa$ -character subsequence of the target sequence. The first spectrum query has 5 free positions, so there may be as many as  $4^5 = 1024$  matches. Each potential match can be considered an independent query, and yields a positive response with probability  $\alpha$ . The total number of responses to the query is follows a binomial distribution with parameter  $p = \alpha$ . Thus, the expected number of responses to this query is  $\alpha \cdot 4^5$ .

The same calculation holds for the second query as well, except that we must query the spectrum once *for every* result to the first query. The sizes of the result sets to the two queries are multiplicative factors, leading to a seed set containing an expected  $\alpha^2 \cdot 4^{10}$  seedlets after two queries. By the same reasoning, after the third (and final) query, we expect to have approximately  $\alpha^3 \cdot 4^{15}$  seeds in the set.

Once the growth stage is completed, the reduction stage is executed, in which each seedlet is *extended* a single character at a time, breadth-first. As they are extended, the seedlets which are *not* substrings of the target sequence will be extended only probabilistically as long as there are probes in the spectrum to support them. The seedlets which are actually present in the target sequence—those which are valid seeds—will be extended with certainty. The reduction stage continues until all of the invalid seedlets are eliminated, leaving only valid seeds which may be used interchangeably as starting points for sequencing.

With every single-character shift performed in the reduction stage, each of the invalid seedlets is extended if and only if there is a fooling probe present in the spectrum confirming it. Such a probe must match a query string with  $\kappa - 1$  specified positions. The query strings used to confirm each seed during every reduction step have the exact same form as the query strings used during sequence extension: the  $(\lambda - 1)$ -character suffix of the seedlet with a single free position appended. For example, query strings of this type for seedlets constructed using (3, 4)-probe have the form NNNNNNNNNN?: an 11-character suffix of the seedlet followed by a universal base.

For *any* probing pattern, there are four possible matches to a query with a single free position. Each of the potential matches is present in the spectrum with probability  $\alpha$ . Thus, the expected number of extensions to an invalid seedlet in a reduction-step query is  $4\alpha$ . For all reasonable target sequence lengths,  $4\alpha < 1$ , so the number of seedlets is expected to be reduced by a factor of  $4\alpha$  with every shift.

If the size of the seed set when the growth stage has completed is  $|set_{max}|$ , then the average size of the seed set ( $|set_s|$ ) after  $s$  reduction steps is given by the equation:

$$|set_s| = |set_{max}| \cdot (4\alpha)^s.$$

By letting  $|set_s| < 1$  and solving for  $s$ , we get the number of shifts expected to eliminate all of the invalid seedlets. Note that  $s$  need not be calculated in advance (using the expected final size of the seed set), but should instead be calculated during the execution of the seed algorithm, after the growth stage has completed. The average number of reduction steps  $s$  required to eliminate all invalid seeds from the set is:

$$s > \ln(|set_{max}|) \cdot \ln\left(\frac{1}{4\alpha}\right) \quad (8.1)$$

The number of seedlets remaining in the set after each reduction step is an exponentially decreasing sequence. And even though it takes an expected  $s$  reduction steps to eliminate all of the invalid seeds, some may remain. Making only the assumption that the seedlets are independently extended or eliminated during the reduction process, it is possible to gain a measure of confidence that only valid seeds remain in the set after additional shifts are performed.

Consider an individual (invalid) seedlet present in the set at the beginning of the reduction process. The probability that it (or one of its descendants) remains in the set after  $s$  reduction steps is bounded above by  $p_s = (4\alpha)^s$ . Since there were originally  $|set_{max}|$  seeds in the set, the probability that all invalid seeds have been eliminated from the set is  $\pi_s \approx (1 - (4\alpha)^s)^{|set_{max}|}$ . However,  $(4\alpha)^s \approx \frac{1}{set_{max}}$ , so  $\pi_s \approx 1/e$ , for any reasonably large  $|set_{max}|$ . This indicates that there is only slightly more than a 1/3 chance that all of the invalid seeds have been eliminated after  $s$  steps. We would like to improve on this.



If we want to achieve some confidence factor ( $\gamma$ ) that no invalid seeds remain in the set, then we need to execute a total of  $s'$  reduction steps such that  $((4\alpha)^{s'})^{|set_{max}|} \leq \gamma$ . This is equivalent to determining the number of steps required such that:

$$(1 - (4\alpha)^{s'})^{|set_{max}|} \leq \gamma \quad (8.2)$$

Which can be restated (somewhat inelegantly) as:

$$s' \geq \frac{\ln(1 - e^{\frac{\ln(\gamma)}{|set_{max}|}})}{\ln(4\alpha)} \quad (8.3)$$

For a seed set of a given size, and a particular  $\alpha$ , the number of steps required to eliminate all of the invalid seeds from the set can be easily calculated. Typically, a confidence factor of at least  $\gamma = 0.9$  is desired, indicating that any of the remaining seeds may probably be used as a valid seed. Higher confidence is not necessarily required; if sequencing fails using one of the seeds, another attempt at reconstruction may be made using one of the other seeds from the set. The next chapter will discuss in greater detail the advantages which can be gained by using multiple seeds in sequencing.

**Example 8.2.** Consider the (3,3)-probing pattern NNN . . N . . N . . N. The growth stage stage of seed set construction for these probes consists of only two fill steps, as follows:

Step	N N N . . N . . N . . N														
1	N	N	N	?	.	.	?	.	.	?	.	.	?	.	?
	N	N	N	N	.	N	N	.	N	N	.	N	N	.	N
2	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Each query string produces an expected  $\alpha \cdot 4^4$  responses, so the expected size of the seed set when the growth stage has completed is  $\alpha^2 \cdot 4^8$ . If a seed is generated for an 800-character sequence (and corresponding to  $\alpha \approx 0.1774$ , we expect to produce

2063 seedlets. By using Equation (4.3) to solve for  $s' > \frac{\ln(1-e^{\frac{\ln(0.9)}{2063}})}{\ln(4\alpha)} = 28.81$ , we find that it will take an expected 29 steps to eliminate the invalid seeds from the set, with confidence  $\gamma = 0.9$ .

If the growth stage seed construction produces more or fewer seedlets than expected, the value of  $s'$  may change. For instance, if the seed set contains 4000 seedlets after the two growth steps, solving for  $s' > 30.74$  indicates that we need to perform an expected of 31 reduction steps to eliminate all invalid seeds from the set, with confidence 0.9. ■

There are two reasons why the seed set may contain more than a single element even after  $s'$  shifts. First, the initially selected probe may occur at some additional location in the target sequence (an event which occurs with probability  $\alpha$ ). If this occurs, then upon completion of the growth phase, there will be a seedlet in the set corresponding to each position in the target sequence which matches the probe.

**Example 8.3.** Recall from the detailed example earlier in this chapter that there were two probes in the spectrum which occurred at two points in the sequence (the probes C . . G . . TCG and C . . G . . TGA). If either of these probes were selected as the initial probe during seed construction, then at the end of the growth stage there would be (at least) two valid seeds in the set, each corresponding to a subsequence of the target at each of the locations where the initial probe is found. ■

Secondly, there is a *size* issue. For instance, using (4,4)-probes, when  $m = 6000$  (implying that  $\alpha = 1 - e^{-6000/4^8} \approx 0.09$ ) the seed set will grow to approximately 783000 seedlets after the 12 unknown positions in the initial probe are filled. With so many seedlets in the set, it is very likely that at least one of the invalid seedlets leads to an event akin to a Mode 2 failure (introduced in [HP01] and discussed in detail in Chapter 4). That is, there is a set of fooling probes in the spectrum which is sufficient to *cover* the difference between one of the invalid seedlets and a similar substring of the target sequence, such that an invalid seed can nevertheless lead into a valid point in the sequence.

Fortunately, neither of these events produces an invalid seed; any seed so produced may be used to start sequencing. It is pointless to extend each potential seed until all but one of them reaches the end of the target sequence and is thereby eliminated.

Rather, we should continue the reduction phase of seed construction only long enough to be sufficiently confident that any remaining seedlets are overwhelmingly likely to be real substrings of the target sequence. This is accomplished by shifting each seedlet  $s'$  times. Once this threshold has been reached, all remaining seedlets are assumed to be correct, and may be used interchangeably as starting points for *extend*.

There is one more potential problem. If there are  $\zeta$  seedlets remaining in the set after  $s'$  shifts are completed, then there are  $\zeta$  sequences being extended in parallel. Each of the sequences is just as likely to produce a branch as the single putative sequence produced by the standard *extend* algorithm. To avoid polluting the set of valid seeds with one of these invalid branches, we restrict the selection of seeds to those from which we can extract  $\lambda$ -character segments containing no unresolved branches.

### 8.2.1 Different Shift Strategies

The naive approach to seed construction is limited in application to shorter sequences (of length  $m \leq 7000$ ) since the size of the seed set becomes too large as  $\alpha$  increases. Of course, increasing the value of  $\kappa$  means that much longer sequences can be processed, but that just postpones the problem. Without changing the probing pattern, we want to construct a seed in such a manner that minimizes the maximum expected seed set size. This is done by finding a better method to fill in the unknown positions in the initial probe. The series of shifts constituting the growth stage of seed construction will be called a *shift strategy*.

**Definition 8.4.** *A shift strategy is described by a sequence of offsets relative to the initial probe. For each offset position, a query is performed, and the results are combined with the query string so that all positions which are specified in the query or resulting probes are now instantiated.*

**Example 8.4.** Using this nomenclature, the shift strategy described above for (4,4) probes is [1,2,3]. ■

One obvious solution is to take advantage of the structure of the probe, and perform shifts of more than a single character at a time. For instance, by shifting

four times instead of three, and by shifting 4 characters to the right each time, we get the strategy [4,8,12,16] for (4,4)-direct probes. It looks like this:

Step	
	N N N N . . . N . . . N . . . N . . . N
1	? ? ? N . . . N . . . N . . . N . . . ? N N N N N N N N . . . N . . . N . . . N . . . N
2	? ? ? N . . . N . . . N . . . N . . . ? N N N N N N N N N N N N . . . N . . . N . . . N . . . N
3	? ? ? N . . . N . . . N . . . N . . . ? N N N N N N N N N N N N N N N N . . . N . . . N . . . N . . . N
3	? ? ? N . . . N . . . N . . . N . . . ? N . . . N . . . N . . . N . . . N

This may not immediately appear to offer an advantage over the previous shifting strategy: there are now 4 shifts instead of 3, an *increase* of one step. However, note that each shift has only 4 free positions instead of 5, so each step increases the expected size of the seed set by a factor of  $4^4 \cdot \alpha$ . The final size of the seed set is expected to be  $\alpha^4 \cdot 4^{16} \approx 282000$  when  $\alpha = 0.09$ . The additional shift introduced one extra factor of  $4\alpha$  into the equation, reducing the maximum set size by about 2/3.

In fact, since  $\alpha < 1/4$  for any realistic target sequence length, additional factors of  $4\alpha$  will always result in a decrease in the maximum expected size of the seed set. This simple observation forms the basis of a search for a better seed construction strategy.

The minimum constraint on seed construction is that all of the unspecified positions in the original probe must be filled. Positions are filled during the growth stage when a query is performed containing a free position corresponding to one of the unknown positions in the initial probe. For (4,4)-probes, there are 12 unknown positions, requiring a series of spectrum queries containing a total of at least 12 free positions: one for each unknown position in the initial probe.

In addition to the free positions which must be filled in the initial probe, additional free positions are inevitably introduced *outside the original probe's span*. These will

be called *new* free positions. Every time the probing pattern is shifted with respect to the original placement of the probe, at least one new free position is added. Thus, any complete shifting strategy consisting of  $\phi$  shifts for a probing pattern with length  $\lambda$ , must contain a total of at least  $\lambda - \kappa + \phi$  free positions. Each free position introduces a potential factor of 4 into the maximum seed set size.

On the other hand, every time a search is performed in the spectrum, we inhibit the growth of the seed set by a factor of  $\alpha$ , since each potential response to a query is present only with probability  $\alpha$ .

Note that free positions which fall outside the span of the initial probe do not contribute any useful information to the seed-building process. Yet they still add potential factors of 4 to the maximum set size. Moreover, observe that a shift which introduces *one* new free position contributes a factor of  $4\alpha$  to the maximum seed set size, while a shift with *two* new free positions contributes a factor of  $4^2 \cdot \alpha$ . While  $4\alpha < 1$  for any reasonable target sequence length,  $4^2 \cdot \alpha > 1$  when  $\alpha > 0.0625$ . When  $\alpha$  is small enough that  $4^2 \cdot \alpha < 1$ , seed construction is trivial. Taking this into consideration, shifts which introduce more than a single new free position are excluded from consideration.

Thus we have the following: a shifting strategy which uses  $\phi$  shifts to fill all  $\lambda - \kappa$  unspecified positions in the initial probe produces a total of exactly  $\phi$  free positions outside the original probe's span. Each of the  $\phi$  shifts also contributes a factor of  $\alpha$ , so the total expected maximum size of the seed set can be calculated as a product of two factors. In the expression below, the first factor,  $(4\alpha)^\phi$ , accounts for the shifting steps and the new free positions they introduce; the second factor,  $4^{\lambda-\kappa}$ , accounts for the unknown positions in the original probe. Thus, the maximum expected set size is:

$$|set_{max}| \approx (4\alpha)^\phi \cdot 4^{\lambda-\kappa} = \alpha^\phi \cdot 4^{\phi+\lambda-\kappa}$$

Note that each shift contributes a factor of  $4\alpha$ , *independent* of the number of free positions filled by the query. Typically,  $(0.1 < \alpha < 0.2)$ , and  $4\alpha$  is always less than 1. Note also that for a given probing pattern,  $\lambda - \kappa$  is a constant. This leads to the following theorem:

**Theorem 8.1.** *Given a probing pattern with length  $\lambda$  and  $\kappa$  specified positions, a*

shifting strategy  $S_0$  fills in all unspecified positions in  $\phi_0$  steps. If there exists another strategy  $S_1$  which fills in all unspecified positions in  $\phi_1 = \phi_0 + 1$  steps, the seed set produced by strategy  $S_1$  is  $4\alpha$  smaller than the set produced by  $S_0$ .

Therefore, the optimal strategy for the growth stage of seed construction is the strategy which uses the maximum number of shifts to produce a seed. An interesting special case now arises: it is possible during the growth stage of seed construction to perform spectrum queries which contain no free positions *inside* the initial probe's range; the response to such queries do not fill in any unknown positions in the initial probe. For example, take the following series of shifts for a  $(4, 3)$ -probing pattern:

Step	
	N N N N . . . N . . . N . . . N . . . N
1	N N N ? . . . ? . . . ? . . . ? . . . ? N N N N N . . N N . . N N . . N N . . N N
2	N N N ? . . . ? . . . ? . . . ? . . . ? N N N N N N . N N N . N N N . N N N . N N N
3	? ? N N . . . N . . . N . . . N . . . N

Step 3 (a shift of two positions to the left with respect to the initial probe: an offset of  $-2$ ) defines a shift which contains two new free positions, but no free positions within the range of the original probe. At first, it would seem that we would never want to consider such shifts: they have the potential to increase the seed set size and yet don't fill in any positions. However, if we treat these shifts slightly differently than other growth-stage steps, they can be used to *reduce* the size of the seed set.

### 8.2.2 Spectrum Scans

Until now, the growth stage of the seed construction process has included only steps which actually fill in unknown positions in the initial probe. Here we introduce a new class of spectrum query which does not serve to fill in the initial probe, which will be called *spectrum scans*. They are similar to queries in that they interrogate the spectrum. However, they do not produce sets of probes which match the query

string, but rather simply confirm or deny the existence of *any* matching probes in the spectrum.

**Definition 8.5.** A spectrum scan is a spectrum query which returns true if there are one or more probes in the spectrum which match the query string, and false if there are no probes in the spectrum which match the query string.

Spectrum scans can be used as shifting steps which serve only to reduce the current size of the seed set. They are used only to provide negative information about a seedlet in the set: based on the response to a scan, a seedlet is either removed from or left in the seed set. No unspecified positions are filled, and no new seedlets are created.

**Example 8.5.** Consider again the following series of shifts for a (4, 3) probing pattern:

Step		
	N N N N . . . N . . . N . . . N . . . N	
1	N N N ? . . . ? . . . ? . . . ? N N N N N . . N N . . N N . . N N . . N N	query
2	N N N ? . . . ? . . . ? . . . ? . . . ? N N N N N N . N N N . N N N . N N N . N N N	query
3	? ? N N . . . N . . . N . . . N . . . N N N N N N N N N . N N N . N N N . N N N . N N N	scan
4	N N N ? . . . ? . . . ? . . . ? . . . ? N	query

Note that the shift at Step 3 contains two new free positions, and no free positions within the initial probe's range. If this step were performed as a standard spectrum query, the seed set size would increase by an expected factor of  $4^2 \cdot \alpha$ , and add no useful information. However, if Step 3 is performed as a *scan*, each of the seedlets in the seed set after Step 2 must be confirmed by one of the 16 possible matches to the query in Step 3 (the *scan* returns *true*). If no match is found in the spectrum (the *scan* returns *false*), then that seedlet is eliminated from the set.

For instance, assume that the seedlet AGCTAG.CGA.AGT.TTA exists in the seed set after Step 2. If there are no matches to the query ??AG...G...A...T, then it is removed from the set before step 4.

Finally, notice that the seedlets in the set after Steps 2 and 3 have the exact same form: the scan performed during Step 3 does not fill or add *any* positions at all. ■

A fill strategy that includes scans may reach its maximum size before the final shift is completed. We now examine the likelihood of removing a seedlet from the set based on a scan.

Previously, we ignored any shift which produced a query with more than one free position outside the original probe range. If we hold to that rule, then scans will be queries with only a single free position: by definition, they have no free positions within the original range of the probe. Such queries produce at most at most 4 matches in the spectrum, so the likelihood of producing 0 matches is  $(1 - \alpha)^4$ . When  $\alpha = 0.1$ , performing a scan is likely to eliminate  $(1 - \alpha)^4 \approx 66\%$  of all seeds, and reduce the seed set to about 1/3 of its previous size.

Since a scan cannot possibly increase the size of the seed set, it is advantageous to perform any scan which reduces the size of the seed set. For instance, a scan which contains 2 free positions has a probability of  $(1 - \alpha)^{16}$  of producing no matches in the spectrum, thereby eliminating a seedlet. When  $\alpha = 0.1$ , this corresponds to a 19% chance of eliminating a seedlet from the set. Can scans with more than two free positions also be used?

In general, let

$$c_f = 1 - (1 - \alpha)^{4^f}$$

be the probability that there is at least one match in the spectrum to an  $f$ -character scan. And let

$$q_f = \alpha \cdot 4^f$$

be an upper-bound approximation to the expected number of responses to a spectrum query with  $f$  free positions. Values of  $c_f$  and  $q_f$  for  $\alpha = 0.1$  and  $\alpha = 0.2$  are given in the following table, along with the expected number of matches to the corresponding  $f$ -character query string:

$\alpha$	$c_1$	$c_2$	$c_3$	$q_1$	$q_2$	$q_3$
0.1	0.344	0.815	0.999	0.4	1.6	6.4
0.2	0.590	0.972	1.000	0.8	3.2	12.8



Determining whether or not the work required to perform a scan will reduce the overall work performed in constructing the seed is somewhat complex. We need to estimate the expected number of spectrum checks that will result from keeping each of the seedlets currently in the set, and compare that number with the number of spectrum checks required to perform the scan.

Scans with three or more free positions are extremely unlikely to eliminate any seeds from the set, and they are exponentially more expensive to perform (requiring  $4^f$  spectrum checks for a scan  $c_f$ ). Here is work-based justification for scans, limiting our attention to scans with one or two free positions:

- **Single-character scans**

Single-character scans are equivalent to a reduction step, so they may be performed at any point in the seed building sequence. The earlier they are performed, the more benefit they produce, since at any step prior to the complete filling of the seed, they allow the elimination of not just a single potential seed, but all of the children that would have resulted from allowing that seed to remain in the set.

- **Double-character scans**

The work-based justification of double-character scans is slightly more complex than single-character scans. We want to perform double-character scans (scans which include two free positions outside the range of the initial probe) when the cost of performing the scan is lower than the cost of individually eliminating all of the descendants of each of the probes expected to be eliminated by performing the scan.

The cost of a double-character scan is 16 spectrum checks per seedlet: one for each of the four possible matches to the query string. The additional cost incurred by *not* performing the scan is equal to the probability of eliminating a seed, multiplied by the total number of checks required to grow and ultimately eliminate all descendants of the tree spawned from the seedlet which would have been eliminated.

Given a shifting strategy  $S$ , we can calculate exactly the expected size of the tree spawned from a seedlet at any point in the construction process. Let  $f_i$  be

the number of free positions in each query  $\phi_i$ , and let  $|set_i|$  be the size of the set after performing the same query. If  $j$  shifts have already been performed, then the total number of descendants to each current seedlet in the growth stage is  $D_g$ , which can be calculated as:

$$D_g = \sum_{i=j+1}^{\phi} \alpha 4^{f_i+1} |set_{i-1}|$$

Now we must consider all of the descendants to the current seedlet in the reduction stage. Given that there are  $\phi$  steps in the growth stage, and we have completed  $j$  steps which have filled  $F = \sum_{i=0}^j f_i$  of the initial  $\lambda - \kappa$  unspecified positions, we expect there to be approximately  $|set_{\phi}| = \alpha^{\phi-j} \cdot 4^{(\phi-j)+\lambda-\kappa-F}$  seedlets descended from each current seedlet at the end of the growth stage.

The number of descendants of these seedlets in the reduction stage is  $D_r$ , and can be estimated as follows:

$$D_r = \sum_{i=1}^s (\alpha 4)^i \cdot |set_{\phi}|$$

Recall that the probability of eliminating a seed from the set by performing a double-character scan is  $1 - c_2 = (1 - \alpha)^{16}$ . Four checks must be performed for each seedlet at every level in the tree spawned from the current one. We conclude that double-character scans should be performed whenever

$$16 < 4 \cdot (1 - c_2) \cdot (D_g + D_r)$$

In practice, these calculations were simplified further, and two-character scans were used as long as there were 2 or more free characters left. This results in slightly more work being performed by the seed construction algorithm, but limits the ultimate size of the seed set: the ‘optimal’ strategy in terms of work performed requires more memory than strategies with unjustified double-character scans. Slightly looser requirements for performing double-character scans assign some value to the memory taken by the algorithm.

Now, with  $u$  single-character scans and  $v$  double-character scans, we can express the expected size of the set upon completion of the growth stage as

$$|set_\phi| = c_1^u c_2^v \alpha^\phi 4^{\phi+\lambda-\kappa}$$

The next step is to find an optimal strategy incorporating scans.

### 8.2.3 Searching for a Better Fill Strategy

Extensive experiments indicate that it is not possible to design a simple greedy algorithm to find the optimal fill strategy for a particular probing pattern. Often, a shift which produces a query with more free positions than another available shift at a certain point in the growth stage will allow subsequent shifts which produce an ultimately smaller seed set. To find an optimal (or near-optimal) shifting strategy, a randomized algorithm was designed.

The algorithm accepts one parameter: a probing pattern  $P$  with length  $\lambda$ . The result of the algorithm is a series of offsets, all in the range  $[-(\lambda - 1), \lambda - 1]$ , corresponding to the shifts and queries which should be performed to fill in all unspecified positions in a randomly selected starting probe.

Each shift is selected by the randomized algorithm by searching over all admissible shifts (shifts which introduce only one new free position outside of the original probe range), with the following rules:

1. If one or more single-character scans are available, perform all of them before any other shifts.
2. If one or more double-character scans are available, and there are still 2 or more unfilled positions in the seedlets, perform all of the double-character scans before any other shifts.
3. Out of all admissible shifts label them all according to the number ( $f_i$ ) of free positions they produce. If the optimal shift has  $f'$  free positions, select a different shift  $i$  with probability  $p = 2^{-(f_i - f')}$ .

By iterating over this shift-selection process until all positions in the initial probe are filled, a single fill strategy is produced. The algorithm is repeated 10000 times,

and the strategy which produces the smallest maximum set size for a particular value of  $\alpha$  is selected. Note that strategies may differ based on  $\alpha$ , because of the varying effects that scans can have on set size.

The ultimate result of the above algorithm is a seed strategy for (4,4)-probes which reduces the maximum expected size of the seed set to  $c_1^3 c_2^2 \alpha^7 4^{19}$ , when  $\alpha = 0.2$ . A slightly different seed strategy produces better results when  $\alpha = 0.1$ ; this discrepancy is due to the scan steps introducing multipliers which are not linear functions of  $\alpha$ .

The span of the initial probe is shown using a bold 'N' character to delimit the ends of the probe. Filling steps are shown on the lines labelled with 'F', followed immediately by another line which shows the characters which have been 'filled' or specified so far. Scan steps (which specify no additional characters) are shown on lines beginning with 'S'. At the end of each line corresponding to a scan or fill step, three numbers are printed. The first number gives the position of the shift relative to the initial probe position. The second gives the number of free positions in the query. Furthermore, the actual free positions in a query are denoted with the '?' character. And the third number gives the expected size of the seed set after that step has completed, using  $\alpha = 0.2$ . For example, a fill step which shifts the probing pattern 2 characters to the left, and results in a query with 4 free positions, would be printed as:

S/F	Query String	Offset	Free Pos	Set Size
F	? . . . ? . . . ? . . . ? . . . N N N N	-2	4	2620

The complete fill strategy for (4,4)-probes is:

S/F	Query String	Offset	Free Pos	Set Size
	.....N...N...N...N...NNNN.			
F	.....?...N...N...N...N???N	-4	4	41
=	.....N...N...N...N...NNNNNNNN.			
F	.....?...?...?...?...NNNN	-2	4	2620
=	.....N.N.N.N.N.N.N.N.NNNNNNNNN.			
S	.....N...N...N...N...NN??	2	2	2546
F	.....?...N...N...N...N?NN	-6	2	8147
=	.....N.N.N.N.N.N.N.N.NNNNNNNNNNN.			
F	.....?...N...N...N...N?NN	-8	2	26068
=	.....N.N.N.N.N.N.N.N.N.NNNNNNNNNNNNN.			
F	.....?...?...?...N...NNNN	-3	3	333620
=	.....N.N.NNN.NNN.NNN.NNNNNNNNNNNNN.			
S	.....?...N...N...N...NNNN	-7	1	196944
S	.....N...N...N...N...NNN?	1	1	116261
F	.....?...N...N...N...NNN?	-12	2	371970
=	.....N...N.N.NNN.NNN.NNNNNNNNNNNNNNNNNNN.			
S	.....?...?...N...N...NNNN	-11	2	361493
S	.....?...N...N...N...NNNN	-10	1	213399
F	.....?...N...N...N...NNN?	-16	2	682754
=	.....N...N...N.N.NNN.NNNNNNNNNNNNNNNNNNNNN.			

The size of the seed set for  $\alpha = 0.1$  and  $\alpha = 0.2$  for randomized fill methods (which will be called *fill\**), as well as the two previous methods (called *shift-1* and *shift-4*, according to the number of positions shifted in each step) is shown in the following table. Scan-steps are omitted, so that the set size is shown after each fill-step. Also note that the strategy for  $\alpha = 0.1$  differs from that for  $\alpha = 0.2$ . The maximum set size reached for each shift strategy is shaded, since it does not always correspond to the final size of the seed set.

Step	$\alpha = 0.1$			$\alpha = 0.2$		
	<i>fill*</i>	<i>shift-1</i>	<i>shift-4</i>	<i>fill*</i>	<i>shift-1</i>	<i>shift-4</i>
1	25	102	25	51	205	51
2	655	$1.049 \times 10^4$	655	2621	$4.194 \times 10^4$	2621
3	1440	$1.074 \times 10^6$	$1.678 \times 10^4$	8151	$8.590 \times 10^6$	$1.342 \times 10^5$
4	3169		$4.295 \times 10^5$	$2.608 \times 10^4$		$6.872 \times 10^6$
5	1743			$3.338 \times 10^5$		
6	2788			$3.618 \times 10^5$		
7				$6.83 \times 10^5$		

There are several things which should be noted in the above table. First, for  $\alpha = 0.1$ , the size of the seed set for the *fill\** strategy peaks at step 4. This is due to the judicious use of scan-steps during the growth stage. Second, the maximum size achieved for  $\alpha = 0.1$  by the *fill\** strategy is about 3% of the size reached by the initial naive fill strategy (*shift-1*). While the advantage of the *fill\** strategy is diminished when  $\alpha = 0.2$ , the maximum seed set size is still only 8% of the maximum size reached by the naive *shift-1* strategy. This reduction makes it possible to consider constructing seeds for sequences which approach (or even exceed) the limit achievable by our algorithm ( $m \approx 15000$ ).

### 8.3 Probe Structure

We have not been able to make any specific predictions regarding the maximum seed set size which will be generated by a good shift strategy for a given probing pattern. However, we were able to make some general observation regarding the effect of probe structure on seed construction.

It is possible to reduce the expected size of the seed set by choosing probing patterns different from the standard  $(s, r)$ -probes. Of course, if the length of the pattern is increased or decreased, a corresponding increase or decrease in the set size is to be expected: every additional unspecified position in the probe increases the expected set size by a factor of at least  $4\alpha$ . But even considering only patterns with  $\lambda = 20$  and  $\kappa = 8$ , striking results can be observed.

For instance, a brief experimental search results in the discovery of a pattern

$N \dots N \dots NNNNNN$  (essentially a  $(6, 2)$ -probe with an additional universal base inserted in each of the gaps) which yields a fill strategy with a maximum expected size of  $c_1^5 \alpha^8 \cdot 4^{20}$ . This results in a final (and maximum) set size of only  $2.01 \times 10^5$  seedlets when  $\alpha = 0.2$ ; less than  $1/3$  the size of the set produced by a  $(4, 4)$ -pattern. And the pattern  $N \dots NNNNNN$  produces a maximum set size of only  $4.1 \times 10^4$  seedlets when  $\alpha = 0.2$ . Furthermore, the fill strategy which produces such a small seed set is optimal, in that it uses 12 shifts to fill the 12 unspecified positions in the probe. There is no fill strategy for *any* probing pattern with  $\kappa = 8$  and  $\lambda = 20$  which could use more shifts.

These non-standard probes are obviously better suited to seed construction for longer sequences (with correspondingly higher  $\alpha$ ) than the standard  $(s, r)$ -probes. The drawback to patterns which are suited to efficient seed construction is that they have poor autocorrelation functions and are ill-suited to actual sequence reconstruction.

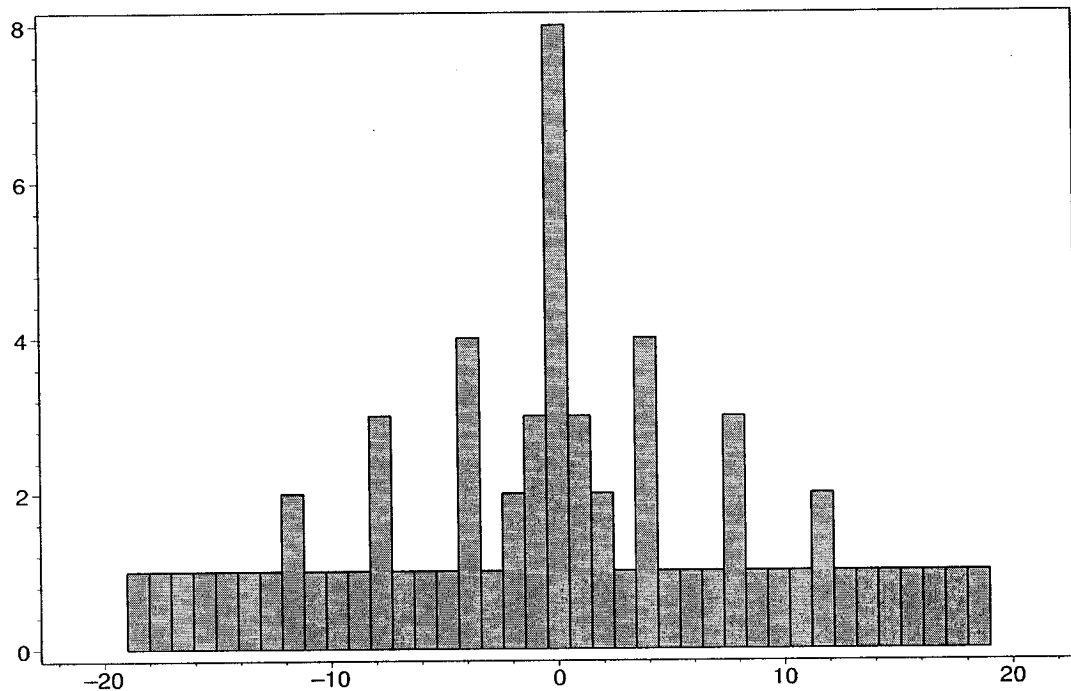


Figure 8.1: Autocorrelation function for  $(4, 4)$ -probing pattern.

At the other end of the spectrum, the ideal probing pattern for sequence reconstruction is a Golomb ruler: a pattern which has auto-correlation of at most 1 for any off-peak shift. The shortest (optimal) Golomb ruler with 8 marks (specified bases)

has length 35; this is significantly longer than an  $(s, r)$  probing pattern, but serves as a useful example of worst-case seed construction<sup>1</sup>. Since the first shift performed in the growth stage of seed construction intersects with at most one specified base from the initial probe, the expected size of the seed set after only *one* shift is  $\alpha 4^7$  seedlets. It is impossible to improve on this result for the initial shift because there is no offset at which the Golomb-pattern has autocorrelation of more than 1 position. The final (and maximum) size of the seed set constructed using a the best known fill strategy for a Golomb-ruler probing pattern is  $c_1 c_2^2 \alpha^{10} 4^{37}$  after 10 shifts, effectively making it impossible to construct a seed at all with such a pattern.

### 8.3.1 Autocorrelation and Seed Strategy

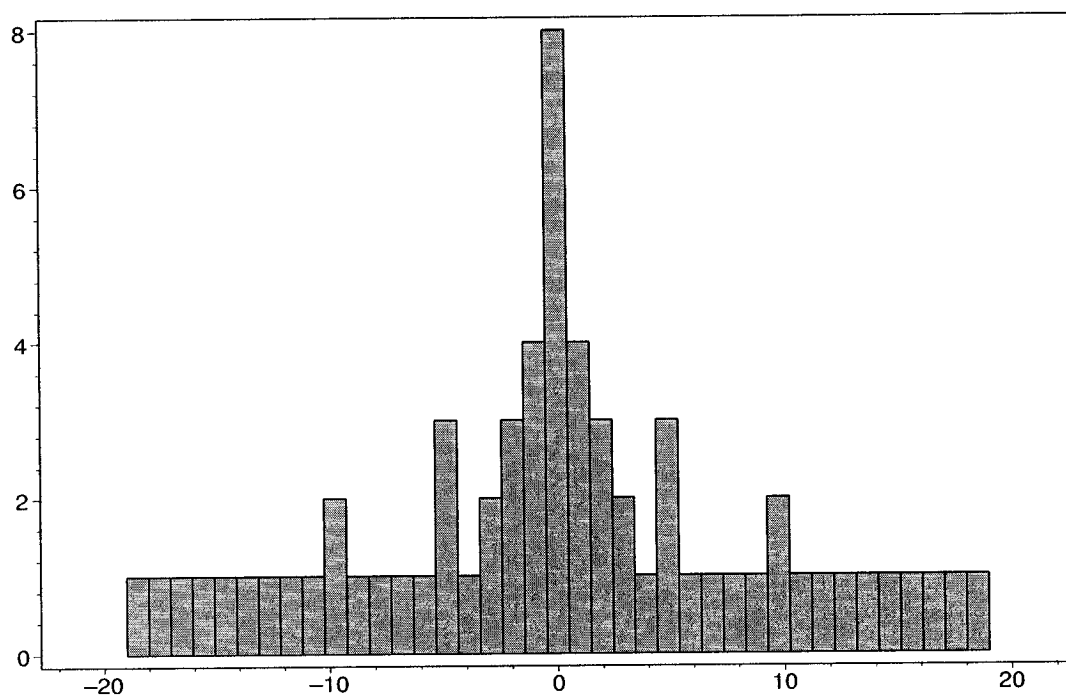


Figure 8.2: Autocorrelation function for (5,3)-probing pattern.

By comparing (4,4)-probes with the patterns  $N \dots N \dots NNNNN$  and

<sup>1</sup>The 8-mark ruler used here was  $NN..N....N....N.....N.....N.N$ , from the Optimal Golomb Ruler page on the internet at <http://www.research.ibm.com/people/s/shearer/gropt.html>



N . . . . . NNNNNNN, we note that in general, the size of the seed set decreases as the gaps in the probe grow fewer and larger. In particular, the best seed strategy is produced for a probe with only a single gap, of length 12. Such probes are not suited to sequence reconstruction, due their poor autocorrelation functions.

However, (4,4)-probes and (5,3)-probes are nearly identical in terms of sequencing performance, and (5,3)-probes are significantly better for seed construction. The best-known fill strategy for (5,3)-probes produces a final seed set size of  $4.27 \times 10^5$  when  $\alpha = 0.2$ , corresponding to a seed construction strategy yielding  $c_1^4 \alpha^7 4^{19}$ .

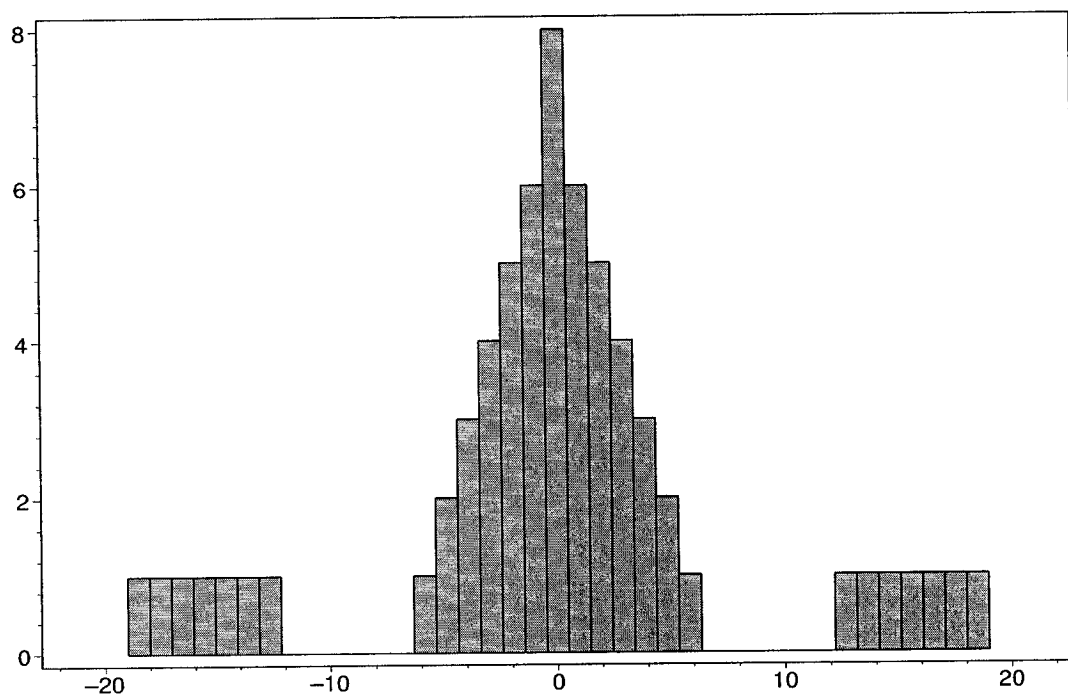


Figure 8.3: Autocorrelation function for N . . . . . NNNNNNN probing pattern.

In general, there appears to be a trend in the autocorrelation function of these probes, which corresponds well with the observed maximum seed set sizes. As the central peak in the autocorrelation function increases in prominence, the size of the seed set decreases. The histograms for the autocorrelation of the (4,4)– and (5,3)-probes and the N . . . . . NNNNNNN pattern are included as figures for reference. Also, the following table shows the maximum seed set size achieved by the best-known seed strategy for a variety of probes. For all probes,  $\kappa = 8$ , and for all but the

OGR-probe, the overall probe length  $\lambda = 20$ .

Probing Pattern	$ set_{max} $ of Best Fill Strategy
NNNN...N...N...N...N (4,4)	682750
NNNNN...N...N...N...N (5,3)	414700
NNNNNN.....N.....N	201930
NNNNNNNN.....N	41140
N..N..N..N..N..N..N..N	103900
NNNN.....NNNN	133190
OGR-8 probe	$31.45 \times 10^6$

The inference we can draw from these observations is fairly simple. We know that probing patterns which have very low autocorrelation functions perform better during sequencing, since the correction factor required to account for the possibility of overlapping fooling probes is lower. However, the difference in sequence performance between  $(s, r)$ -probes and an OGR probing pattern is barely significant: an increase of about 2-3% in terms of sequence length. On the other hand, it is effectively impossible to construct a seed from the spectrum of an OGR probe, due to the explosive growth of the seed set. For seed construction,  $(s, r)$ -probes have a remarkable advantage, due largely to their periodic nature. In fact, the ease with which seeds can be constructed for  $(s, r)$ -probes gives them an overwhelming advantage over longer probes with ‘better’ autocorrelation functions.

## Chapter 9

# Conclusion & Future Work

Sequencing-by-Hybridization has been studied for almost 15 years. The initial attempts to develop SBH as feasible sequencing method [BS91, L+88, D89, P89, P91] yielded results which fell short of providing any improvement over traditional sequencing techniques. The information theoretic bound on the length of fragments which could be reconstructed using the technique on the maximum fragment length which could be sequenced with a fixed confidence level  $\epsilon$  using probes with  $\kappa$  natural bases was  $\frac{4\kappa}{2}$ .

The performance of the initial algorithms was shown to be limited to the square root of the information-theoretic bound [DFS94, S96]. However, more recent developments have proven that the method has a great deal of promise. By making use of universal bases, Preparata, Frieze & Upfal [PFU99] presented an algorithm which achieved performance that asymptotically approached the theoretic bound.

In another paper in 2000, Preparata & Upfal [PU00] presented another, more advanced SBH algorithm which achieved performance within a constant factor of about 0.4 of the information-theoretic bound: this is the basic gapped-SBH algorithm which is described in Chapter 2. Their algorithm and analysis of its performance are used as the foundation of the current work.

## 9.1 Contributions

The research described in this dissertation has taken several forms, most notably enhancements to the existing SBH algorithm and further analysis of its behaviour. Although these improvements only affect the constants in an SBH method previously shown to be asymptotically optimal, the potential applications of SBH are such that a constant factor improvement can be very significant. Other than enhancements to the sequencing algorithm, we also developed a method of seed construction that allows sequencing to commence even in the absence of a known, biochemically affixed primer string. Finally, we measured the performance of the SBH algorithm on natural DNA sources. From this, we developed a preliminary model of DNA that offers a degree of predictive power, and allows a variety of natural DNA to be represented by a model of only a few parameters. These contributions can be divided into four basic areas, which are each described in turn.

- **Algorithmic Enhancements**

We have extended previous algorithms to increase the performance of gapped-probe SBH by over 50%, to nearly  $2/3$  of the information-theoretic bound, when using (4,4) or (5,3)-probing patterns. All of the performance increases discussed below assume the use of these specific probes. There were three specific enhancements made to the SBH algorithm from [PU00].

The first was the *polling provision* described in Chapter 5, which makes an informed decision between two alternate extensions at a failure point. The polling provision increases the predicted and observed maximum fragment length by about 25% over the standard gapped-SBH algorithm.

Second, the *tandem-spectrum* sequencing algorithm allows a 30% increase in the length of feasible target sequences, over and above the required increase in chip cost, and independent of the polling provision.

When the polling provision and tandem-spectrum sequencing algorithm are used together, they allow target sequences of length  $m > 42000$  bases (with doubled chip size) to be successfully reconstructed with confidence  $\epsilon = 0.9$ . Figure 9.1 shows a comparison of all four techniques.

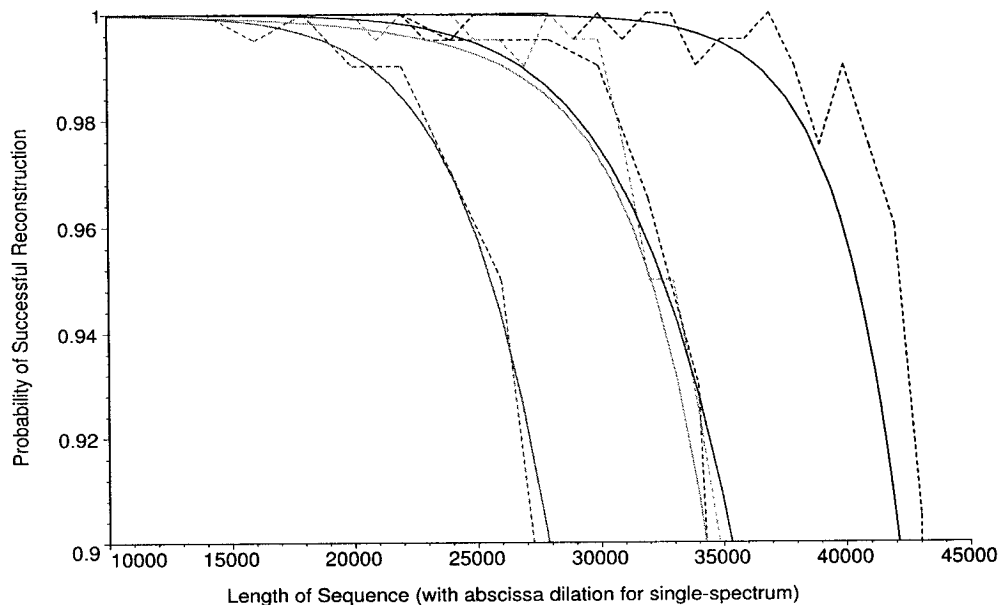


Figure 9.1: A comparison of single-spectrum and tandem-spectrum sequencing with and without the polling provisions, using random DNA and (4,4)-probes. Solid lines represent the predicted values for all four conditions, and dotted lines the observed values. From left to right, we have single-spectrum without voting; tandem-spectrum without voting; single-spectrum with voting; and tandem-spectrum with voting.

Finally, the *repeat-recovery* technique, while not really applicable to random DNA, allows a minor increase in the length of natural DNA sequences which can be reconstructed, and indicates that targeted algorithms may further improve sequencing performance on natural DNA.

### • Combinatorial Analysis

The analysis of the size of the path trees created by EXTEND's branching mode gives greater insight into the behaviour of the SBH algorithm than has previously been available. We developed a detailed model of the growth of the path trees that naturally allows a very accurate prediction of the amount of work—in terms of spectrum CHECKS—performed by the sequencing algorithm.

Furthermore, we developed an analytical model of Mode 2 failures, which are the dominant failure mode for tandem-spectrum SBH. Prior to our work, only Mode 1 failures had been well-understood.

- **Seed Construction**

The seed construction algorithm described in Chapter 8 allows sequencing to be started at any location in the sequence, negating the need for primers to be affixed to the end(s) of the target sequence. This gives gapped-SBH a nice degree of algorithmic completeness: other than the hybridization process, no other biochemical steps must be completed before the computational sequencing algorithm begins. Furthermore, the ability to construct a seed from the spectrum of a target may allow multiple fragments of the target sequence to be produced when the algorithm fails, rather than a single shortened piece.

- **Measuring and Modeling Natural DNA**

In order to facilitate comparison between different sequencing techniques, their performance has traditionally been evaluated only in terms of their expected performance on random DNA. We continued to uphold this convention throughout the current dissertation, but we have also measured our algorithms' performance on natural DNA sequences. This additional information offers an initial view of how well SBH may perform in real-world situations. Furthermore, by examining the behaviour of the SBH on natural DNA, we were able to begin developing algorithmic methods to compensate for the structural problems inherent in such sequences.

The *repeat-recovery* technique was the first of such methods, and although it did not offer the performance gains that were hoped for, it did successfully overcome some of the short-period failures that occur when sequencing natural DNA. However, there is a lot of room to improve the method in the future.

## 9.2 Future Work

There are many avenues open for further research on this topic. We begin by mentioning minor extensions and improvements to the methods developed in this thesis, and move on to more elaborate projects.

- **Improved Repeat-recovery Algorithm**

The repeat-recovery algorithm, in its current state, allows recovery from only the most basic of repeat failures. It should not be too difficult to extend the current algorithm to handle more complex cases, with multiple potential continuations and more than one repeated segment.

- **Better Model of Natural DNA**

A simple model of natural DNA presented in Appendix C is sufficient for our current purposes, but it could also be used as a starting point for building a more rigorous model. If the repeated subsequences present in DNA are due to one (or a few) biological processes, then a model that accurately reflected the behaviour of these processes could be useful not only to computational biologists, but researchers in related fields as well.

- **Eulerian Path Sequencing**

When the polling algorithm selects an extension at a the point of Mode 2 failure, it then immediately ‘forgets’ the paths that it chose between, along with the identity of the path that it chose. This information could be stored, instead of discarded. Then, if the same ‘choice point’ was encountered again, the algorithm would know that it should not select the same extension as it did in the previous encounter. This strategy, along with a reasonably accurate estimate of the total length of the sequence, would allow an Eulerian path traversal over all choice points encountered during sequencing. Such a traversal could be expected to offer (at least) a small performance improvement, and might also help to select between multiple alternate continuations from periodic segments.

- **Sequencing with Noisy Data**

Attempts [DH00, H+02] have been made to extend SBH to function in the presence of noisy data. While they have extended a less advanced version of the gapped-SBH algorithm (introduced in [PFU99]), the results have been encouraging. Halperin et. al. [H+02] demonstrated a sequencing method using a collection of randomly generated probing patterns which achieved performance only a factor of  $\log m$  worse than for the noise-free case. And Leong et. al. [LPSW02] used another approach to control for noise in SBH. However, most

research into SBH with noisy data has used the initial [PFU99] version of the gapped-SBH algorithm. More work needs to be done to extend the more advanced versions of the SBH algorithm to function in the presence of noise.



# Appendix A

## GenBank Accession Numbers

The following GenBank files were used in this thesis:

Organism name /	Chromosome	Accession Number	Date
<i>a. thaliana</i>	chr. I	NC_003070	August 13, 2001
<i>a. thaliana</i>	chr. III	NC_003074	August 13, 2001
<i>h. sapiens</i>	chr. 3	NT_015150	August 1, 2002
<i>h. sapiens</i>	chr. 11	NT_017854	August 1, 2002
<i>p. falciparum</i>	chr. 3	MAL3	April 15, 1999
<i>s. cerevisiae</i>	chr. IV	NC_001136	September 22, 2002
<i>e. coli</i>	N/A	NC_000913	October 26, 2001
<i>h. influenzae</i>	N/A	NC_000907	February 14, 2002
<i>s. typhimurium</i>	N/A	NC_003197	November 7, 2001
<i>s. pneumoniae</i>	N/A	NC_003098	October 3, 2001
<i>m. acetivorans</i>	N/A	NC_003552	April 9, 2002
<i>s. solfataricus</i>	N/A	NC_002754	October 3, 2001

# Appendix B

## Obtaining the Simulator Source

The data generated in this dissertation was generated using an SBH simulator program written in C++ on Solaris and Linux (RedHat and Debian). It consists of about 15 thousand lines of source code (including blank lines and comments), and several additional helper programs which are used to extract coding sequences from GenBank files; calculate Markov Models from data; generate data from Markov Models; and generate random probing patterns (or sets of patterns).

While it was designed with a single purpose—to supply experimental verification of the analytical predictions made in this thesis—it may be useful to any other researchers who are working on SBH algorithms. (On the other hand, it may not.)

### B.1 Obtaining the Source Code

A tarball of the simulator source code, along with some related scripts and other utility programs may be obtained at <http://www.cs.brown.edu/~sah/seq.tar.gz>.

The simulator has been compiled, at various times, using different versions of the Sun and gnu compilers, but development for the last 8 months was conducted entirely using versions 2.95 and 3.1 of the gnu C++ compiler on Debian Linux 2.2 and Redhat Linux 7.2, respectively. Minor changes to the source may be required to make the code conform more precisely to the C++ standard (if such a standard can even be said to exist).

# Appendix C

## Model of Duplicated Subsequences in DNA

The performance of SBH degrades significantly when the target sequences are drawn from natural DNA. Mode 1 failures are more common, and Mode 2 failures occur with dramatically greater frequency. There are also repeat-failures which do not occur at all in random DNA, but appear with some frequency in natural data. To try and understand what it is about natural DNA that produces more and different types of failures than random data, we took preliminary steps to build a better model for natural DNA.

### C.1 Natural DNA and Models

A great deal of work (see for example, [F+94, LLY00]) has gone into calculating the information content, or entropy of DNA. While the results of these studies indicate that DNA appears to be nearly random, most of the techniques for estimating information content assume that the data source being observed is stationary. This assumption is inherently flawed. Introns are qualitatively different from exons: one type of DNA is used to code for proteins, and the other is not. Coding regions differ from intergenic regions even more drastically. Any measure of information or entropy which assumes a memoryless model is bound to misrepresent the data.

Furthermore, exons *cannot* be random sequences: the proteins into which they are

translated have specific functions which must be encoded in their sequences. There are also higher-order patterns in DNA caused by natural processes of infection by viruses, recombination of DNA molecules, ‘jumping genes’ called transposons, and the non-random effect of natural selection. A detailed examination of the primary structure of DNA—the sequence itself—can provide valuable help in refining our algorithms, and may be of some use to biologists and other bioinformaticians as well.

## C.2 Suffix Trees

The most important aspect of natural DNA from our perspective is the frequency and length of duplicate strings in a fragment. Duplicates of length  $(\lambda - 1)$  or greater automatically cause the SBH algorithm to fail, and even shorter repeats significantly enhance the likelihood of a Mode 2 failure. The suffix tree is a natural way of studying the occurrence of repeats at a high level.

Suffix trees are data structures which contain all possible suffixes of a string. Nodes in the tree have a degree of at most the size of the alphabet (in the case of DNA, this limit is 4), and every unique suffix corresponds to a leaf in the tree. Thus, the string **CAGCAGT**, which is 7 characters long, would have 7 leaf-nodes—one for every possible suffix—including the complete 7-character string. Every unique directed path from the root of the tree to a leaf node reproduces a unique suffix of the tree. Early methods for constructing suffix trees were slow for strings of  $10^5$  characters or more, but Ukkonen’s algorithm [U95] constructs a suffix tree in amortized linear ( $O(n)$ ) time.

The usefulness of the suffix tree in finding duplicate strings lies in the internal nodes of the tree, not the leaves. If a node has at least two children, then the string from the root leading to that node must occur at least twice in the tree. In fact, it is possible to count the number of occurrences of any string in the sequence by finding its corresponding node in the tree and counting the number of leaves in the subtree rooted at that node.

By traversing the tree and counting the number of internal nodes at depth  $n$ , we can count how many different strings of length  $n$  are duplicated at least once in a sequence, since every internal node in the tree corresponds to a string which occurs

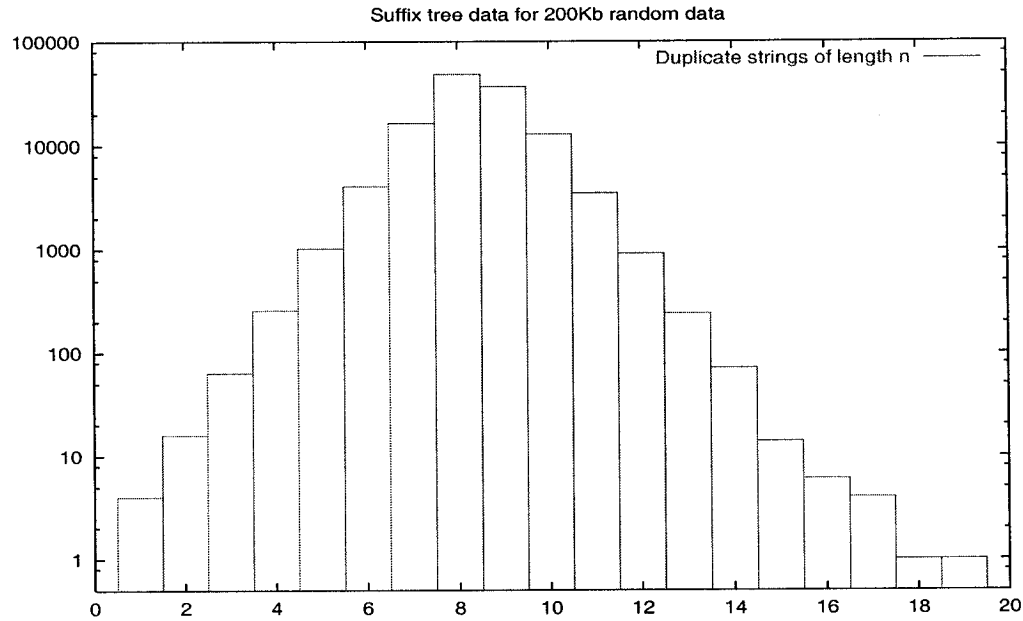


Figure C.1: The suffix tree for a 200Kb random DNA sequence shows a full tree up to length 7, followed by a probabilistic exponential falloff for longer strings.

at least twice in the sequence. In random sequences, the tree appears to be *full* at short lengths: for  $m = 2 \times 10^6$ , all possible sequences of length  $n \leq 7$  are present in the tree, indicating that they occur at least twice in the sequence. At lengths  $n > 7$ , duplicate strings of length  $n$  are present with probability decreasing with length. Recall from Chapter 3 that the probability of finding a particular  $n$ -character string in a random  $m$ -character sequence is  $\alpha_{(m,n)}$ . Thus, the probability of finding at least *one* duplicate string of length  $n$  in an  $m$ -character sequence is approximately  $m \cdot \alpha_{(m,n)}$ . The likelihood of finding any duplicate strings of 20 characters or greater in strings of less than 20Kb is very small. In fact, even with  $m = 200\,000$ , there is only a 3.6% chance of seeing a duplicated string of length 20.

A plot illustrating the distribution of the length of duplicated strings observed in a typical 200Kb random sequence of DNA is shown in Figure C.1.

Without a good model of natural DNA, there is no way to predict the incidence of long duplicated strings in natural organisms. However, as Figure C.2 shows, the DNA natural organisms shows a much higher incidence of long duplicated strings than random sequences. This is important, because every duplicate string of length

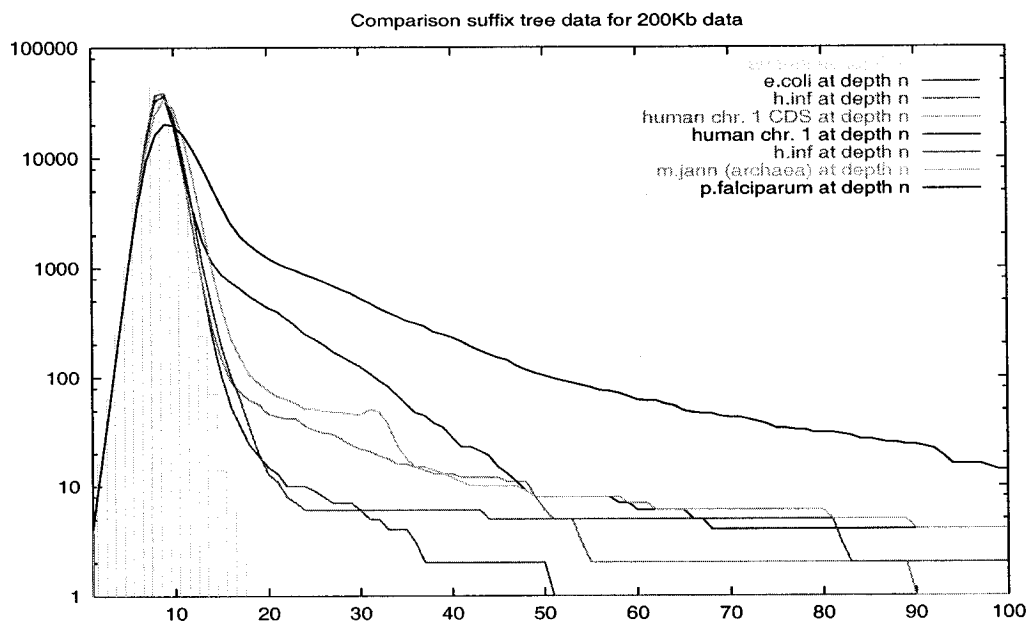


Figure C.2: Suffix trees for natural DNA demonstrate an increased incidence of long duplicated strings.

$(\lambda - 1)$  or greater causes the basic algorithm to fail. A model can help us extend the basic sequencing method to recover from some of these failures. But perhaps more importantly, we also would like to be able to better predict our method's chance of success.

### C.3 Markov Models and Entropy

It is obvious that there are more repeated strings in natural DNA than would result from a simple uniform random generating model. Somewhat misleadingly, the incidence of individual bases in natural DNA sequences appears to be fairly uniform, although there is some skew to the occurrence probabilities. If some nucleotides are more common than others, then sequences which consist of the more common nucleotides will occur (and thus be duplicated) with greater frequency than others. The following table shows the occurrence probabilities observed for five different natural DNA sequences:

Data source	A	C	G	T
human chr. 1	0.3011	0.2001	0.1998	0.299
human chr. 1 CDS	0.2394	0.2593	0.2589	0.2424
<i>e. coli</i>	0.2463	0.2543	0.2535	0.246
<i>h. inf</i>	0.3102	0.1916	0.1899	0.3083
<i>p. falciparum</i>	0.399	0.09902	0.09975	0.4022

While we know that a simple memoryless model does not suffice as a model of natural DNA, it is reasonable to ask how well a stationary Markov Model with a memory of the previous  $\xi$  states would reflect natural data. First it is necessary to determine what value of  $\xi$  would most accurately represent the DNA being modeled. To do this, nine Markov Models (with  $\xi = 0, 1, \dots, 8$ ) were trained on 2.5Mb of coding sequences from human chromosome I and *e. coli*. Once the models were trained, the estimated entropy  $e = -\sum p \log(p)$  per state was determined.

The following table shows that the entropy value appears to asymptotically approach about 1.9 bits per base until the 7th-order model, when it drops precipitously.

MM order	<i>e. coli</i>	human chr. 1 CDS
0	1.99983	1.99915
1	1.98142	1.95409
2	1.96005	1.94837
3	1.9455	1.93929
4	1.93736	1.94563
5	1.93049	1.9311
6	1.9228	1.91924
7	1.88741	1.88585
8	1.75188	1.64665

Upon examination, it was found that about 10% of the states in the 7th-order Markov Models had been visited 3 or fewer times, and nearly 15% of the states in the 8th-order models had been visited 3 or fewer times. With so few visits to so many states, the transition probabilities out of these states could not possibly accurately represent the underlying DNA, and the low entropy values for these low-visit states was skewing the overall average entropy value for the model.

We created Markov Models for 8 different natural sources of DNA, setting the value of  $\xi$  to 5 or 6, depending on the number of training states available. If fewer than 1Mb were available,  $\xi$  was set to 5; otherwise it was set to 6. Each Markov Model was then used to generate 2Mb sequences, and the suffix-trees were created for both the natural DNA sources and the MM-generated data.

From the suffix trees, we counted the incidence of duplicate strings of up to 25 characters, and compared these suffix trees to those for natural DNA. In general, any difference between the incidence of repeated strings in the natural DNA and Markovian-generated DNA will have to be dealt with separately from the Markov Model. Figure C.3 shows the difference between suffix tree sizes for natural *e. coli* and data produced by a 6th-order Markov Model of the same.

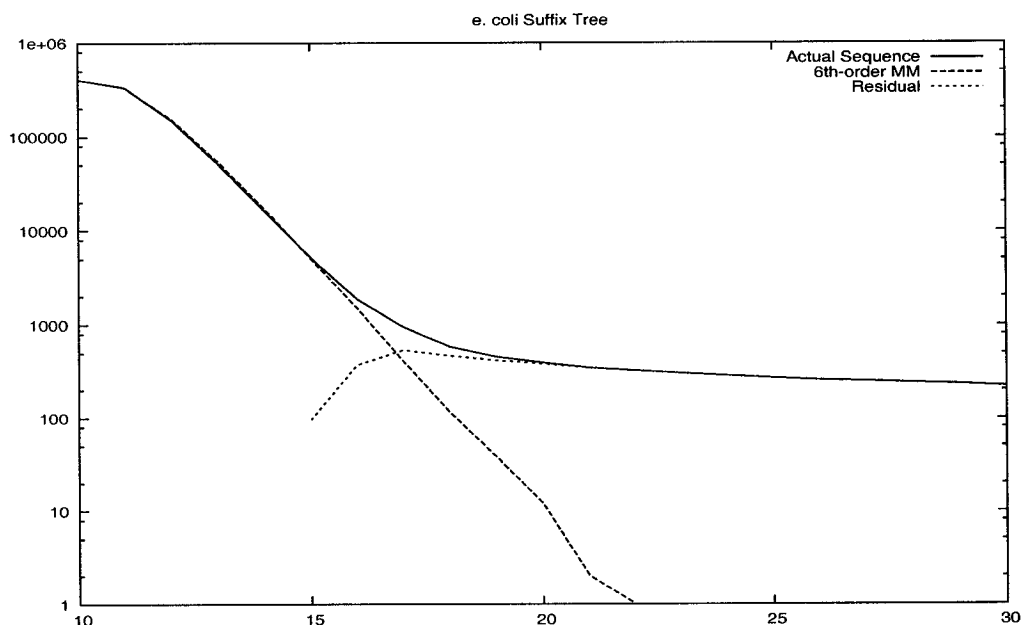


Figure C.3: Duplicate strings in *e. coli* compared to those predicted by a 6th-order Markov Model. The Markov Model accurately produces the correct number of duplicate strings up to about length 16.

In general, the Markov Models are not particularly accurate models. They can accurately reproduce the occurrence of duplicate strings of DNA for lengths  $n \leq 16$ , although it begins to underpredict duplicate strings, at about length 13. The Markov Model almost never produces any repeats longer than 22 characters at all.



Figure C.3 shows the results of such an experiment for a 6th-order Markov Model of *e. coli*. The graph shows, for lengths 5 to 22, the number of duplicate strings over-predicted (negative number) or under-predicted (positive number) by the Markov Model. There is an atypical spike around length 11, where the model actually underpredicts the number of duplicate strings of that length. This is unusual, and ultimately not very relevant to the construction of a more complex model, since strings of that length do not typically have much influence on sequencing failures.

These simple experiments show that although a simple Markovian model is obviously not sufficient to accurately model the repeat-structure of natural DNA, it is significantly better than a uniform random data. We move on to develop a better model of natural DNA, using a Markov Model as a base.

## C.4 Building a Model

Any good model of DNA should be able to represent a wide variety of natural DNA sources, with only parameter changes—the structure of the model should be left unmodified. For the purposes of measuring or estimating the performance of DNA, an effective model must generate sequences which exhibit the same repeat structure as natural DNA, as it is the repeat structure which has the greatest impact on sequencing performance. Starting with a sequence generated by a 5th- or 6th-order Markov Model, it is possible to reproduce the repeat structure of natural DNA by copying substrings from one location to another in a way that hopefully mimics the processes of evolution that generate the modeled natural DNA.

The most important aspect of the repeat structure is the length of the duplicated strings. The gap between two copies of a duplicate string is also important, since duplicate strings which do not fall into the same sequencable fragment cannot possibly affect the sequencing of that fragment. Finally, the gap between duplicate-string pairs—the inverse of the expected number of duplicate pairs—must be included in the model. This value measures the distance between the first copy of one duplicated string and the first copy of the next duplicated string. Duplicate strings can be interleaved—for two strings *a* and *b*, they may have the order

$\dots a_1 \dots b_1 \dots a_2 \dots b_2 \dots$ —the gap between duplicate strings measures the space between  $a_1$  and  $b_1$ .

To simplify the discussion of the model slightly, a pair of duplicate strings will be called a ‘duplicate event’. The constructed model has three parameters to represent these values:

- $\tau$ , the length of the duplicate strings.
- $t$ , the ‘small gap’ or within-pair spacing between two occurrences of the same string.
- $T$ , the ‘large gap’ or spacing between two different pairs.

Unfortunately, the three parameters are not independent. Shorter duplicate strings occur with greater frequency than longer ones, so the large gap size  $T$  decreases with  $\tau$ . There may also be other subtle interactions between the parameters  $\tau$  and  $t$ , with the small gap size  $t$  decreasing as the size of the duplicated string  $\tau$  increases. Furthermore, the underlying Markovian generating model used as a starting point for the model produces many of the shorter ( $\tau \leq 16$ ) duplicate string events.

A starting point is needed to estimate appropriate model parameters, and the suffix tree offers a valuable visualisation of two of them. The maximum length  $\tau$  of the duplicate strings determines the length of the ‘tail’ of the suffix tree, and the large gap spacing  $T$  determines the incidence of duplicate events, or the height of the tail.

Figure C.4 shows the suffix tree for coding sequences (CDS) taken from human chromosome 1. There are a few extremely long repeated strings of length  $\tau \approx 1000$  which occur in these sequences. In *e. coli*, the duplicate strings are not this long, but there are still repeats of several hundred bases. Typically, extremely long repeats are separated by a small gap of  $t \approx 10^5$  bases. While these widely-spread pairs of long duplicated strings may be interesting biologically, they are not typical of the observed pattern of duplicate events. To make the construction of a model feasible, we need to restrict our focus to the duplicate strings which cause sequencing failures.

To restrict the model to relevant events, repeats which will not affect the performance of our sequencing method are ignored. First, the model is restricted to duplicated strings which occur within 20Kb of one another (events with a small gap

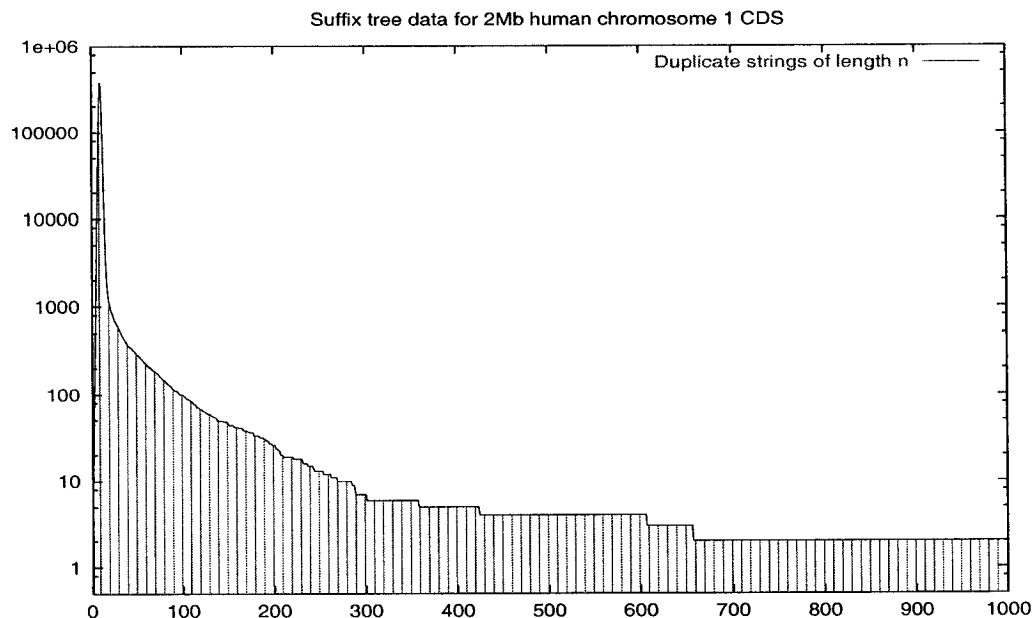


Figure C.4: The suffix tree for human coding sequences shows several extremely long repeated strings.

$t$  of less than 20Kb). We will also only attempt to model the distribution of string lengths  $\tau$  up to 100 bases; longer strings will be treated as though they have some length  $100 > \tau > 50$ . This will have the effect of reducing the average duplicate string length somewhat, but will not affect the usefulness of the model, as any duplicate event of length  $\tau \geq \lambda$  results in a Mode 2 failure.

#### C.4.1 Length of duplicate strings: $\tau$

Our model should predict the incidence of duplicate strings which are long enough to affect the performance of the sequencing algorithm. However, many of the shorter ( $\tau \leq 16$ ) events are produced by a Markovian model. To avoid overpredicting the incidence of shorter events in the model, we consider only events which are quite rare in the underlying Markovian sequence. The threshold was set at the point where the Markovian model generated fewer than 20% of the observed duplicate events of a particular length. For *e. coli*, this corresponded to  $\tau \geq 16$ .

A histogram of all duplicated strings of length  $\tau \geq 16$  was constructed to measure the distribution of  $\tau$  in *e. coli*. Using Maple we manually fit a mixture of an

exponential and a uniform random distribution to match the observed values.

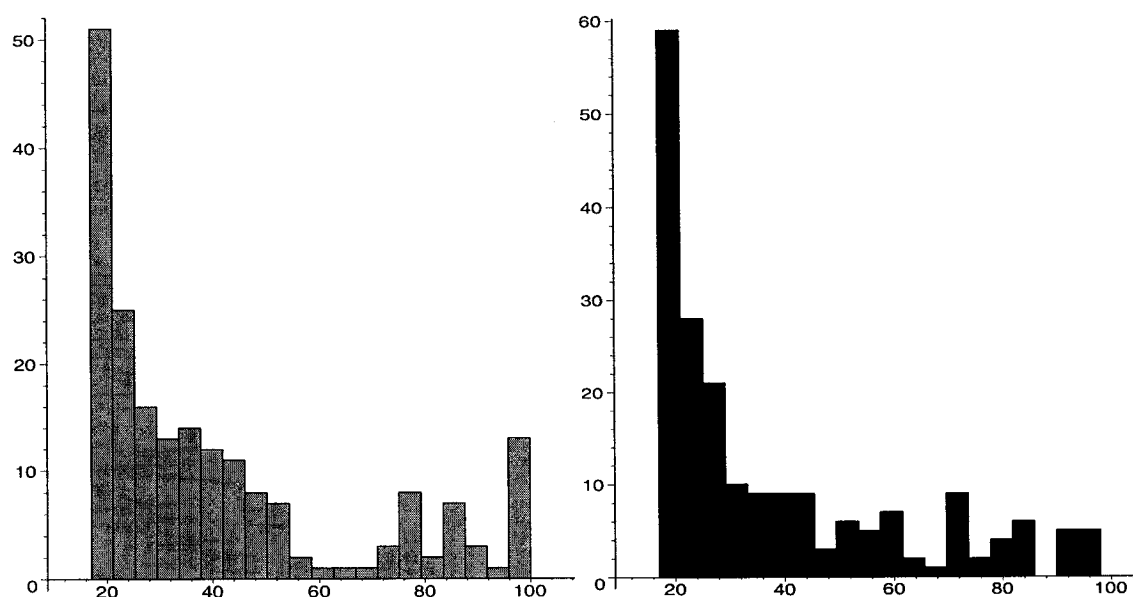


Figure C.5: The histograms of the lengths of duplicate strings observed in *e. coli* (left) and predicted by a blended model (right).

The parameters necessary to model the duplicate string length are the mean of the exponential  $e_{mean}$ , the offset of the exponential (since it begins at 15+, not at 0)  $e_{offset}$ , the range of the uniform distribution  $u_{min}$  and  $u_{max}$ , and the portion of the total distribution attributable to the exponential,  $e_{percent}$ .

The actual parameters used to generate the above data for *e. coli* are presented in the following table, along with parameters for a model of Human Chromosome 1 coding sequences.<sup>1</sup>

<sup>1</sup>The Human Genome Project is producing new annotations at a very rapid rate. The draft of Human Chromosome 1 which was initially used to produce this model was obtained in February 2001. It contained only *half* of the annotated coding sequences of an October 2001 draft of the same sequence. The distribution of repeated strings appears to have changed significantly, and the model had to be updated.

Parameter	<i>e. coli</i>	human chr. 1
$e_{mean}$	10	8
$e_{offset}$	17	16
$e_{percent}$	0.7	0.8
$u_{min}$	24	30
$u_{max}$	100	100

#### C.4.2 Gap between duplicated strings (intra-gap spacing): $t$

To accurately model the spacing between two copies of a duplicate string, it is again important to only include the events which are not produced by the Markov Model. The spacing between duplicate strings produced by the Markov Model should be uniformly distributed; the gap between the non-Markovian duplicates is not. To avoid interference from the Markovian events and their uniform distribution, only events of length  $\tau \geq 17$  are included in the model of the small gap size.

This model is further restricted to repeated strings which have a small gap size  $t \leq 20^4$  bases. 20Kb is well above the practical maximum length (for now) for single-spectrum sequencing of real DNA. Any duplicate strings which are separated by more than 20Kb will never fall into the same sequencable fragment.

With Maple it is a simple process to informally fit a distribution to the observed small gap sizes. Again, as in the model for  $\tau$ , a good fit can be achieved using a mixture of an exponential and a uniform distribution. The only modification is that no offset is used for the exponential distribution of  $t$ , and thus the parameter  $e_{offset}$  is not required. The parameters used for a model of *e. coli* and Human Chromosome 1 coding sequences are presented in the following table:

Parameter	<i>e. coli</i>	human chr. 1
$e_{mean}$	600	1200
$e_{percent}$	0.84	0.96
$u_{min}$	2000	2000
$u_{max}$	17500	20000

### C.4.3 Gap between events (inter-gap spacing): $T$

The gap between events (large gap, or  $T$ ) is tricky to model accurately: it appears to be extremely well represented as a mixture of two exponential distributions, one with a much shorter mean than the other. However, there is some difficulty in deciding which events to include in the model. While it would seem to be enough to simply ignore all duplicate strings of length  $\tau < 17$ —as with the model of the small gap size  $t$ —the *number* of events included directly determines the mean value of  $T$ .

Furthermore, the underlying Markovian *overpredicts* some of the shorter events; most notably those of length  $\tau = 14$ . It is not entirely clear what should be done to compensate for this overprediction. For the time being, it appears to be sufficient to include only events with  $\tau \geq 16$ ; the number of non-Markovian events ignored with length 15 or less is approximately equal to the number of Markovian events included with length 16 or greater.

To measure the mean values of the two exponential distributions, we separate all of the gap sizes into two sets: gaps of 500 bases or fewer, and gaps of greater than 500 bases. A random exponential distribution with mean equal to the mean gap size of the smaller set yields a nearly perfect match.

This method yields the following values for the exponential parameters of  $T$ :  $e_{small}$  and  $e_{big}$ , as well as the proportion of gaps distributed according to the smaller exponential  $s$ .

Parameter	<i>e. coli</i>	human chr. 1
$e_{small}$	88	45
$e_{big}$	30000	9300
$s$	0.65	0.72

## C.5 Evaluating the Model

In order to validate the model, we performed simulated trials on sequences generated by the two models, and the original natural DNA sources they were modeled on. The results are shown in Figure C.6. The performance curve for the original and modeled *e. coli* sequences agree remarkably well. For human coding sequences, there is a

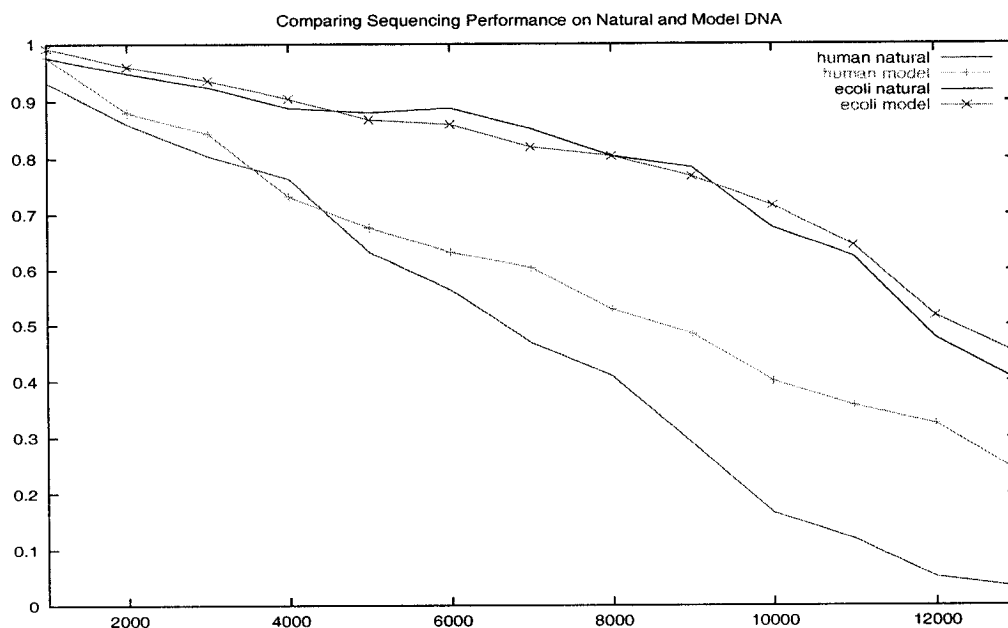


Figure C.6: Comparing natural and modeled *e. coli* and human coding sequences.

general trend in the right direction, but the model is not perfect. There are several possible reasons for this.

First (and most likely), there are factors affecting the sequencing of human DNA which are not reflected in the model. Second (and also quite likely), the selection effects introduced by the current methods of gene annotation could be introducing significant artifacts into the sequences being modeled—many of the coding sequences used to create the model are annotated based only on the prediction of a single gene-finding software tool. The available data is changing rapidly, and the model should be re-evaluated in often.

# Bibliography

- [BS91] W. Bains and G.C. Smith, A novel method for DNA sequence determination. *Jour. of Theoretical Biology*(1988), 135, 303-307.
- [DH00] K. Doi and H. Imai, Sequencing by hybridization in the presence of hybridization errors. In *Proceedings of the Workshop on Genome Informatics (GIW '00)*, 11:53-62, 2000.
- [DFS94] M.E. Dyer, A.M. Frieze, and S. Suen, The probability of unique solutions of sequencing by hybridization. *Journal of Computational Biology*, 1 (1994) 105-110.
- [D89] R. Drmanac, I. Labat, I. Bruckner, and R. Crkvenjakov, Sequencing of megabase plus DNA by hybridization. *Genomics*, (1989) , 4, 114-128.
- [F+94] M. Farach, M. Noordewier, S. Savari, L. Shepp, A. Wyner, and J. Ziv, On the entropy of DNA: Algorithms and measurements based on memory and rapid convergence. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994.
- [H+02] E. Halperin, S. Halperin , T. Hartman, R. Shamir, Handling long targets and errors in sequencing by hybridization. In *Proceedings of the Sixth Annual Conference on Computational Biology (RECOMB 2002)*, 2002, 176-185.
- [HP01] S.A. Heath and F.P. Preparata, Enhanced sequence reconstruction with DNA microarray application. In *proceedings of COCOON 2001*, (2001), 64-74.
- [HPY02] S.A. Heath, F.P. Preparata and J. Young, Sequencing by hybridization using direct and reverse cooperating spectra. In *Proceedings of the Sixth Annual Conference on Computational Biology (RECOMB 2002)*, 2002, 186-193.
- [KB01] G. Kelly, M. Behlke, Universal Bases. Technical Report, *Integrated DNA Technologies* (<http://www.idtdna.com>), 2001.



- [LB94] D. Loakes and D.M. Brown, 5-Nitroindole as a universal base analogue. *Nucleic Acids Research*, (1994), 22, 20, 4039-4043.
- [L+88] Yu.P. Lysov, V.L. Florentiev, A.A. Khorlin, K.R. Khrapko, V.V. Shih, and A.D. Mirzabekov, Sequencing by hybridization via oligonucleotides. A novel method. *Dokl. Acad. Sci. USSR*, (1988) 303, 1508-1511.
- [LLY00] K. Lancot, M. Li, E.H. Yang, Estimating DNA sequence entropy. *In proceedings of SODA 2000*.
- [LPSW02] Hon-Wai Leong, Franco Preparata, W.-K. Sung, and H. Willy On the Control of Hybridization Noise in DNA Sequencing-by-Hybridization. *In proceedings of WABI 2002*.
- [P89] P.A.Pevzner, l-tuple DNA sequencing: computer analysis. *Journ. Biomolecul. Struct. & Dynamics* (1989) 7, 1, 63-73.
- [P91] 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. *Journ. Biomolecul. Struct. & Dynamics* (1991) 9, 2, 399-410.
- [PFU99] F.P. Preparata, A.M. Frieze, E. Upfal. On the Power of Universal Bases in Sequencing by Hybridization. *Third Annual International Conference on Computational Molecular Biology*. April 11 - 14, 1999, Lyon, France, pp. 295-301.
- [PU00] F.P. Preparata and E. Upfal. Sequencing-by-Hybridization at the information-theory bound: An optimal algorithm. *Journal of Computational Biology*, 7, 3/4, 621-630 (2000).
- [S96] E.M. Southern, DNA chips: analysing sequence by hybridization to oligonucleotide on a large scale, *Trends in Genetics*, 12, 3, 110-115 (1996).
- [U95] E. Ukkonen, On-Line Construction of Suffix Trees. *Algorithmica*, 14: 249-260. (1995).