Finding, Evaluating, and Counting DNA Physical Maps

by

Lee Aaron Newberg

B.S. in Mathematics (Massachusetts Institute of Technology) 1986
B.S. in Physics (Massachusetts Institute of Technology) 1986
M.S. (University of California at Berkeley) 1991

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Richard M. Karp, Chair
Eugene L. Lawler
Terence P. Speed

1993

Finding, Evaluating, and Counting DNA Physical Maps

by

Lee Aaron Newberg

Abstract

Finding, Evaluating, and Counting DNA Physical Maps

by

Lee Aaron Newberg

Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Richard M. Karp, Chair

The International Human Genome Project seeks to analyze the DNA which can be found inside of every cell of every one of us. The goal is to map and sequence the DNA so that the location and chemical encoding of every inheritable trait is known. This dissertation includes several algorithms for finding and evaluating DNA clone orderings and probed partial digest maps as well as combinatorial results about them.

Using an extension of a statistical model given by E. Lander and M. Waterman to define the likelihood of a clone ordering conditioned upon hybridization data we give algorithms for computing the likelihood of a map and for finding a map of maximum likelihood. The dynamic programming algorithm for computing likelihoods runs in time $O(ms)$ where m is the number of oligonucleotide probes, and s is the size of the placement. The probability for each probe is computed via $O(s)$ conditional probabilities. We use the Expectation-Maximization technique to maximize likelihoods.

Using Occam's Razor to define the quality of clone orderings conditioned upon hybridization data we give algorithms for computing the simplicity of a map and for finding a map of maximum simplicity. We give an efficient algorithm that finds and evaluates the maximum interleaving, a best interleaving compatible with a given permutation of clones. We apply heuristics from the Traveling Salesperson Problem to search the space of permutations for a globally simplest interleaving.

Using a minimum-error heuristic to define the quality of a clone ordering we give algorithms for computing the number of errors implied by a map and for finding a map with a minimum number of errors. The algorithms perform well even in the presence of many data errors.

The statistical, simplicity, and minimum-error algorithms are more powerful than existing methods which restrict themselves to comparing clones in pairs. Our algorithms allow that the data from other clones may affect one's opinion about whether two clones

overlap and are able to produce better maps from less data.

We give an efficient branch-and-bound algorithm for creating a clone ordering from the data of a probed partial digestion experiment. The algorithm works well even when there are errors in the length measurements because, in part, of the algorithm's heavier reliance on the lengths of the shorter fragments which are measured more accurately in experiment. The algorithm will generate all solutions, in descending order of quality, until terminated.

We derive the number of topologically-distinct solutions $c(n)$ to the Clone Ordering Problem for n clones. We show $c(1) = 1, c(2) = 2, c(3) = 10$, and, for $n > 3$, that $c(n) = (4n-5)c(n-1) - (4n-7)c(n-2) + (n-2)c(n-3)$. We show that the exponential generating function $C(x) \stackrel{\text{def}}{=} \sum_{n=0}^{\infty} c(n)\frac{x^n}{n!} = \exp\left(\frac{1+2x-\sqrt{1-4x}}{4}\right)$. We show that $c(n) \sim \frac{e^{3/8}\sqrt{2}}{8n}\left(\frac{4n}{e}\right)^n$.

We derive a bound on the number of possible solutions to the Probed Partial Digest Mapping Problem. We show that a multiset of N lengths, measured without error, can have as many as $\Omega(N^t)$ solutions for any $t < \zeta^{-1}(2)$ where $\zeta(t)$ is the Riemann Zeta Function and $\zeta^{-1}(2) \approx 1.73$.

Richard M. Karp Date

To Dad and Mom:

By your example you have shown me how to chose a goal and strive for perfection.

Table of Contents

List of Figures

List of Tables

List of Symbols

For all Chapters

$\boldsymbol{A}$	The set of all atomic intervals.
$\boldsymbol{a}$	An atomic interval.
$\boldsymbol{a}_i$	An atomic interval
$\boldsymbol{C}$	The set of all clones.
$\boldsymbol{c}$	The coverage of the DNA.
$\boldsymbol{c}$	A clone.
$\boldsymbol{c}_i$	A clone.
$\boldsymbol{c}_0$	A dummy clone to the left of all other clones.
$\boldsymbol{c}_{n+1}$	A dummy clone to the right of all other clones.
$\boldsymbol{\chi}$	A marking of points, atomic intervals, or clones.
$\boldsymbol{\chi}(\boldsymbol{a},\boldsymbol{p})$	A marking of atomic intervals.
$\boldsymbol{\chi}(\boldsymbol{c},\boldsymbol{p})$	A marking of clones.
$\boldsymbol{\chi}(\boldsymbol{x},\boldsymbol{p})$	A marking of points.
$\boldsymbol{D}$	The hybridization data.
$\boldsymbol{D}(\boldsymbol{c},\boldsymbol{p})$	The hybridization datum for the clone c and the probe p.
$\mathrm{E}[\cdot]$	The expected value of a random variable.
$\boldsymbol{e}$	2.718281828...
$\boldsymbol{f}_n$	The false-negative rate for the hybridization data.
$\boldsymbol{f}_p$	The false-positive rate for the hybridization data.
$\boldsymbol{h}$	The height of a placement.
$\boldsymbol{I}$	An interleaving.
$\boldsymbol{i}$	The index of a clone, probe, or atomic interval.
$\boldsymbol{j}$	The index of a clone, probe, or atomic interval.
$\boldsymbol{\ell}$	The length of a clone or atomic interval.
$\boldsymbol{\ell}(\boldsymbol{a})$	The length of the atomic interval a.
$\boldsymbol{\ell}(\boldsymbol{c})$	The length of the clone c.
$\boldsymbol{M}$	The set of all markings of points.
$\boldsymbol{M}_A$	The set of all markings of atomic intervals.
$\boldsymbol{M}_C$	The set of all markings of clones.
$\boldsymbol{m}$	The number of probes.
$\boldsymbol{N}$	The length of the DNA.

$\boldsymbol{n}$	The number of clones (i.e., DNA fragments).
$\boldsymbol{P}$	The set of all probes.
$\boldsymbol{p}$	A probe.
$\boldsymbol{p}_i$	A probe.
$\boldsymbol{\pi}$	A permutation of clones.
$\boldsymbol{\pi}_i$	The ith permutation.
$\boldsymbol{\pi(i)}$	The index (i.e., label, name) of the clone with ith leftmost left endpoint.
$\Pr[\cdot]$	The probability of an event.
$\boldsymbol{s}$	The size of a placement.
$\boldsymbol{\vec{x}}$	A placement of clones.
$\boldsymbol{x}_i$	The left endpoint of clone c_i.
$\boldsymbol{y}_i$	The right endpoint fo clone c_i.

For Chapter 2

$\boldsymbol{A(a)}$	The set of clones active over the atomic interval a.
$\boldsymbol{\alpha(a)}$	A conditional probability defined by the atomic interval a. It is the likelihood of all the data to the left of a.
$\boldsymbol{\alpha_l(a,c)}$	A conditional probability defined by the atomic interval a and the clone c which is active over a. It is the likelihood of all the data to the left of a and the event that only those active clones starting at or to the left of c are marked to the left of a.
$\boldsymbol{\alpha_r(a,c)}$	A conditional probability defined by the atomic interval a and the clone c which is active over a. It is the likelihood of all the data to the left of a and the event that only those active clones starting at or to the left of c are marked to the left of the right endpoint of a.
$\boldsymbol{\beta(a)}$	A conditional probability defined by the atomic interval a. It is the likelihood of all the data to the right of a.
$\boldsymbol{c_{al}(a)}$	The clone active over the atomic interval a with the leftmost right endpoint.
$\boldsymbol{c_{ar}(a)}$	The clone active over the atomic interval a with the rightmost left endpoint.
$\boldsymbol{c_{\text{ending}}(a)}$	The clone that ends at the some place as the atomic interval a.
$\boldsymbol{D}_m$	The hybridization data that might arise from m probes.
$\boldsymbol{g}$	The total length of all the gaps.
$\boldsymbol{\gamma}$	The LaGrange multiplier for the length of the DNA.
$\boldsymbol{\gamma_c}$	The LaGrange multiplier for the length of the clone c.
$\boldsymbol{k}$	The number of islands.
$\boldsymbol{k'}$	The number of contigs.
$\boldsymbol{L(a)}$	The set of clones to the left of the atomic interval a.
$\boldsymbol{\nu(a,p)}$	The number of marks by probe p in atomic interval a.
$\boldsymbol{\nu_{ap}}$	The expected value of $\nu(a,p)$.
$\boldsymbol{R(a)}$	The set of clones to the right of the atomic interval a.
$\boldsymbol{S}$	The matrix of S_{ij} values.
$\boldsymbol{S_{ij}}$	A second partial derivative of the likelihood of the data. It is $\frac{\partial^2 \Pr[D\|\vec{x}_M]}{\partial\xi_i\partial\xi_j}$.

$\vec{\boldsymbol{\xi}}$	The vector of ξ_i's. It is usually implicitly restricted to those $(n-k)$ components that do not include gaps.
$\boldsymbol{\xi_i}$	The left endpoint of clone c_i relative to the left endpoint of the clone immediately to the left.

For Chapter 3

$\boldsymbol{A}$	The probe distance matrix of a_{ij} values.
$\boldsymbol{A^{(m)}}$	The probe distance matrix of $a_{ij}^{(m)}$ values.
$\boldsymbol{A^*}$	The interval distance matrix of a_{ij}^* values.
$\boldsymbol{a_{ij}}$	The number of probes for which the data for clones c_i and c_j differs.
$\boldsymbol{a_{ij}^{(m)}}$	For m probes, the number of probes for which the data for clones c_i and c_j differs.
$\boldsymbol{a_{ij}^*}$	The length of the symmetric difference between clones c_i and c_j as viewed as intervals.
$\boldsymbol{c^+(r,p)}$	The minimum number of positive elementary events needed to mark every clone in the run r with the probe p.
$\boldsymbol{c^-(r,p)}$	The minimum number of negative elementary events sufficient to imply that no clone in the run r contains p.
$\boldsymbol{c(\pi)}$	The cost of the permutation π.
$\boldsymbol{E[j]}$	The last clone that must end before clone c_j begins.
$\boldsymbol{f_D(I)}$	The weighted sum of the minimum number of positive and negative elementary events needed to imply the data D given the interleaving I.
$\boldsymbol{\gamma}$	The permutation of clones determined by the "greedy algorithm."
$GOOD(\boldsymbol{r,p})$	The event that the clones in the run r are validly marked by the probe p.
$\boldsymbol{I^*}$	The maximum interleaving with respect to a permutation of the clones.
$\boldsymbol{p_{ij}}$	The a priori probability that a given probe differentiates clones c_i and c_j.
$\boldsymbol{p_+}$	An approximation to the value that a given probe marks a given atomic interval.
$\boldsymbol{\phi}$	The weight for c^- terms in $f_D(I)$.
$\boldsymbol{r}$	A consecutive run of clones.
$\boldsymbol{S_I^+}$	A minimum set of positive elementary events for an interleaving I.
$\boldsymbol{S_I^-}$	A minimum set of negative elementary events for an interleaving I.
$\boldsymbol{\sigma}$	The permutation of clones determined by using the 2-OPT heuristic with $f_D(I) = \sum c^-(r,p) + \sum c^+(r,p)$.
$\boldsymbol{\tau}$	The true permutation of clones.
$\boldsymbol{\theta}$	The weight for c^+ terms in $f_D(I)$.
$\boldsymbol{x_{ij}}$	The length of overlap between clones c_i and c_j.
$\boldsymbol{\xi}$	A permutation of the clones which gives a shortest Hamiltonian path according to the distance matrix A.

For Chapter 4

$\boldsymbol{c}$	The minimum number of consecutive clones that contain a given probe.

$cost(\boldsymbol{\sigma})$	The cost of a signature σ; the distance to a nearest proper signature.
$cost_\tau(\boldsymbol{\sigma})$	The cost of a signature σ with respect to a proper signature τ.
$\boldsymbol{N}_\tau(\boldsymbol{\sigma})$	The number of false negatives in the signature σ assuming that τ is the true signature.
$\boldsymbol{P}_\tau(\boldsymbol{\sigma})$	The number of false positives in the signature σ assuming that τ is the true signature.
$\boldsymbol{\phi}$	The cost of a false negative.
$\mathrm{run}(\boldsymbol{i},\boldsymbol{j})$	The cost for σ_i minimized over all proper τ_i ending with j ones.
$\boldsymbol{S}$	The regular expression which defines a proper signature.
$\boldsymbol{\sigma}$	A probe's signature.
$\boldsymbol{\sigma}(\boldsymbol{i})$	The hybridization datum for the ith clone from the left.
$\boldsymbol{\sigma}_i$	The signature for the first (i.e., leftmost) i clones.
$\boldsymbol{\tau}$	A proper probe's signature.
$\boldsymbol{\theta}$	The cost of a false positive.

For Chapter 5

$\boldsymbol{A}(\boldsymbol{x})$	The generating function for $a(n)$.
$\boldsymbol{a}(\boldsymbol{n})$	The number of topologically distinct islands for n indistinguishable clones when an asymmetric island is considered distinguishable from its reflection.
$\boldsymbol{B}(\boldsymbol{x})$	The exponential generating function for $b(n)$.
$\boldsymbol{b}(\boldsymbol{n})$	The number of topologically distinct islands for n distinguishable clones when an island is considered indistinguishable from its reflection.
$\boldsymbol{C}(\boldsymbol{x})$	The exponential generating function for $c(n)$.
$\boldsymbol{c}(\boldsymbol{n})$	The number of topologically distinct clone orderings for n distinguishable clones when maps are considered indistinguishable from those obtained by permuted or reflecting islands.
$\boldsymbol{e}_i$	The location of the ith leftmost clone endpoint. There are $2n$ endpoints.
$\boldsymbol{k}$	The number of islands.
$\boldsymbol{x}$	A formal variable for the generating functions.

For Chapter 6

$\boldsymbol{B}$	The fragment that has both x_0 and y_0 as endpoints.
$\boldsymbol{b}$	The cost bound for depth-first search.
$\boldsymbol{C}$	The computed fragment lengths, $(X \cup B) + (Y \cup B) - B$.
$\boldsymbol{C}_i$	The i smallest elements of C.
$\boldsymbol{c}_i$	The ith smallest element of C.
$Cost(\boldsymbol{M},\boldsymbol{C})$	The cost of an optimal matching between M and C.
$Cost_\mu(\boldsymbol{M},\boldsymbol{C})$	The cost of the matching μ between M and C.
$cost(\boldsymbol{m},\boldsymbol{c})$	The cost of pairing a measured fragment length to a computed fragment length.
cutoff	A special element of C that matches the excess elements of M.

ϵ	The error level for the input data.
i	The number of Branch-and-Bound iterations.
ℓ	The length of the undigested clone.
M	The experimentally measured fragment lengths.
M_i	The i smallest elements of M.
m_i	The ith smallest element of M.
μ	A matching of measured fragment lengths to computed fragment lengths.
N	The fragments with neither x_0 nor y_0 as an endpoint.
$w(b)$	The number of nodes explored for bound b.
X	The fragments with x_0 but not y_0 as an endpoint.
x	The number of cut sites to the left of the probe site.
x_i	The ith nearest cut site to the left of the probe site.
Y	The fragments with y_0 but not x_0 as an endpoint.
y	The number of cut sites to the right of the probe site.
y_i	The ith nearest cut site to the right of the probe site.
Z	the fragment lengths not yet assigned to N, X, Y, or B.

For Chapter 7

$(a_1, \ldots, a_\ell)$	A mixed radix representation.
c_i	The ith cut site.
C	The set of cut sites.
$G(s)$	A generating function for $H(N)$ over a subset of the integers.
$H(N)$	The number of ways to factor N into the product of integers, each greater than one, where the order of the factors is significant.
ℓ	The size of the mixed radix basis.
N_i	The product of the first $i-1$ factors of the basis.
N	The number of lengths in S.
$(n_1, \ldots, n_\ell)$	A mixed radix basis.
#PPD(N)	The maximum number of solutions over all input multisets of size N.
#ppd(S)	The number of solutions for the input multiset S.
S	The input multiset of fragment lengths.
s	A formal variable for the generating functions.
S_N	The set $\{0, 1, \ldots, N-1\}$.
t	A formal variable for the generating functions.
X	The set of distances to the cut sites on the left.
Y	The set of distances to the cut sites on the right.
$Z(s)$	The generating function $\sum_N \frac{1}{N^s}$ over a subset of the integers.
$\zeta(s)$	The Riemann Zeta Function, $\zeta(s) = \sum_{N=1}^{\infty} \frac{1}{N^s}$

Definitions Index

Acknowledgements

The chapters of this dissertation are also being submitted to journals for publication. I thank my coauthors for permission to use the material in this dissertation. See Table 0.1.

Chapter	Cite
2	[New94b]
3	[AKNW95]
4	[New94a]
5	[New94c]
6	[KN95]
7	[NN93]

Table 0.1: Material Published (or to be Published) Elsewhere

I give thanks to Luciano Margara and Bruno Codenotti for the use of their Lin-Kernighan Traveling Salesperson Problem code. I also give many thanks to Dave Blackston, Diane Hernek, Jon Jarvis, Ivan Labat, Gene Lawler, Aleks Milosavljević, Heidi Newberg, Duska Sidjanin, Terry Speed, Ramesh Subramonian, David Wolfe, and Geoff Zweig for their questions and suggestions.

I am most indebted to my collaborators Farid Alizadeh, Dalit Naor, Deborah Weisser, and especially my advisor, Dick Karp for their contributions. The ideas and feedback from them have greatly strengthened this work.

Chapter 1

Foundations

Deoxyribonucleic acid (DNA) is a long molecule in which our genetic endowment is encoded. It determines what we inherit from our parents. The DNA molecule may be regarded as two intertwined **strands** each consisting of a **sequence** of **nucleotides**: adenine, thymine, cytosine, and guanine. These nucleotides are represented by the letters $\{A, T, C, G\}$ and a DNA strand can be thought of as a string of these letters.

A single strand of DNA is **chiral**, meaning that it is distinguishable from its mirror reflection. That is, a sequence of nucleotides is biologically different from the reverse sequence of nucleotides. By convention, single-strand DNA sequences are written from left to right in what is known as the $5'$ **to** $3'$ (five-prime to three-prime) direction. The paired strands of double-stranded DNA run in opposite directions. The nucleotides A and T are **complements**, as are C and G, and the sequences of the paired strands are **reverse complements**. See Figure 1.1.

A nucleotide together with its complement on the other strand is called a **base pair**. Human DNA molecules (**chromosomes**) are 10^7 to 10^8 base pairs long. The collec-

$3' \longleftarrow$ CAGAAATCTCAGTGAGTGAGAGTACATTCC $\longleftarrow 5'$

$5' \longrightarrow$ GTCTTTAGAGTCACTCACTCTCATGTAAGG $\longrightarrow 3'$

Figure 1.1: Sequences Which Are Reverse Complements.

tion of all 46 DNA molecules in the nucleus of a human cell is 3×10^9 base pairs long and is known as the human **genome**.

Dispersed throughout the human genome, accounting for but 10% of it, are approximately 10^5 short substrings, typically 3000 to 5000 base pairs long, known as **genes**. Genes are chemical encodings of information which determine our physical traits. It is not known whether the remaining 90% of the genome contains useful information. The fundamental goal in analyzing a genome is to locate and sequence genes. There are many approaches and subgoals. Both [Wil91] and [Kar93] provide broad overviews of the subject; the latter from a theory of computer science perspective. The biological technologies involved are described in [ABK+89].

1.1 Physical Maps

The process of gene location is achieved via **physical maps**. A physical map could be compared to a map of a city. A city map gives the location of landmarks (which might include museums, hospitals, highways, and streets) throughout the city whereas a physical map gives the location of **markers** (which come in many varieties) distributed throughout the genome. Although these markers need not be genes they are useful because they are easily detectable experimentally. Through various experimental and statistical means the markers which are close to an unmapped gene can be found. With a physical map showing the location of the markers the approximate location of the gene can be determined immediately.

A **restriction map** is a physical map that shows the locations where the genome is cut by a **restriction enzyme**. A restriction enzyme recognizes a sequence of 5 to 10 base pairs and cuts the genome at occurrences of the sequence. These locations are called **cut sites** or **restriction sites**. The process of cutting is known as **digestion** and the result is known as a **digest**.

A **clone ordering** (also known as a **contig map** or **cloned DNA map**) is a kind of physical map that provides the locations of short DNA fragments. Because these fragments are produced by cloning (i.e., copying) they are known as **clones**. A collection of clones is called a **library**. A clone is a copy of an interval of the DNA and is typically 10^3 to 10^6 base pairs long. Because these clones are smaller than the long DNA molecules it is easier to perform experiments on them. In particular, one can more easily locate

and **sequence** (i.e., find the string of base pairs encoding) a gene once a fragment that contains it is known. A library of clones is built by first making many copies of the original genome. These copies are then cut up into fragments in different ways using a variety of restriction enzymes. Those fragments that are of an easily-handled length are retained and cloned. Because the fragments come from many copies of the original genome they need not represent disjoint substrings of it. Those clones that are not disjoint are said to **overlap**. The total length of all the clones divided by the length of genome is called the **coverage** of the library and is typically between 5 and 20. (See [CdJB+89, BSP+90, ABK+89] for more information on the biological aspects of creating clones.)

The task of finding a clone ordering from analysis of the clones is called the **Clone Ordering Problem** (or **Contig Reconstruction Problem**). Dependent upon the amount of information available, a clone ordering for n clones may be known at various levels of detail. A **placement** of clones is a precise specification of the location on the DNA of each clone. When the length of each clone is known a placement will be denoted by an n-dimensional vector $\vec{x}$ where the ith coordinate x_i is the location of the left endpoint of the clone labeled i. An **interleaving** of clones is a specification of the linear order of the $2n$ clone endpoints on the DNA. Each interleaving is an equivalence class of topologically equivalent placements. A **permutation** of clones is a specification of the linear order of the n left endpoints of the clones. Where the right endpoints fall within the linear order of the left endpoints is not specified. Each permutation is an equivalence class of interleavings or placements. Although knowledge of the correct placement would be ideal, knowledge of the interleaving of a spanning set of clones is sufficient for many biological purposes. Knowledge of the permutation is sufficient in many cases.

It will be convenient to define an **atomic interval** with respect to a placement as a maximal interval of the form $[x, y)$ which does not contain any clone endpoint in its interior. If there are n clones, there will be $2n$ clone endpoints and the DNA will decompose into $2n + 1$ atomic intervals which are disjoint. A clone is **active** in an atomic interval if it contains the atomic interval. The **height** of an atomic interval is the number of clones that contain it. The **size** of a placement is the sum of the heights of its atomic intervals. The maximum of the heights is the **height of the placement**.

There may be some portions of the genome not covered by any of the clones. These regions are called **gaps**. The remaining parts of the genome comprise a collection of connected components known as **islands**. Those islands with more than one clone are also

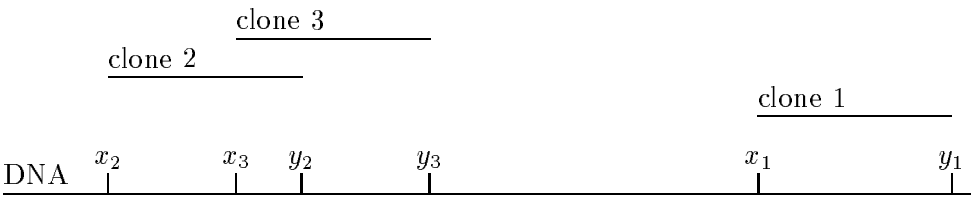

Figure 1.2: A Placement of Clones

called **contigs**.

For example, for three clones of length 1000 on a DNA fragment of length 5000, the placement $\vec{x} = (3904, 541, 1203)$ is shown in Figure 1.2. The clones' right endpoints are $\vec{y} = (4904, 1541, 2203)$. The interleaving implied by this placement is $x_2 < x_3 < y_2 < y_3 < x_1 < y_1$. The permutation implied by this placement or this interleaving is $x_2 < x_3 < x_1$. The intervals [2023, 3904) and [541, 1203) are two of the seven atomic intervals. The former is a gap and for the latter, only clone 2 is active. Clones 2 and 3 form an island that is a contig. Clone 1 is an island.

1.2 Fingerprinting and Hybridization

In the process of creating the clones, their original positions in the genome are lost. The information is recovered by **fingerprinting** the clones. In the most general sense, a fingerprint is a description of a clone's features sufficient to distinguish it from other clones. Clones can be fingerprinted via **hybridization**: An **oligonucleotide** is a small single-stranded DNA molecule which **hybridizes** (i.e., attaches) to its reverse complement wherever it occurs on a single strand of DNA. Typically an oligonucleotide is 6 to 10 nucleotides long and hybridizes to many places on the DNA. Clones are shorter than the full DNA so most of them will hybridize to an oligonucleotide only once or not at all. If an oligonucleotide is labeled, radioactively or fluorescently, it is called a **probe** and its hybridization to a clone becomes easily detectable. Although it is possible to detect that there is at least one hybridization it is much more difficult to determine the number of places that a probe hybridizes to a clone.

A large scale hybridization experiment may involve $n =$ thousands of clones and

m = hundreds of probes. (See [DSL+90, CNH+90, EL89, HSB89, DDL+92, DDS+93] for biological descriptions of hybridization experiments.) For each clone, the experiment tells us which probes hybridize to it. This information is an example of a clone fingerprint. This collection of clone fingerprints is called the **hybridization data**. Using this data the task is to build a clone ordering.

The task is analogous to the following problem. Suppose you take a very long one-column newspaper article and make many copies of it. You then rip up these copies in different ways producing fragments of the article. For each fragment and for each of a list of keywords you are told whether or not the fragment contains at least one occurrence of the keyword. From this data you must reconstruct the positions of the fragments in the original article. Note that only the existence, but not the number of occurrences, of each keyword is given. Furthermore, the order of the keywords occurring in a fragment is not given. Thus this task is different from the Shortest Superstring Problem discussed in [BJL+94, Tur89].

A **marking** is a description of where the probes occur on the DNA. As with a clone ordering, the amount of information available determines how precisely the marking can be described. The most precise description gives the exact location of each probe hybridization. Less precisely, a marking may indicate which atomic intervals contain which probe sites. Even less precisely, a marking may only indicate which clones hybridize to which probes.

Those points, atomic intervals, and clones which hybridize to a probe are said to be **marked** by the probe. A proposed marking which is consistent with the hybridization data is called **valid**.

1.3 Probed Partial Digests

Hybridization is not the only way to fingerprint a clone. The fingerprint can be a restriction map of the clone. In this case, a restriction enzyme that cuts frequently (i.e., in more places than the restriction enzymes used to create the clones) is most useful. These fingerprints are useful because clones which overlap will have restriction maps which overlap.

A **probed partial digestion** experiment will give data for constructing a restriction map of a clone. Under certain experimental conditions a restriction enzyme cuts at some but not all of the cut sites. This process is called **partial digestion**. In an (unprobed) partial digestion experiment many copies of a clone are partially digested at once. For each pair of cut sites a fragment is obtained which is a copy of the DNA between

those two cut sites. However, it is not known which pair of cut sites produces which fragment. The approximate lengths of these fragments are measured using a process called **gel electrophoresis**.

For a **probed partial digestion** experiment, a probe is chosen which hybridizes to the uncut clone in a unique location. As with partial digestion experiments, the copies of the clone are partially digested and a fragment is obtained for each pair of cut sites. Using gel electrophoresis the length of each fragment that hybridizes to the radioactively or fluorescently labeled probe is measured. That is, the distance between cut sites is obtained for each pair that straddles the probe site. As with unprobed partial digestion, it is not known which pair of cut sites produces which measurement.

1.4 Overview

This dissertation includes several algorithms and combinatorial arguments about DNA physical maps. Part I gives algorithms for constructing clone orderings from hybridization data. These algorithms use different quality measures to evaluate proposed clone orderings. The measure which is best depends upon the circumstances under which the data is collected so we include them all.

In Chapter 2 we use a plausible statistical model to define the likelihood of a clone ordering conditioned upon the results of hybridization experiments. We provide algorithms for computing the likelihood of a map and for finding a map of maximum likelihood.

In Chapter 3 we use Occam's Razor to define the quality of clone orderings — a map is good if it is simple. That is, a map is good if little additional information is needed to reconstruct the results of the hybridization experiment. We give an efficient algorithm that finds and evaluates a best interleaving compatible with a given permutation of clones. We apply heuristics from the Traveling Salesperson Problem to search the space of all permutations for a globally best interleaving.

In Chapter 4 we evaluate clone orderings based upon a minimum-error heuristic. In the presence of experimental error, there may be a difference between that data predicted by a clone ordering and the actual hybridization data. Those maps which predict the actual results with the minimum number of errors are considered best. We give an efficient algorithm that evaluates a permutation of clones. As with Chapter 3 we can apply heuristics from the Traveling Salesperson Problem to search the space of all permutations for one that

is globally best.

The algorithms of Chapters 2, 3, and 4 are more powerful than existing methods (see [CAT93, CNH+90, EL89]) which restrict themselves to comparing clones in pairs. Our algorithms allow that the data from other clones may affect one's opinion about whether two clones overlap. Hence, our algorithms are able to produce better maps from less data.

Part I ends with Chapter 5 in which we count the number of possible solutions to the Clone Ordering Problem. We give a generating function, recurrence relation, and asymptotic limit for the number of topologically-distinct interleavings.

Part II discusses the Probed Partial Digest Problem. In Chapter 6 we give an efficient branch-and-bound algorithm for constructing a clone's restriction map from the data of a probed partial digest experiment. The algorithm produces one or more candidate solutions, each of which designates the locations of the cut sites and specifies the end points of each fragment.

In Chapter 7 we count the number of possible solutions to the Probed Partial Digest Mapping Problem. We provide a lower bound to the number of solutions for worst-case experimental data.

We have two hopes for this dissertation. Firstly, we hope that it will provide a broad overview and some in-depth examples for the computer science theorist interested in computational biology. Secondly, we hope that the algorithms in the following chapters will become generally accepted in biology laboratories and will speed the progress of the Human Genome Project.

Part I

Hybridization Mapping Problem

Chapter 2

A Statistical Approach

(This material will also appear as [New94b].)

In this chapter we give statistical algorithms for finding and evaluating clone orderings based upon results from hybridization experiments of DNA fragments to oligonucleotide probes. Because there may be more than one map that is consistent with the data, we wish to find those maps which are best. In this chapter we describe a plausible statistical model and define "best" to mean most likely according to this model.

2.1 Model and Definitions

The statistical model we use is an extension of the one presented in [LW88].

The **DNA** is represented by the $[0, N)$ interval of the real line. All lengths can be scaled so N is arbitrary. Typically it is chosen so that the length of clone is approximately 1. The fact that DNA can be cut only between nucleotides is ignored and cuts are allowed at any real coordinate.

Each of n distinguishable **clones** is represented by a subinterval of known length which, for technical reasons, contains its left endpoint but not its right endpoint. A priori, each clone may occur uniformly at random on the DNA. That is, the left endpoint of a clone of length ℓ is equally likely to be anywhere in the interval $(0, N - \ell)$. The case that two or more clone endpoints coincide or that a clone endpoint coincides with an endpoint of the DNA has probability zero and will not be considered. The length may vary from clone to clone though we will write simply ℓ rather than $\ell(c)$ when the clone is obvious. The library of all clones is denoted by C.

Each of m distinguishable **probes** is represented by a finite set of points on the DNA which represents the locations to which it hybridizes. A priori, these points are described by a Poisson process on the DNA. That is, the probability that there is exactly one occurrence of a probe in an interval $[x, x + dx)$ is $\lambda dx + o(dx)$ independent of any occurrences in any set disjoint from $[x, x + dx)$. The probability of two or more occurrences is $o(dx)$. The rate λ is independent of the interval chosen. It may vary from probe to probe though we will write simply λ instead of $\lambda(p)$ when the probe is obvious. The collection of all probes is denoted by P. For example, see [Bev69, Ric88] for a description of Poisson processes.

The **hybridization data** is denoted by a function D. If the data is error-free it describes which clones contain which probes. More precisely,

$$D(c,p) \ = \left\{ \begin{array}{ll} 1 & \text{if } c \text{ contains at least one occurrence of } p \\ 0 & \text{otherwise.} \end{array} \right\}$$

If errors in the data are possible, they will be described by a false-negative rate f_n and a false-positive rate f_p. If a clone c contains a probe p then $D(c,p) = 1$ if and only if there is no false-negative for this datum, i.e., with probability $1 - f_n$. If c does not contain p then $D(c,p) = 1$ if and only if there is a false-positive for this datum, i.e., with probability f_p. Otherwise, $D(c,p) = 0$. The error rates f_n and f_p are assumed to be known in advance. They are allowed to be dependent on the clone-probe pair in question though we will write simply f_p and f_n rather than $f_p(c,p)$ and $f_n(c,p)$.

In the above definitions, the probability distributions for all clones, all probes, and all data errors are mutually independent.

A placement will be denoted by a vector $\vec{x}$ where the ith coordinate $x_i \in (0, N - \ell)$ is the location of the left endpoint of the clone labeled i. An interleaving and a permutation are equivalence classes of placements as previously defined. The collection of atomic intervals will be denoted by A. The length of an atomic interval $a \in A$ is denoted by $\ell(a)$ or just ℓ if a is obvious. The height of the placement is denoted by h and its size by s.

In its most precise form a marking is denoted by a function

$$\chi(x,p) \ = \left\{ \begin{array}{ll} 1 & \text{if probe } p \text{ occurs at the point } x \\ 0 & \text{otherwise.} \end{array} \right.$$

The set of all such markings is denoted by M. Any marking of points implicitly implies a

marking of the atomic intervals.

$$\chi(a,p) = \begin{cases} 1 & \text{if } \exists x \in a \text{ such that } \chi(x,p) = 1 \\ 0 & \text{otherwise} \end{cases}$$

The set of all markings of atomic intervals is denoted by M_A. Any marking of points or of atomic intervals implicitly implies a marking of the clones:

$$\chi(c,p) = \begin{cases} 1 & \text{if } \exists a \subseteq c \text{ such that } \chi(a,p) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The set of all such markings is denoted by M_C.

For a probe p, those points, atomic intervals, and clones for which $\chi = 1$ are said to be **marked** by p. A marking χ of any type is called **valid** if for all c and for all p, $\chi(c,p) = D(c,p)$ where D is the error-free data and $\chi(c,p)$ is the implied marking of clones. A clone c is validly marked if for all p, $\chi(c,p) = D(c,p)$.

Real DNA is not random and contains highly repetitive subsequences and thus both the assumption that the clones' left endpoints are uniformly distributed and the assumption that probe occurrences are Poisson are suspect. Fortunately, many of the repeated sequences are known (e.g., ALU (see [JM91, JWM92])). If the restriction enzymes and oligonucleotides are chosen so that the short sequences they detect are not subsequences of ALU, etc. the assumptions are reasonable, though still not entirely accurate.

2.2 Likelihood of a Placement

Our task is to find a likely clone ordering at the desired level of detail. For instance, we may wish to find a placement $\vec{x}$ which is most likely. That is, we wish $\vec{x}$ to maximize $\Pr[\vec{x}|D]$ the probability of the placement given the hybridization data. Using Bayes' Theorem we have

$$\Pr[\vec{x}|D] = \frac{\Pr[D|\vec{x}]\Pr[\vec{x}]}{\Pr[D]}.$$

Because of our data model, $\Pr[\vec{x}]$ is independent of $\vec{x}$. Furthermore, although the a priori likelihood of the hybridization data $\Pr[D]$ may be hard to calculate, it is also a constant independent of $\vec{x}$. Thus, a most likely placement also maximizes $\Pr[D|\vec{x}]$.

Because the probes are independent, the probability of the hybridization data D is just the product over probes of the probability of that probe's hybridization data. Without

loss of generality, in the rest of this section we assume that there is only one probe. We describe a dynamic programming algorithm for computing the likelihood of a single probe's hybridization data D given a clone placement $\vec{x}$.

We calculate the likelihood of D by way of the likelihood of a marking using the formula

$$\Pr[D|\vec{x}] \quad = \quad \sum_{\chi \in M_A} \Pr[D|\chi, \vec{x}] \Pr[\chi|\vec{x}]. \tag{2.1}$$

A priori, the probability that the probe is not contained in an atomic interval $a \in A$ of length ℓ is given by the formula $e^{-\lambda\ell}$ where λ is the known Poisson rate for p. The probability that a contains one or more occurrences of p is $1 - e^{-\lambda\ell}$. Because the probe is Poisson, the a priori probability of a marking χ is

$$\Pr[\chi|\vec{x}] = \prod_{a \in A} \begin{cases} 1 - e^{-\lambda\ell} & \text{if } \chi(a,p) = 1 \\ e^{-\lambda\ell} & \text{if } \chi(a,p) = 0. \end{cases} \tag{2.2}$$

If there are no false negatives or positives, $\Pr[D|\chi, \vec{x}]$ is 1 if χ is valid and is 0 otherwise. More generally, when false negatives and positives are possible, the likelihood is

$$\Pr[D|\chi, \vec{x}] = \prod_{c \in C} \begin{cases} 1 - f_n & \text{if } D(c,p) = 1 \text{ and } \chi(c,p) = 1 \\ f_p & \text{if } D(c,p) = 1 \text{ and } \chi(c,p) = 0 \\ f_n & \text{if } D(c,p) = 0 \text{ and } \chi(c,p) = 1 \\ 1 - f_p & \text{if } D(c,p) = 0 \text{ and } \chi(c,p) = 0. \end{cases} \tag{2.3}$$

Since there are $2n + 1$ atomic intervals, for a single probe there are 2^{2n+1} possible markings of atomic intervals. Thus, the direct calculation of $\Pr[D|\vec{x}]$ using Equation (2.1) would be quite costly.

2.2.1 Definition of $\alpha(a), \alpha_l(a,c)$, and $\alpha_r(a,c)$.

We will use a dynamic programming algorithm that calculates $\Pr[D|\vec{x}]$ by scanning the DNA from left to right. The algorithm is in the spirit of those described in [Rab89] for Hidden Markov Models.

The placement $\vec{x}$ implies a permutation of the clones which is based on the order of their left endpoints. Without loss of generality, we will assume that the clones have been relabeled so that they are called $c_1, \ldots, c_n$ from left to right. To simplify matters we will introduce two dummy clones c_0 and c_{n+1}. They do not have a precise location on the DNA

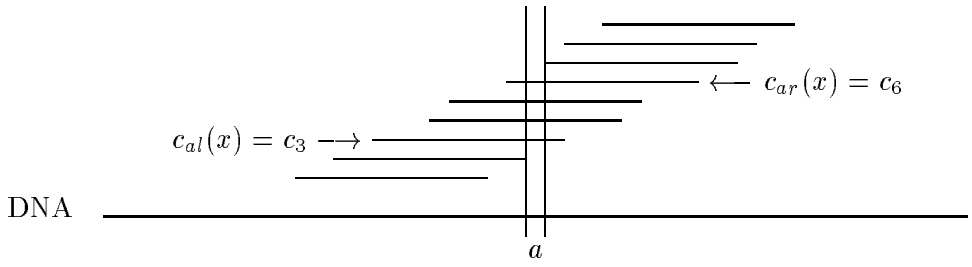

Figure 2.1: The Clones Left of, Active at, and Right of a. Clones are shown above the portion of the DNA which they represent. Clones $\{c_1, c_2\}$ are to the left of a; clones $\{c_3, c_4, c_5, c_6\}$ are active at a; and clones $\{c_7, c_8, c_9\}$ are to the right of a.

and do not affect the definition of the atomic intervals but, are considered to be respectively entirely to the left of and entirely to the right of all other clones. Clone c_i is said to *begin before* or *begin to the left of* clone c_j if $i < j$. The $2n + 1$ atomic intervals are labeled $a_0, \ldots, a_{2n}$ from left to right.

For an atomic interval a, let $L(a)$ be the set of non-dummy clones that are entirely to the left of a on the DNA; let $A(a)$ be the set of those clones that are active over a; and let $R(a)$ be the set of non-dummy clones that are entirely to the right of a. See Figure 2.1.

The sets $L(a), A(a)$, and $R(a)$ are disjoint and their union is C. If no clone includes another (because, for instance, all clones are the same length) then the sets $L(a), A(a)$, and $R(a)$ will contain clones with consecutive indices. All indices in $L(a)$ will be less than all indices in $A(a)$ which in turn will be less than all indices in $R(a)$.

We calculate $\Pr[D|\vec{x}]$ by considering the DNA from left to right. We define $\alpha(a)$ for every atomic interval $a \in A$ in such a way that $\alpha(a_0) = 1$ and $\alpha(a_{2n}) = \Pr[D|\vec{x}]$. Each $\alpha(a)$ can be computed from the values for the preceding atomic intervals. This gives an algorithm for calculating $\Pr[D|\vec{x}]$.

Let

$$\begin{aligned} \alpha(a) &= \Pr[(\bigwedge_{c \in L(a)} D(c,p))|\vec{x}] \\ &= \sum_{\chi \in M_A} \Pr[(\bigwedge_{c \in L(a)} D(c,p))|\chi, \vec{x}] \Pr[\chi|\vec{x}]. \end{aligned} \tag{2.4}$$

It will be convenient to put markings into equivalence classes based on which clones

of $A(a)$ they mark when the marks to the right of a point $x \in a$ are ignored. The above sum over all markings will be broken up into sums over these classes. Because of the following lemma we know that, regardless of the choice of x, there are $|A(a)|+1$ classes where $|A(a)|$ is the height of a.

Lemma 2.1 *Consider a single probe p. Let $\chi \in M$ be a marking, let $a \in A$ be an atomic interval, and let $x \in a$ be a point in the atomic interval. Considering only marks of χ to the left of x we have that there exists a clone $c \in A(a) \cup \{c_0\}$ such that*

- *Each clone of $A(a)$ with a left endpoint at or to the left of the left endpoint of c is marked, and*
- *Every other clone of $A(a)$ is not marked.*

The dummy clone c_0 is introduced so that the set of clones applicable in the first criterion can be empty.

Proof: We will show that when $c', c'' \in A(a)$ with the left endpoint of c' before the left endpoint of c'' then if c'' marked, so is c'. Since both c' and c'' are active at x, we have that $c'' \cap [0,x) \subseteq c' \cap [0,x)$. Since c'' is marked at or before x, c' must also be. ■

With this lemma in mind, for every $a \in A$ and every clone $c \in A(a) \cup \{c_0\}$ we define $\alpha_l(a,c)$ and $\alpha_r(a,c)$. For $a \in A$ and $c \in A(a) \cup \{c_0\}$, we define $\alpha_l(a,c)$ to be

$$\alpha_l(a,c) = \sum_{\text{some markings}} \Pr[(\bigwedge_{c' \in L(a)} D(c',p))|\chi,\vec{x}]\Pr[\chi|\vec{x}] \tag{2.5}$$

where the summation is over those markings in M_A satisfying two criteria. When only marks to the left of the *left* endpoint of a are considered:

- Each clone of $A(a)$ with a left endpoint at or to the left of the left endpoint of c is marked, and
- Every other clone of $A(a)$ is not marked.

If the set of clones for either of these conditions is empty then the condition is automatically satisfied. In the event that $L(a) = \emptyset$, the empty conjunction $\bigwedge_{c' \in L(a)} D(c',p)$ is defined to be true.

The function $\alpha_r(a,c)$ is defined similarly except that marks which are to the left of the *right* endpoint of a are considered.

From the lemma we know that, for a fixed a, every marking is included in exactly one of the $\alpha_l(a,c)$ and exactly one of the $\alpha_r(a,c)$. It follows immediately that $\alpha(a) = \sum_{c\in A(a)\cup\{c_0\}} \alpha_l(a,c) = \sum_{c\in A(a)\cup\{c_0\}} \alpha_r(a,c)$. Furthermore, $\alpha_l(a_0,c_0) = \alpha_r(a_0,c_0) = 1$ and $\alpha_l(a_{2n},c_0) = \alpha_r(a_{2n},c_0) = \Pr[D|\vec{x}]$. We will show how to compute the $\alpha_r(a_i,\cdot)$ values from the $\alpha_l(a_i,\cdot)$ values and how to compute the $\alpha_l(a_i,\cdot)$ values from the $\alpha_r(a_{i-1},\cdot)$ values. This will give us the algorithm we desire.

2.2.2 Calculation of $\alpha(a), \alpha_l(a,c)$, and $\alpha_r(a,c)$.

Let $c_{al}(a)$ be the clone in $A(a)$ with the leftmost right endpoint (i.e., the first clone to end) or, when $A(a) = \emptyset$, let $c_{al}(a) = c_{n+1}$. Let $c_{ar}(a)$ be the clone in $A(a)$ with the rightmost left endpoint (i.e., the last clone to begin) or, when $A(a) = \emptyset$, let $c_{ar}(a) = c_0$. See Figure 2.1.

First we will show how to calculate $\alpha_r(a,\cdot)$ from $\alpha_l(a,\cdot)$: Marks which occur in a may move a marking from one equivalence class to another. Any marking χ with $\chi(a,p) = 1$ will contribute to $\alpha_r(a, c_{ar}(a))$ regardless of the sum to which it contributed in the calculation of the $\alpha_l(a,\cdot)$'s. On the other hand, if $\chi(a,p) = 0$ then the marking will contribute to $\alpha_r(a,c)$ if and only if it contributed to $\alpha_l(a,c)$. Since the marks in a are independent of the marks before a, Equation (2.2) gives

$$\alpha_r(a,c) = \begin{cases} e^{-\lambda\ell}\alpha_l(a,c) & \text{if } c \in A(a)\cup\{c_0\}\backslash\{c_{ar}(a)\} \\ e^{-\lambda\ell}\alpha_l(a,c) + (1-e^{-\lambda\ell})\sum_{c'\in A(a)\cup\{c_0\}} \alpha_l(a,c') & \text{if } c = c_{ar}(a) \end{cases}$$

where ℓ is the length of a.

The calculation of $\alpha_l(a_i,\cdot)$ from $\alpha_r(a_{i-1},\cdot)$ breaks into two cases. If the clone endpoint separating a_{i-1} from a_i is the left end of a clone then $L(a_i) = L(a_{i-1})$ and $\alpha_l(a_i,c)$ is identically equal to $\alpha_r(a_{i-1},c)$ for all c in which the latter is defined. The latter is not defined for $c = c_{ar}(a_i)$ but in this case we have $\alpha_l(a_i, c_{ar}(a)) = 0$.

The harder case is that in which the endpoint separating a_{i-1} from a_i is the right end of a clone. In this case, $L(a_i) = L(a_{i-1}) \cup \{c_{\text{ending}}\}$ where $c_{\text{ending}} = c_{al}(a_{i-1})$ is the clone that is ending. Because of this, we must incorporate a factor of

$$\begin{aligned} \frac{\Pr[(\bigwedge_{c'\in L(a_i)} D(c',p))|\chi,\vec{x}]}{\Pr[(\bigwedge_{c'\in L(a_{i-1})} D(c',p))|\chi,\vec{x}]} &= \Pr[D(c_{\text{ending}},p)|(\bigwedge_{c'\in L(a_{i-1})} D(c',p)),\chi,\vec{x}] \\ &= \Pr[D(c_{\text{ending}},p)|\chi,\vec{x}] \end{aligned}$$

into the summand of Equation (2.5). Notice that the last equality follows from the fact that, given a marking χ, the datum $D(c_{\text{ending}}, p)$ is independent of the remaining hybridization data.

For a given marking χ, either c_{ending} is marked or it is not marked. The values $\alpha_r(a_{i-1}, c)$ for $\{c \in A(a_{i-1}) \cup \{c_0\} : c \text{ begins before the left endpoint of } c_{\text{ending}}\}$ are sums over markings that do not mark c_{ending}. As indicated by Equation (2.3), for these sums

$$\Pr[D(c_{\text{ending}}, p)|\chi, \vec{x}] = \begin{cases} f_p & \text{if } D(c_{\text{ending}}, p) = 1 \\ 1 - f_p & \text{if } D(c_{\text{ending}}, p) = 0. \end{cases}$$

The remaining values of $\alpha_r(a_{i-1}, \cdot)$ are sums over markings that mark c_{ending}. For these sums, Equation (2.3) indicates

$$\Pr[D(c_{\text{ending}}, p)|\chi, \vec{x}] = \begin{cases} 1 - f_n & \text{if } D(c_{\text{ending}}, p) = 1 \\ f_n & \text{if } D(c_{\text{ending}}, p) = 0. \end{cases}$$

In the special case where no clone includes another, the clones end in the same order in which they begin. Only $\alpha_r(a_{i-1}, c_0)$ is the sum of markings that do not mark c_{ending}. The remaining $\alpha_r(a_{i-1}, \cdot)$'s are sums over markings that mark c_{ending}. Thus in this situation, if $D(c_{\text{ending}}, p) = 0$ then

$$\alpha_l(a_i, c) = \begin{cases} f_n \alpha_r(a_{i-1}, c_{\text{ending}}) + (1 - f_p)\alpha_r(a_{i-1}, c_0) & \text{for } c = c_0 \\ f_n \alpha_r(a_{i-1}, c) & \text{for } c \in A(a_i) \end{cases}$$

and if $D(c_{\text{ending}}, p) = 1$ then

$$\alpha_l(a_i, c) = \begin{cases} (1 - f_n)\alpha_r(a_{i-1}, c_{\text{ending}}) + f_p \alpha_r(a_{i-1}, c_0) & \text{for } c = c_0 \\ (1 - f_n)\alpha_r(a_{i-1}, c) & \text{for } c \in A(a_i). \end{cases}$$

More generally, when clones can include other clones,

$$\alpha_l(a_i, c) = \begin{cases} \Pr[D(c_{\text{ending}}, p)|\chi(c_{\text{ending}}, p) = 0, \vec{x}] \cdot \alpha_r(a_{i-1}, c_0) & \\ + \Pr[D(c_{\text{ending}}, p)|\chi(c_{\text{ending}}, p) = 1, \vec{x}] \cdot \alpha_r(a_{i-1}, c_{\text{ending}}) & \text{for } c = c_0 \\ \Pr[D(c_{\text{ending}}, p)|\chi \text{ contributes to } \alpha_r(a_{i-1}, c), \vec{x}] \cdot \alpha_r(a_{i-1}, c) & \text{for } c \in A(a_i). \end{cases}$$

The formulae in this section can be combined into an algorithm that computes $\Pr[D|\vec{x}]$ in time $O(s)$ where s is the size of the placement $\vec{x}$. The algorithm computes $\alpha_l(a, \cdot)$ and $\alpha_r(a, \cdot)$ for every atomic interval a from left to right. An implementation of the algorithm that assumes that no clone includes another is given in Figure 2.2. Note that moving the memory allocation outside of this subroutine will be more efficient if the algorithm is called more than once.

```
double probability(
  int      n,                          /* The number of clones */
  int      N,                          /* The DNA length */
  double   start[ ],                   /* The array of clone left endpoints, sorted */
  double   len[ ],                     /* The array of clone lengths */
  int      data[ ],                    /* The hybridization data for the clones */
  double   rate,                       /* The Poisson rate of the probe */
  double   fp, fn)                     /* The false-positive and false-negative rates */
{
  int      first, last;                /* Clones [first, last] are active */
  double   position, newposition;      /* The left and right atomic-interval endpoints */
  double   length;                     /* The length of the atomic interval */
  double *alphal, *alphar;             /* alphal[0..clone] and alphar[0..clone] for the implied atomic interval */
  int      clone;                      /* The argument of alphal and alphar */

  alphal = malloc((clone + 1) * sizeof(double));          /* Allocate memory */
  alphar = malloc((clone + 1) * sizeof(double));          /* Allocate memory */
  start[0] = 0;      start[n+1] = N;   /* The dummy clones */
  len[0] = len[n+1] = 0;
  first = 1; last = 0;                 /* The atomic interval a0 */
  position = start[0];                 /* The left endpoint of a0 */
  clone = first - 1;                   /* The clone c0 will always be first - 1 */
  alphal[clone] = 1.0;                 /* The implied atomic interval is a0 */

  /* Repeat until the last clone ends */
  while (first != n + 1) {

    /* Compute alphar for this atomic interval */
    newposition = min(start[first] + len[first], start[last + 1]);
    length = newposition - position;
    alphar[last] = alphal[last];
    for (clone = first - 1; clone < last; clone++) {
      alphar[clone] = alphal[clone] * exp(-rate * length);
      alphar[last] += alphal[clone] * (1 - exp(-rate * length));}

    /* Compute alphal for the next atomic interval */
    position = newposition;
    if (start[first] + len[first] > start[last + 1]) {          /* A clone is beginning */
      for (clone = first - 1; clone <= last; clone++) {
        alphal[clone] = alphar[clone];}
      alphal[last + 1] = 0;
      last++;}                         /* update active clone range */
    else {                             /* A clone is ending */
      if (data[first] == 0) {          /* The clone ending does not hybridize to the probe */
        for (clone = first; clone <= last; clone++) {
          alphal[clone] = alphar[clone] * fn;}
        alphal[first] += alphar[first - 1] * (1 - fp);}
      else {                           /* The clone ending does hybridize to the probe */
        for (clone = first; clone <= last; clone++) {
          alphal[clone] = alphar[clone] * (1 - fn);}
        alphal[first] += alphar[first - 1] * fp;}
      first++;}}                       /* update active clone range */

  free(alphal);      free(alphar);     /* Deallocate memory */
  return alphar[n];                    /* The solution */
}
```

Figure 2.2: Dynamic Programming Algorithm To Compute $\Pr[D|\vec{x}]$ When No Clone Contains Another. The algorithm assumes that there is only one probe and that the clones have already been sorted by their left endpoints.

2.2.3 Most Likely Marking

For a given placement $\vec{x}$ and hybridization data D we may wish to find the likelihood of a most likely marking, $\max\{\Pr[\chi|D,\vec{x}] : \chi \in M_A\}$. By Bayes' theorem,

$$\Pr[\chi|D,\vec{x}] = \frac{\Pr[D,\chi|\vec{x}]}{\Pr[D|\vec{x}]}.$$

We have just described how to compute the denominator of the right-hand side. The maximum value of the numerator can be computed using a slight variation of that algorithm.

In each formula where an α_l or α_r value is computed from others, wherever addition of (possibly weighted) α_l or α_r values is indicated, a maximum function should be used instead. For instance, $e^{-\lambda\ell}\alpha_l(a,c) + (1-e^{-\lambda\ell})\sum_{c'\in A(a)\cup\{c_0\}}\alpha_l(a,c')$ should be replaced with

$$\max\left(\{e^{-\lambda\ell}\alpha_l(a,c)\} \cup \{(1-e^{-\lambda\ell})\alpha_l(a,c') : c' \in A(a)\cup\{c_0\}\}\right).$$

By keeping track of which α_l and α_r values depend on which other values, the markings which maximize the likelihood can be found efficiently. The details are left to the reader.

2.3 Most Likely Placement

We may find ourselves in a situation in which we know the interleaving I of a collection of clones but not their exact placement $\vec{x}$. The task is then to find a placement compatible with the interleaving which, given the hybridization data D, is most likely. That is, we wish to maximize $\Pr[\vec{x}|D,I]$. Using Bayes' Theorem we have

$$\Pr[\vec{x}|D,I] = \frac{\Pr[D,I|\vec{x}]\Pr[\vec{x}]}{\Pr[D,I]}.$$

Since, $\Pr[\vec{x}]$ and $\Pr[D,I]$ are independent of $\vec{x}$ it is sufficient to maximize $\Pr[D,I|\vec{x}]$. Furthermore, $\Pr[D,I|\vec{x}] = \Pr[D|\vec{x}]$ when $\vec{x}$ is compatible with I. Thus it is sufficient to maximize

$$\Pr[D|\vec{x}]$$

over all placements compatible with I. We will use the Expectation-Maximization technique (EM). The theoretical basis of EM is described in [DLR77] and a well written description of the technique can be found in [Rab89].

EM is not foreign to molecular biology. It is used in [LGA+87] to locate markers on DNA. In that paper, the calculations are based on DNA recombinations which are inferred

from genealogy data. We will base our calculations on markings which are inferred from hybridization data.

EM starts with a guess $\vec{x}$ for a most likely placement. In the general step, the most recent guess is improved, yielding a placement with a higher likelihood. A placement is improved in two steps: the Expectation step and the Maximization step. In the Expectation step, the hybridization data D and the current guess for the placement $\vec{x}$ are used to calculate statistics about the underlying marking which leads to the experimental data — For every probe p and for every atomic interval a, the expected number of occurrences, ν_{ap}, of p in a is calculated. In the Maximization step, the original data D is ignored. The statistics ν_{ap} are assumed to represent the true underlying marking and the placement $\vec{x}'$ which maximizes $\Pr[\nu_{ap}|\vec{x}']$ is calculated. This maximization problem is easier than the original and, although we will not prove it here, its solution $\vec{x}'$ is guaranteed to have a likelihood $\Pr[D|\vec{x}'] \geq \Pr[D|\vec{x}]$.

The two steps are repeated in alternation until the likelihood ceases to increase appreciably. Except in degenerate cases, EM converges geometrically (see [DLR77]) to a placement with a likelihood that is locally optimal. We conjecture that, acting on real hybridization data, EM will converge to a placement which is globally optimal among all those compatible with I.

2.3.1 The Expectation Step

The Expectation step can be carried out on each probe individually. As described below, three passes, each similar to the algorithm described in Section 2.2 are sufficient to calculate ν_{ap}. In the first pass $\alpha(a)$ is calculated and stored for all $a \in A$. The second pass is from right to left. It computes and stores $\beta(a)$. The function $\beta(\cdot)$ is the same as $\alpha(\cdot)$ with all left's and right's flipped. That is, let

$$\begin{aligned} \beta(a) &= \Pr[(\bigwedge_{c' \in R(a)} D(c', p))|\vec{x}] \\ &= \sum_{\chi \in M_A} \Pr[(\bigwedge_{c' \in R(a)} D(c', p))|\chi, \vec{x}] \Pr[\chi|\vec{x}]. \end{aligned} \tag{2.6}$$

The third pass computes ν_{ap} for each atomic interval. Let

$$\nu(a, p) = |\{x \in a : \chi(x, p) = 1\}|$$

be the number of marks in an atomic interval. We wish to calculate ν_{ap}, the expected number of occurrences of p in a given the placement $\vec{x}$ and the data D;

$$\nu_{ap} \stackrel{\text{def}}{=} \mathrm{E}[\nu(a,p)|D,\vec{x}].$$

Because each probe obeys a Poisson distribution, if we do not condition upon the data the expected number of occurrences of a probe in an atomic interval is just the product of the probe's rate λ and the atomic interval's length ℓ. Furthermore, since $\chi(a,p)=0$ if and only if $\nu(a,p)=0$ we have that $\mathrm{E}[\nu(a,p)|\chi(a,p)=1,\vec{x}] = \frac{\lambda\ell}{\Pr[\chi(a,p)\neq 0|\vec{x}]}$.

Because $\nu(a,p)$ does not depend on D when $\chi(a,p)$ and $\vec{x}$ are given we have that

$$\begin{aligned}
\nu_{ap} &= 0\cdot\Pr[\chi(a,p)=0|D,\vec{x}] \\
&\quad +\mathrm{E}[\nu(a,p)|\chi(a,p)=1,D,\vec{x}]\Pr[\chi(a,p)=1|D,\vec{x}] \\
&= 0+\frac{\lambda\ell}{\Pr[\chi(a,p)\neq 0|\vec{x}]}\frac{\Pr[\chi(a,p)=1|\vec{x}]\Pr[D|\chi(a,p)=1,\vec{x}]}{\Pr[D|\vec{x}]} \\
&= \frac{\lambda\ell}{\Pr[D|\vec{x}]}\Pr[D|\chi(a,p)=1,\vec{x}]
\end{aligned}$$

We can factor $\Pr[D|\chi(a,p)=1,\vec{x}]$ into three parts. The clones in $L(a)$ do not contain a nor any atomic interval to its right. Hence the data for these clones depends only on marks to the left of a. Similarly, the data for the clones in $R(a)$ depends only on marks to the right of a. Furthermore, when $\chi(a,p)=1$, the data for the clones in $A(a)$ is independent of any marks outside of a. Thus, using Equations (2.4) and (2.6) we have

$$\begin{aligned}
\nu_{ap} &= \frac{\lambda\ell}{\Pr[D|\vec{x}]}\alpha(a)\beta(a)\Pr[(\bigwedge_{c\in A(a)} D(c,p))|\chi(a,p)=1,\vec{x}] \\
&= \frac{\lambda\ell}{\Pr[D|\vec{x}]}\alpha(a)\beta(a)\prod_{c\in A(a)}\begin{cases} 1-f_n & \text{if } D(c,p)=1 \\ f_n & \text{if } D(c,p)=0 \end{cases}
\end{aligned}$$

The third pass and hence the Expectation step, the entire calculation of the ν_{ap}'s, can be accomplished in $O(s)$ time per probe.

2.3.2 The Maximization Step

We now use the ν_{ap} values to derive a better placement. If all the $\nu(a,p)$'s were known for the underlying marking, it would not be hard to compute and maximize their joint probability $\Pr[\nu|\vec{x}]$. Since probes occur according to a Poisson process, we have

$$\Pr[\nu|\vec{x}] = \prod_{a\in A}\prod_{p\in P}\frac{(\lambda\ell)^{\nu(a,p)}}{\nu(a,p)!}e^{-\lambda\ell}.$$

The EM technique requires that we use ν_{ap} instead of $\nu(a,p)$ in this equation. If there are no restrictions on the ℓ values, this is maximized when the placement gives the values

$$\ell(a) = \frac{\sum_{p\in P} \nu_{ap}}{\sum_{p\in P} \lambda(p)}.$$

For atomic intervals a which are gaps, the new $\ell(a)$ value will equal the old. This reflects the fact that D contains no information for determining the length of atomic intervals not contained by any clone.

The above solution may be infeasible under the restrictions that the lengths of the atomic intervals contained in a clone must sum to the known length of the clone, and that the sum of all the atomic intervals must sum to the DNA length N. If it is important to adhere to these restrictions, the maximization should be carried out using the method of LaGrange multipliers (see, for instance, [TF92]). We maximize $\log(\Pr[\nu|\vec{x}])$ modified by terms for the length restrictions. That is, ignoring terms which do not depend on the $\ell(a)$'s, we maximize

$$\left(\sum_{a\in A}\sum_{p\in P}[\nu_{ap}\log(\ell(a)) - \lambda(p)\ell(a)]\right) - \sum_{c\in C}\gamma_c\left(\sum_{a\subseteq c}\ell(a) - \ell(c)\right) - \gamma\left(\sum_{a\in A}\ell(a) - N\right)$$

where $\{\gamma\}\cup\{\gamma_c : c\in C\}$ are the LaGrange multipliers. The solution is the set of lengths and γ's and satisfies the multilinear equations

$$\begin{array}{lrcl} & \sum_{a\in A}\ell(a) & = & N \\ \text{For all } c\in C, & \sum_{a\subseteq c}\ell(a) & = & \ell(c) \\ \text{For all } a\in A, & \ell(a) & = & \frac{\sum_{p\in P}\nu_{ap}}{\sum_{p\in P}\lambda(p)+\sum_{c\supseteq a}\gamma_c+\gamma}. \end{array}$$

Iteration of the Expectation and Maximization steps yields a sequence of placements with likelihoods that converge geometrically to a local optimum.

2.4 Likelihood of an Interleaving

Rather than striving for the best placement we may wish to find the most likely interleaving given the hybridization data D. Since $\Pr[I|D] = \frac{\Pr[D,I]}{\Pr[D]}$ and since $\Pr[D]$ is a constant independent of I, it is sufficient to find an interleaving which maximizes $\Pr[D,I]$. In this section we discuss how to approximate $\Pr[D,I]$ for any interleaving I. The calculation proceeds via the formula

$$\Pr[D,I] = \int_I \Pr[D|\vec{x}]\Pr[\vec{x}]d\vec{x} \tag{2.7}$$

where the integral is n-dimensional and is restricted to those placements compatible with I.

Let $\vec{x}$ be a placement compatible with the interleaving I. Consider a change of variables for the integral of Equation (2.7). Let

$$\xi_i = \begin{cases} x_i & \text{if } i = 1 \\ x_i - x_{i-1} & \text{otherwise.} \end{cases}$$

The Jacobian (see [TF92]) of this change of variables is 1.

The atomic intervals a_0 and a_{2n} are gaps and there may be other atomic intervals that are gaps. Let g be the sum of the lengths of the gaps. Every placement $\vec{x}'$ derived from $\vec{x}$ by changing the lengths of the gaps (but leaving their sum equal to g) and leaving the lengths of the atomic intervals within the islands unmodified satisfies $\Pr[D|\vec{x}'] = \Pr[D|\vec{x}]$. Thus, we can easily integrate over the ξ_i which include gaps. Let k be the number of islands. The integral is similar to that of the integral over a k-dimensional simplex and provides a multiplicative factor of

$$\frac{g^k}{k!}$$

(By some definitions, two interleavings are considered equivalent if they are identical except that their islands may be permuted. In such a case, the factor should be g^k. If additionally, the interleaving obtained by flipping one or more contigs is considered equivalent then the factor should be $g^k 2^{k'}$ where k' is the number of contigs.)

We will use $\vec{\xi}$ to represent the those $\xi_i's$ which do not include a gap. We approximate the integral over these remaining $n - k$ dimensions by assuming that $\Pr[D|\vec{\xi}]$ approximates an $(n - k)$-dimensional Gaussian near its maximum and is negligible elsewhere. Putting aside the actual hybridization data for a moment, consider the probability distribution over all possible sets of hybridization data for m probes, $\{D_m\}$, where each of the D_m's is weighted by its probability of arising from the true underlying placement. We will argue that, with probability approaching 1 as $m \to \infty$, a randomly chosen D_m will have the property that $\Pr[D_m|\vec{\xi}]$ approximates a Gaussian near its maximum and is negligible elsewhere. (This will not be true for some cases for the underlying true placement. However, these cases arise with probability 0 according to our model.) We will thus conclude that in most cases $\Pr[D|\vec{\xi}]$ has the same property.

The argument that $\Pr[D_m|\vec{\xi}]$ is likely to be similar to a Gaussian relies on the independence of the probes. One can prove that when flipping an unbiased coin m times, with probability 1, the number of heads observed deviates from the expected number by

$o(m)$. For a fixed $\vec{\xi}$ one can argue similarly for $\log \Pr[D_m|\vec{\xi}]$ which is the sum of log-probabilities for m probes. With probability 1,

$$\begin{aligned} \log \Pr[D_m|\vec{\xi}] &= \mathrm{E}[\log \Pr[D_m|\vec{\xi}]] + o(m) \\ &= m \cdot \mathrm{E}[\log \Pr[D_1|\vec{\xi}]] + o(m) \end{aligned} \tag{2.8}$$

where the expectations are taken over the probability distribution on the D_m's. Now, $\mathrm{E}[\log \Pr[D_1|\vec{\xi}]]$ is maximized by a placement which we will call $\vec{x}_M$. Except in some cases for the underlying true placement which arise with probability 0 (no proof) this placement is unique up to a reapportionment of lengths among the gaps. Near $\vec{x}_M$, $\mathrm{E}[\log \Pr[D_m|\vec{\xi}]]$ is a downward facing paraboloid (i.e., its first derivatives are zero and the matrix of its pure and mixed second derivatives is negative definite.) The second derivatives are proportional to m and hence grow without bound as m goes to infinity. Because this growth is faster than the $o(m)$ from Equation (2.8) we conclude that the shape of $\Pr[D_m|\vec{\xi}]$ converges to a Gaussian near $\vec{x}_M$ as $m \to \infty$. For a point $\vec{x}$ not near $\vec{x}_M$ the difference

$$\mathrm{E}[\log \Pr[D_m|\vec{x}_M]] - \mathrm{E}[\log \Pr[D_m|\vec{x}]]$$

grows proportionally to m. For m large we conclude that $\Pr[D_m|\vec{x}]$ is negligible.

Armed with our Gaussian approximation, we can proceed. We redefine $\vec{x}_M$ to be a placement which maximizes $\Pr[D|\vec{x}]$). If $\Pr[D|\vec{x}_M]$ and the $O([n-k]^2)$ mixed and pure second derivatives of $\log \Pr[D|\vec{\xi}]$ at $\vec{x}_M$ are known the volume under the approximating Gaussian can be computed exactly. We can find $\vec{x}_M$ and $\Pr[D|\vec{x}_M]$ using the EM algorithm from Section 2.3.

The second-derivatives can be approximated to any desired degree of accuracy using differences. Let S be the $(n-k) \times (n-k)$ matrix of second-derivatives where the (i,j) entry of S is given by

$$S_{ij} = \left. \frac{\partial^2 \log \Pr[D|\vec{\xi}]}{\partial \xi_i \partial \xi_j} \right|_{\vec{x}_M}$$

Let $\vec{e}_i$ be the vector which has a one in the ith position and zeros elsewhere. Using $O([n-k]^2)$ iterations of the $\Pr[D|\vec{x}]$ algorithm we can approximate S. For sufficiently small $\Delta\xi$ we approximate

$$\begin{aligned} (\Delta\xi)^2 S_{ij} \approx\ & \log \Pr[D|\vec{\xi} + \Delta\xi\vec{e}_i + \Delta\xi\vec{e}_j] - \log \Pr[D|\vec{\xi} + \Delta\xi\vec{e}_j] \\ & - \log \Pr[D|\vec{\xi} + \Delta\xi\vec{e}_i] + \log \Pr[D|\vec{\xi}] \end{aligned}$$

If all the mixed second-derivatives were zero, the volume under the Gaussian would be

$$\frac{\Pr[D|\vec{x}_M]}{\prod_{c\in C}(N-\ell(c))}\left(\frac{(2\pi)^{n-k}}{\det(-S)}\right)^{1/2}. \tag{2.9}$$

Because S is a symmetric matrix there exists an orthogonal matrix R such that RSR^{T} is diagonal. The diagonal matrix has the same determinate as S. Furthermore, the Jacobian of the change of variables defined by R is $\det(R) = 1$. Thus Equation (2.9) gives the volume under the Gaussian even when the mixed second-derivatives are nonzero. The likelihood of the interleaving is

$$\left(\frac{g^k}{k!}\right)\left(\frac{\Pr[D|\vec{x}_M]}{\prod_{c\in C}(N-\ell(c))}\right)\left(\frac{(2\pi)^{n-k}}{\det(-S)}\right)^{1/2}.$$

Chapter 3

Using Occam's Razor

(This material will also appear as [AKNW95].)

In this chapter we give discrete algorithms for finding and evaluating clone orderings based upon results from hybridization experiments of DNA fragments to oligonucleotide probes. As we noted in Chapter 2 there may be more than one map that is consistent with the data and we wish to find those maps which are best. The statistical model of Chapter 2 may not be appropriate for a given situation and in this chapter we discuss an alternate definition of "best" clone ordering. In this chapter, we invoke Occam's Razor; the simpler the solution, the better it is.

3.1 Algorithm

We restrict ourselves to the noiseless case, in which the data is free of experimental error. Furthermore, we assume that all clones are of length one. We wish to determine the best interleaving of clones.

3.1.1 Definition of $f_D(I)$

We invoke the principle of Occam's Razor and we will define $f_D(I)$ to be a measure of the amount of additional information (beyond the knowledge of I) necessary to reconstruct a marking of atomic intervals that implies the data D. An interleaving which minimizes $f_D(I)$ is simplest and we will consider it best.

Consider a permutation of the clones. With respect to a particular probe p, define a **positive run** as a maximal sequence of consecutive clones, all of which are incident with

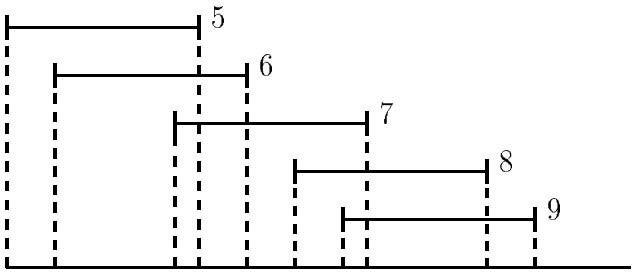

Figure 3.1: Placement of Clones in a Negative Run. None of these clones contain the probe p.

probe p, and a **negative run** as a maximal sequence of consecutive clones, none of which are incident with probe p.

Fix an interleaving I. We shall express the event D as a conjunction of nearly independent events associated with distinct runs. For any negative run (r, p), the event $GOOD(r, p)$ occurs if for every clone $c \in r$, probe p does not occur in any atomic interval which is contained by c. For any positive run (r, p), the event $GOOD(r, p)$ occurs if, for every clone c in the run, probe p occurs in some atomic interval which is contained by c but is not contained by any of the clones in the neighboring negative runs. We have that $D = \cap_{(r,p)} GOOD(r, p)$.

We now introduce a simple approximation to the information content of the event $GOOD(r, p)|I$. Define a **negative elementary event** as an event of the form "clone c does not contain probe p" (i.e., $D(c, p) = 0$); such an event implies $\chi(a, p) = 0$ for every atomic interval a contained in c. For a negative run (r, p) define $c^-(r, p)$ to be the minimum number of negative elementary events with respect to I needed to imply $GOOD(r, p)$. For instance, suppose that for a probe p, the sequence of clones $5 - 9$ is a negative run whose actual placement is as shown in Figure 3.1. The union of clones 5, 7 and 9 contains all the other clones in the run, so the three negative elementary events $D(c_5, p) = 0, D(c_7, p) = 0$, and $D(c_9, p) = 0$ together imply the event $GOOD(5 - 9, p)$. On the other hand, no two clones in this particular placement cover all of the clones in the run. Thus, in this case, $c^-(r, p) = 3$.

Define a **positive elementary event** with respect to I as an event of the form "atomic interval a contains probe p"; such an event implies $D(c, p) = 1$ for every clone c

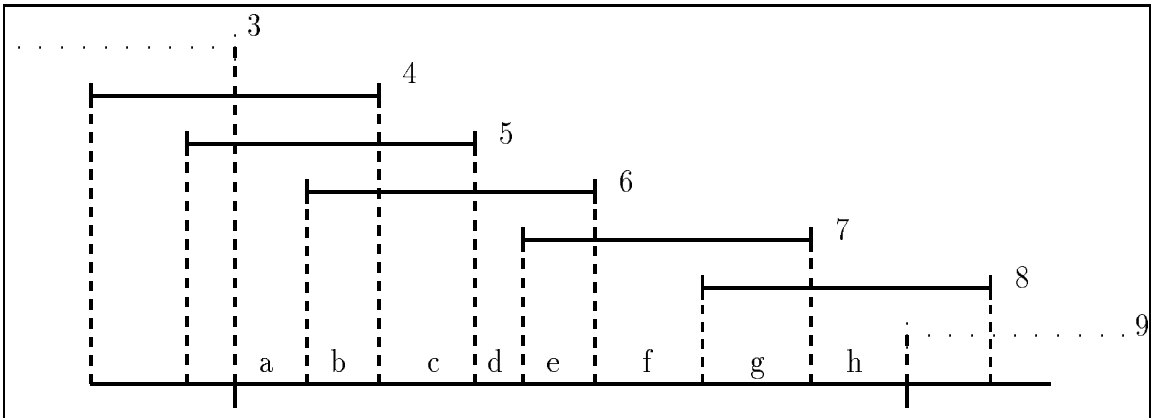

Figure 3.2: Placement of Clones in a Positive Run. The c^+ value of this run is 2, since it is sufficient for probe p to lie on b and g to imply $GOOD(r,p)$.

containing atomic interval a. For a positive run (r,p) define $c^+(r,p)$ to be the minimum number of positive elementary events with respect to I needed to imply $GOOD(r,p)$. For instance, let $4-8$ be a positive run with respect to probe p, and let its actual placement be as shown in Figure 3.2. Note that the dotted clones 3 and 9 must belong, respectively, to the preceding and succeeding negative runs and therefore probe p does not belong to these two clones. The two events "atomic interval b is marked by p" and "atomic interval g is marked by p" (i.e., $\chi(b,p)=1$ and $\chi(g,p)=1$) are sufficient to mark all the clones in the run. On the other hand, no single atomic interval of this particular interleaving can mark all the clones of the run. Thus, in this case, $c^+(r,p)=2$.

We define $f_D(I)$ to be the weighted sum

$$f_D(I) = \phi \sum c^-(r,p) + \theta \sum c^+(r,p)$$

where ϕ and θ are nonnegative, the first sum is over negative runs, and the second sum is over positive runs. The function $f_D(I)$ is a measure of how much additional information is needed to imply the data D. The lower its value, the simpler and better the interleaving is.

Alternate Definition of $f_D(I)$

The function $f_D(I)$ can also be viewed as a rough approximation to $-\log \Pr[I|D]$ (according to the statistical model of Chapter 2). Thus, the desire to maximize $\Pr[I|D]$

also motivates the goal of minimizing $f_D(I)$. By Bayes' Theorem,

$$\Pr[I|D] = \frac{\Pr[D|I]\Pr[I]}{\Pr[D]}.$$

Since D is fixed, maximizing $\Pr[I|D]$ is equivalent to maximizing $\Pr[D|I]\Pr[I]$. When the number of clones is at all large $\Pr[D|I]$ varies much more among interleavings than does $\Pr[I]$. Thus, as an approximation, we neglect the *a priori* probability of an interleaving and seek to maximize $\Pr[D|I]$ rather than $\Pr[I|D]$. Because $D = \cap_{(r,p)} \Pr[GOOD(r,p)]$ and because the events $GOOD(r,p)$ are nearly independent we approximate $\Pr[D|I]$ by $\prod_{(r,p)} \Pr[GOOD(r,p)|I]$. We seek to maximize this value.

Except when the coverage is very low, the length of a negative run is approximately equal to, but not more than $c^-(r,p)$. Even when the coverage is low, the length is never less than $\lceil \frac{c^-(r,p)}{2} \rceil$. Thus, we use $c^-(r,p)$ as an approximation to the length of run (r,p), and estimate $\Pr[GOOD(r,p)|I]$, the conditional probability that p does not occur in any clone of r, by $e^{-\lambda c^-(r,p)}$ where λ is the hybridization rate.

For positive runs the situation is more involved. The event $GOOD(r,p)$ is a conjunction of some events each of which is a disjunction of positive elementary events. For instance, again consider the positive run $4-8$ in Figure 3.2. The event $GOOD$ for this run and probe p is represented by the boolean expression

$$GOOD(4-8,p) = (a+b)(a+b+c)(b+c+d+e)(e+f+g)(g+h)$$

where a represents the elementary event "probe p is in atomic interval a" and similarly for $b, \ldots, h$. If we make the simplifying assumption that probabilities of the positive elementary events are roughly equal, say to p_+, then $\Pr[GOOD(r,p|I)]$ is a polynomial in p_+ which, for a given value of p_+, can be computed efficiently via a variation of the dynamic programming algorithm in Figure 2.2. The polynomial is of the form

$$\Theta(p_+^{c^+(r,p)}) + O(p_+^{c^+(r,p)+1}).$$

For p_+ small, the value $p_+^{c^+(r,p)}$ is a decent approximation to $\Pr[GOOD(r,p|I)]$. If $p_+ = \exp(-\theta)$ our approximation to $\Pr[D|I]$ is

$$\exp[-(\lambda \sum c^-(r,p) + \theta \sum c^+(r,p))],$$

where the first sum is over negative runs and the second sum is over positive runs. To maximize $\Pr[D|I]$ we need to minimize the exponent $\phi \sum c^-(r,p) + \theta \sum c^+(r,p)$. However, this last quantity is our new friend $f_D(I)$.

3.1.2 Local Improvement Strategies

To approximate $\min_I f_D(I)$ we employ a local improvement strategy similar to the ones commonly used for solving the Traveling Salesperson Problem. We associate with every interleaving I a unique ordering π of the clones, given by the order of occurrence of the left-hand end points in the interleaving I; thus $\pi(i) = j$ if clone c_j occupies the ith position in the left-to-right ordering of the clones. Interleaving I is said to be **compatible** with the permutation π. In outline, the algorithm is as follows.

1. Let $c(\pi) = \min_I f_D(I)$, where I ranges over the interleavings compatible with π. We will show that $c(\pi)$ can be computed efficiently;

2. Define a **neighborhood structure** on the set of permutations of $\{1, \ldots, n\}$. Such a structure may be described by a vertex-transitive graph G whose vertex set is the set of permutations of $\{1, \ldots, n\}$; adjacent vertices in this graph are considered to be neighbors.

3. Choose an initial permutation π_0 and construct a maximal path $(\pi_0, \ldots, \pi_t)$ in G such that, for $i = 0, \ldots, t-1$, $c(\pi_{i+1}) < c(\pi_i)$;

4. Return the interleaving I of minimum cost among interleavings compatible with π_t.

To implement the approach we provide efficient algorithms for computing $c(\pi)$ and for searching through the neighborhood of a given permutation to determine whether a local improvement is possible.

3.1.3 Interleavings as Lattice Paths

Before describing the algorithms for computing $f_D(I)$ and $c(\pi)$, we introduce a representation of interleavings that will be used throughout the chapter. We exploit the property that all clones have the same length and thus no clone properly includes another.

Let $M_{n,n}$ be an $n \times n$ array of cells. Define a **lattice path** as a path from the $(1,1)$-cell to the (n,n)-cell in which each edge passes from a cell to either its right neighbor or its lower neighbor, and every cell in the path is of the form (i,j) where $j \geq i - 1$; i.e., the path never ventures more than one diagonal below the main diagonal. There is a natural one-to-one correspondence between lattice paths and the interleavings compatible with permutation π. Each cell visited by a lattice path corresponds to an atomic interval.

Cells below the main diagonal correspond to gaps. A cell (i, j) on or above the main diagonal corresponds to an interval in which clones $c_{\pi(i)}, c_{\pi(i+1)}, \ldots, c_{\pi(j)}$ are active but no other clones are. For $j = 2, \ldots, n$, a move to the right in which the path enters column j corresponds to the beginning of clone $c_{\pi(j)}$; for $i = 1, \ldots, n-1$, a downward move in which the path leaves row i corresponds to the end of clone $c_{\pi(i)}$. Note that no cell of a lattice path is strictly up and to the right from another; i.e., a lattice path cannot include cells (i, j) and (i', j') such that $i' < i$ and $j' > j$.

The data D establishes that certain of the interleavings compatible with π are impossible. Suppose there is a probe p that occurs neither on clone $c_{\pi(i)}$ nor on clone $c_{\pi(j)}$ but does occur on some clone $c_{\pi(k)}$, where $i < k < j$. Then $c_{\pi(i)}$ and $c_{\pi(j)}$ cannot overlap in any feasible interleaving compatible with π; for, if they did, their union would cover $c_{\pi(k)}$, and one of them would have to contain p. Thus we can exclude all lattice paths that pass strictly up and to the right from the cell $(i+1, j-1)$, since every such cell implies that $c_{\pi(i)}$ overlaps $c_{\pi(j)}$. Such a cell will be called an **excluded cell**.

Theorem 3.1 *A lattice path corresponds to an interleaving consistent with the data D if and only if it does not pass through any excluded cells.*

Proof: We have already described why a lattice path cannot pass through any excluded cell. We must show that a lattice path that does not pass through any excluded cell is consistent with the data D. That is, we must show that there exists a valid marking for the interleaving corresponding to the lattice path.

Assume without loss of generality that the clones are numbered so that $\pi(i) = i$. Take any instance where $D(c_i, p) = 1$ and consider a cell (k, l) on the lattice path such that $k \leq i \leq l$. If clones $c_k, \ldots, c_l$ all hybridize to p then it is consistent with D to hypothesize that p hybridizes to the atomic interval corresponding to the cell (k, l).

On the other hand, suppose that there is some clone in the atomic interval which does not contain probe p. Because (k, l) was not excluded, all the clones containing this atomic interval which do not contain p must be either to the right or left of clone c_i. Assume without loss of generality that they are to the left of clone c_i and let clone $c_{k'}$ be the leftmost clone containing this atomic interval that hybridizes to p. The first cell of the lattice path in row k' corresponds to an atomic interval in which all the clones contain probe p. Furthermore, c_i is active over this interval. We hypothesize that p hybridizes to this atomic interval.

By placing the probes in atomic intervals found this way, an occurrence of probe p is placed in an atomic interval contained by clone c_i if and only if $D(c_i, p) = 1$. ■

3.1.4 The Maximum Interleaving

Define the **maximum interleaving compatible with** π as the interleaving in which clones $c_{\pi(i)}$ and $c_{\pi(j)}$ intersect if and only if their intersection is not excluded. That is, they overlap if and only if there is no positive run r whose set of clones is contained in $\{c_{\pi(i+1)}, c_{\pi(i+2)}, \ldots, c_{\pi(j-1)}\}$. The lattice path corresponding to this interleaving has on or below it all those cells that are not excluded. In Section 3.1.5 we shall argue that for any permutation π the maximum interleaving compatible with π minimizes $f_D(I)$ over all interleavings compatible with π. First we will describe how to efficiently find the maximum interleaving.

Theorem 3.2 *For given D and π, the maximum interleaving compatible with π can be computed using time and space $O(n + \sum_{c \in C} \sum_{p \in P} D(c, p))$.*

Proof: The algorithm in Figure 3.3 computes *positive_run_list*, a list of all the positive runs for every probe. The algorithm assumes that the hybridization data is presented in an ordered array of lists, where each element i in the array is a list of the probes hybridizing to clone $c_{\pi(i)}$. It is assumed that the probe lists have been sorted in some fixed but arbitrary order. Let $\pi(0) = 0$ and $\pi(n+1) = n+1$ for the first and last clones, c_0 and c_{n+1}, which are **dummy clones** containing no probes. There is a **dummy probe**, highest in the sorting order, at the end of each probe list.

We now calculate the excluded region. Recall that pair (i, j) denotes a (maximal) positive run from clone $c_{\pi(i)}$ to clone $c_{\pi(j)}$. If $i \neq 1$ and $j \neq n$, then the pair (i, j) marks an excluded region in the lattice path matrix; i.e., cell $(i - 1, j + 1)$ and all cells above and to the right of it are excluded. We represent the excluded region in an array of size n, which we call E. Each element of E represents a column in the lattice path matrix. The value of an element of E is the row number which lies on the perimeter within the excluded region.

First we mark all the bottom left endpoints of local excluded regions contributed by positive runs. This takes time $O(\sum_{c \in C} \sum_{p \in P} D(c, p))$. Then we construct the entire excluded region in time $O(n)$. See Figures 3.5 and 3.4.

The lattice path which borders the perimeter of the excluded region represents the maximum interleaving. The path starts in the upper left cell. We construct the path

```
for i = 1 to n + 1 do
   p1 ← head(probe_list[c_π(i−1)])
   p2 ← head(probe_list[c_π(i)])
   while ((p1 ≠ dummy_probe) or (p2 ≠ dummy_probe)) do
      if (p2 < p1)                          /* A new positive run has begun */
         probe_run[p2] ← i                  /* The start of the run */
         p2 ← next(probe_list[c_π(i)])
      else if (p1 < p2)                     /* A positive run has ended */
         add (probe_run[p1], i − 1) to positive_run_list
         p1 ← next(probe_list[c_π(i−1)])
      else                                  /* In the middle of a positive run */
         p1 ← next(probe_list[c_π(i−1)])
         p2 ← next(probe_list[c_π(i)])
      endif
   endwhile
endfor
```

Figure 3.3: An Algorithm to Compute Positive Runs.

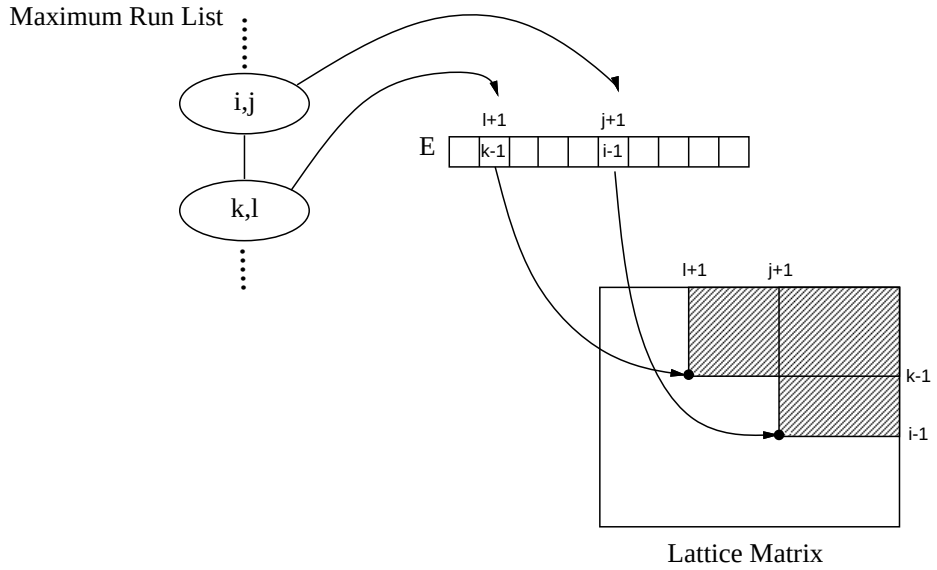

Figure 3.4: The Relation between *positive_run_list*, E, and the Lattice Path Matrix.

```
for i = 1 to n do
   E[i] ← 0
endfor
for each run (i, j) in positive_run_list such that i ≠ 1 and j ≠ n do
   if (E[j + 1] < i − 1) then
      E[j + 1] ← i − 1
   endif
endfor
for i = 1 to n − 1 do
   if (E[i] > E[i + 1]) then
      E[i + 1] ← E[i]
endfor
```

Figure 3.5: Computing the Excluded Region

so that before it enters column j it is in row $E[j]+1$. The path finally exits out of the bottom right cell. The path is constructed in time $O(n)$. Hence the entire running time of the algorithm is $O(n + \sum_{c \in C} \sum_{p \in P} D(c,p))$. ■

Example 3.1 *In Figure 3.6 an example with clone set* $C = \{1, \ldots, 7\}$ *and probe set* $P = \{A, B, C, D, E, F\}$ *is shown together with the data positive and negative runs and corresponding maximum interleaving.*

Consider a fixed placement $\vec{x}$ of the clones and let I be the interleaving associated with this placement. Let π be the order in which the clones occur in $\vec{x}$ (and I). Suppose there are m probes, each occurring in accordance with a Poisson process of rate λ as described in Chapter 2. Let D be the incidence data between the clones and the probes. For the given data D, let I^* be the maximum interleaving compatible with π. We shall argue that I^*, which can be computed from π and D, is likely to resemble the true interleaving I: Any two clones c_i and c_j that overlap in $\vec{x}$ (and hence in I) will also overlap in I^*. If c_i and c_j are disjoint in $\vec{x}$ (and I) and are not consecutive in the ordering π then they will be disjoint in I^* if and only if some clone that lies between c_i and c_j in π contains a probe that occurs in neither c_i nor c_j. In this case the probability that c_i and c_j will overlap in I^* even though they are actually disjoint, goes to zero exponentially with m. On the other hand, if c_i and c_j are consecutive in the ordering then they will overlap in I^*, even if there is actually a gap between them. Thus, as m grows, I^* will eventually become identical to I, except that

	A	B	C	D	E	F	G
1	0	1	0	1	0	0	1
2	0	1	1	0	0	1	0
3	1	1	1	0	0	0	0
4	0	1	1	1	0	0	1
5	1	1	0	0	1	1	1
6	0	0	0	1	1	0	0
7	1	0	0	1	0	1	1

	Positive Runs	Negative Runs
A	3, 5, 7	12, 4, 6
B	12345	67
C	234	1, 567
D	1, 4, 67	23, 5
E	56	1234, 7
F	2, 5, 7	1, 34, 6
G	1, 45, 7	23, 6

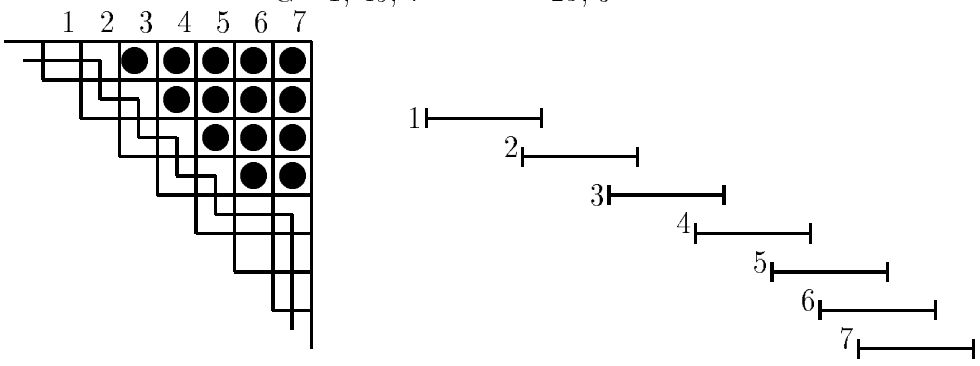

Figure 3.6: The Lattice Path Corresponding to the Maximum Interleaving.

the gaps in I are "squeezed out" in I^*.

3.1.5 Computing $\min_I f_D(I)$

We show in the following theorem and it's corollary that the maximum interleaving compatible with a permutation of the clones π minimizes $f_D(I)$ over all interleavings compatible with π.

Theorem 3.3 *Let I^* be the maximum interleaving with respect to π and D. There is a set of elementary events S^* relative to I^* such that (S^*, I^*) implies D, and, for any interleaving I compatible with π and D, and any set S of elementary events relative to I such that (S, I) implies D, S contains at least as many positive elementary events as S^* does and S contains at least as many negative elementary events as S^* does. In other words, $\sum c^+$ and $\sum c^-$ are minimized in I^*.*

Proof: Define S_I^- and S_I^+ to be minimum sets of negative and positive elementary events for an interleaving I, that imply the data D. Let I be any interleaving compatible with π and D. We want to prove that $|S_{I^*}^+| \leq |S_I^+|$ and $|S_{I^*}^-| \leq |S_I^-|$ for all interleavings I compatible with π and D.

$S_{I^*}^-$: Every pair of clones that overlaps according to I will overlap according to I^*. Thus for the negative run r and probe p, the negative events of S_I^- will imply $GOOD(r, p|I^*)$. The events will not force a clone that should contain a probe, to be probe-free because I^* does not pass through any excluded cells. We conclude that $|S_I^-| \geq |S_{I^*}^-|$.

$S_{I^*}^+$: We will show that for every positive elementary event in S_I^+, "atomic interval a contains probe p," there is a corresponding positive elementary event in $S_{I^*}^+$, "atomic interval a' contains probe p," where the set of clones containing a' is a superset of the set of clones containing a and does not include any clone that does not hybridize to p. The proof similar to that given for Theorem 3.1.

To see that such a a' exists, first consider any atomic interval a^* in I^* which contains all the clones of a. There must be such a a^* because I^* is maximum. Suppose that a^* cannot contain probe p because it includes a clone which does not contain p. Since I^* does not pass through any excluded cell, the clones in a^* which do not contain probe p must be on one side or the other of the clones of a.

Assume without loss of generality that the clones in a^* which do not contain probe p are to the left of the clones of a. Let c_k be the rightmost clone containing a, that does not

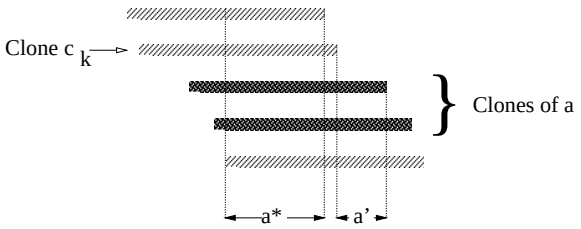

Figure 3.7: A Valid Marking for I^*. c_k, a', a^*, and the clones of a in I^*

contain p (See Figure 3.7). Moving to the right from a^*, we claim we can chose a' to be the first atomic interval which does not contain clone c_k. The atomic interval a' includes all the clones of a and no clones which do not contain probe p. If a' did include a clone which did not contain probe p, it would have to be a clone to the right of a. Then the atomic interval to the left of a' would have been forbidden, for it would have included clones on either side of a which did not contain probe p.

Since every positive elementary event in I^* can be covered by one positive elementary event in I, $S^+_{I^*}$ is no larger than S^+_I. ■

Corollary 3.1 *$f_D(I^*)$ is the minimum, over all I compatible with π of $f_D(I)$ for any nonnegative values of ϕ and θ.*

Proof: c^+ and c^- are both minimized in I^*. ■

Theorem 3.4 *The problem of finding an interleaving that minimizes $f_D(I)$, for arbitrary values of ϕ and θ over all I is NP-hard.*

Proof: It suffices to prove the theorem for $\phi = 0$ and $\theta = 1$. The proof is by reduction from the special case of the Hamiltonian Path Problem where each vertex has degree at least 3. The vertices map to clones, and the edges map to probes. We use the edge incidence matrix D from $G = (V, E)$ as our data. D is an n by m matrix, where $n = |V|$ and $m = |E|$. $D(c_j, p_i) = 1$ and $D(c_k, p_i) = 1$ for the ith edge $(j, k) \in E$. All other entries in column i are 0. D corresponds to a special case of data in which each probe is incident to a unique set of two clones. Thus each clone can have a non-empty overlap with at most two other clones. The sum $\sum_{I^*} c^+$ is minimized over all interleavings when every clone except the

first and last has a non-empty overlap with two other clones. This is precisely the case in which there is one connected component of clones, and thus the graph corresponding to the interleaving has a Hamiltonian path. We conclude that $\sum_{I^*} c^+ \geq (m-1) + 2(n-m+1)$, and there is one connected component of clones if and only if equality holds. ■

Theorem 3.5 *Given interleaving I and data D, one can compute $f_D(I)$ in time $O(n + \sum_{p \in P} \sum_{c \in C} D(c,p))$.*

The algorithm works separately on each run with respect to π. For each negative run (r,p) it places in S the minimum number of negative elementary events needed to imply that no clone in r contains p. For each positive run, it places in S the minimum number of positive elementary events consistent with D needed to imply that each clone in r contains p. In both the positive and negative cases, dynamic programming is used to construct these sets. We omit the details.

Let $c(\pi)$ be the minimum, over all interleavings I compatible with π, of $f_D(I)$. By Corollary 3.1, $c(\pi) = f_D(I^*)$. Combining Theorems 3.2 and 3.5 we find that $c(\pi)$ can be computed in time $O(n + \sum_{p \in P} \sum_{c \in C} D(c,p))$.

3.1.6 Computing $\min_\pi c(\pi)$

Our algorithmic approach to minimizing $c(\pi)$ resembles the 2-opt, 3-opt and Lin-Kernighan local improvement algorithms that are widely used to solve the symmetric Traveling Salesperson Problem ([LK73]). Our algorithms require a neighborhood structure on the set of permutations of $\{1, \ldots, n\}$. Some possible choices are:

The 2-opt neighborhood structure: Permutations π and σ are neighbors if and only if σ can be obtained from π by subdividing π into three parts and reversing the middle part; i.e., if and only if π can be written as the concatenation of α, β and γ, and σ as the concatenation of α, β^R and γ, where β^R is the reversal of β.

the 3-opt neighborhood structure: Permutations π and σ are neighbors if and only if σ can be obtained from π by subdividing π into four parts, possibly reversing one or both of the middle parts, and possibly interchanging the two middle parts.

The local improvement procedure starts with a permutation π_0; at a general step, given a permutation π_i, it searches for a permutation π_{i+1} such that π_i and π_{i+1} are neigh-

bors and $c(\pi_{i+1}) < c(\pi_i)$. The computation terminates at a local minimum when no such neighbor exists.

The main challenge in implementing our local improvement procedures is to search efficiently through the neighborhood of the present permutation, implicitly or explicitly computing the cost of each neighbor. A key observation is that $c(\pi)$ is a sum of independent terms corresponding to the runs associated with π. Thus, in computing $c(\sigma)$ knowing $c(\pi)$, it is sufficient to add in a term for each run created, and subtract out a term for each run destroyed, in moving from σ to π.

3.2 A Simple Approach Based on Hamming Distance

In this section we show that, when m, the number of probes, is sufficiently large, a simple approach based on Hamming distance of clones yields the true ordering of the clones with high probability. Let $A = [a_{ij}]$ be the **probe distance matrix**, that is a_{ij} is the Hamming distance between row i and row j of data D:

$$a_{ij} = \sum_{p \in P} (D(c_i, p) + D(c_j, p) \pmod 2).$$

Also let $A^* = [a^*_{ij}]$ be the **interval distance matrix**, so that a^*_{ij} is the length of the symmetric difference between clones, viewed as intervals of unit length on the real line:

$$a^*_{ij} = \min(2, |x_j - x_i| + |(x_j + 1) - (x_i + 1)|) = 2\min(1, |x_j - x_i|) = 2 - 2x_{ij}$$

where x_i is the left endpoint of clone c_i and x_{ij} is the length of the overlap of clones c_i and c_j. We also assume that there are no gaps in the placement; i.e., the set of intervals representing clones cover the original piece of DNA.

Let τ be the true permutation of clones; i.e., the order in which they appear in the placement. One rough approximation to τ is the so called **greedy** permutation γ, which is constructed as follows: First the two clones among all pairs of clones that are closest to each other (in the hamming metric) are identified and put together. (Throughout, ties are broken arbitrarily.) Then, recursively, having constructed the partial sequence of clones $c_i, \ldots, c_j$, the closest clone to either c_i or c_j is identified and attached to c_i or c_j, whichever is closer.

An alternative approximation to τ is ξ, the permutation induced by the shortest Hamiltonian cycle with respect to the Hamming metric. We add a fictitious empty clone

which hybridizes to no probes to the set of clones. Then we construct a complete graph whose distance matrix is given by A (with an additional row and column corresponding to the empty clone.) After solving the Traveling Salesperson Problem on A we obtain a cyclic ordering of clones. Then ξ is the permutation obtained from this ordering by removing the empty clone and listing clones starting from the one right after the empty clone and ending with the one right before it. It turns out that ξ, herein referred to as the TSP permutation, minimizes the total number of positive runs.

Theorem 3.6 *Let D be the data with an additional empty clone added. Suppose ξ is the permutation induced by the minimum length Hamiltonian cycle in the complete graph whose vertices are labeled by clones (including the empty clone) and whose edges $\{i, j\}$ are labeled by the Hamming distance between rows i and j of D. If the rows of D are arranged according to ξ, with the empty clone at the top, then the number of positive runs in D is minimum over all other arrangements of rows of D.*

Proof: Let the rows of D be arranged according to some permutation π starting with the empty clone. If for each probe p we walk from the first row through all rows and then go back to the first row, then the number of transitions from 0 to 1 or from 1 to 0 is exactly twice the number of positive runs for probe p. But each transition from 1 to 0 or 0 to 1 increases the Hamming length of the cycle induced by π by one. Therefore, the total number of positive runs over all probes is exactly half the length of the Hamiltonian cycle corresponding to π; in particular, the minimum length Hamiltonian cycle corresponds to a permutation that minimizes the number of positive runs. ■

Notice that if instead of an empty clone we had added the universal clone, a fictitious clone containing all probes, then the TSP permutation would have minimized the number of negative runs.

Now we argue that, as the number of probes gets very large, both the greedy permutation γ and the TSP permutation ξ are likely to be identical to the true permutation τ. (Recall that we do not distinguish between a permutation and its reverse.) The following lemma shows that for the interval distance matrix A^* the greedy and TSP permutations are equal to the true permutation.

Lemma 3.1 *Let A^* be the interval distance matrix corresponding to a placement $\vec{x}$, of unit length clones $\{c_1, \ldots, c_n\}$, that contains no gaps, and let γ^* and ξ^* be the greedy and the*

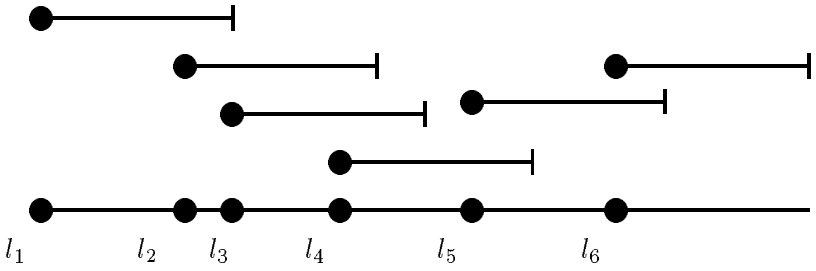

Figure 3.8: Mapping the Interval Distance to 1-Dimensional Euclidean Distance.

TSP permutations obtained from A^*. Then γ^*, ξ^* and the true permutation τ are identical (up to reversal).

Proof: Recall that $a^*_{ij} = 2\min(1, |x_i - x_j|)$. Therefore, the problem of finding the shortest Hamiltonian path based on A^* is equivalent to the special case of the Euclidean Traveling Salesperson Problem in which all cities are on a line (that is the one-dimensional Euclidean Traveling Salesperson Problem; see Figure 3.8.) But in this case the problem is trivial: since there are no gaps the optimal path is the one that starts from one of the two extreme cities and goes through all cities as they appear on the line. The greedy algorithm also finds this path from A^*. Therefore, up to reversal $\gamma^* = \xi^* = \tau$. ■

Since we do not know the original placement of clones, we we do not know A^*. However, observe that in order to guarantee that both the greedy and TSP permutations are identical to the true permutation it suffices to have any matrix A with the property that for all indices i, j, k, l, $a_{ij} > a_{kl}$ if and only if $a^*_{ij} > a^*_{kl}$. Thus if we can show that for sufficiently large m the probe distance matrix must have this property with high probability, then we have also shown that the greedy and TSP permutations are equal to the true permutation with high probability.

Theorem 3.7 *Let A^* be the interval distance matrix corresponding to a placement $\vec{x}$, of unit length clones $\{c_1, \ldots, c_n\}$, that contains no gaps. Suppose P is a set of m probes distributed independently and according to a Poisson process of rate λ, and $A^{(m)}$ the corresponding probe distance matrix. For each pair of indices ij and kl if $a^*_{ij} < a^*_{kl}$, then as $m \to \infty$,*

$$\Pr\left[a^{(m)}_{ij} > a^{(m)}_{kl}\right] \to 0$$

Proof: For a pair of clones c_i and c_j with an overlap of length x_{ij}, a probe p contributes the their Hamming distance if and only if p appears in exactly one of the two clones. The probability of this event is given by

$$p_{ij} = 2e^{-\lambda}(1 - e^{-\lambda(1-x_{ij})}).$$

The mutually independent events that probes $\{p_1, p_2, \ldots\}$ contribute 1 unit to the Hamming distance between clones c_i and c_j may be viewed as a sequence of independent Bernoulli trials of probability p_{ij}. Therefore, as $m \to \infty$, by the law of large numbers, for any fixed positive real number ϵ

$$\Pr\left[\left|\frac{a_{ij}^{(m)}}{m} - p_{ij}\right| > \epsilon\right] \to 0$$

But, since $a_{ij}^* = 2 - 2x_{ij}$, $a_{ij}^* < a_{kl}^*$ if and only if $p_{ij} < p_{kl}$, and therefore $\Pr[a_{ij}^{(m)} > a_{kl}^{(m)}]$ must also tend to zero. ■

By the remark preceding Theorem 3.7, as m grows the probability that γ and ξ are equal to each other and to the true permutation tends to 1.

As we argued in the preceding section, once the order of clones is known, the maximum interleaving corresponding to that order is most likely to be the true interleaving. So with sufficiently large m one can solve the most likely interleaving problem rather easily. However, for moderate size m the greedy and TSP permutations are not very close to the true permutation. Nevertheless the greedy permutation is often a good point to start the local search strategies discussed in Section 3.1.6.

3.3 Discussion

In this section we present some computational results based on our experiments with the $c^+ + c^-$ objective function. In the current implementation we set the coefficients ϕ and θ of c^+ and c^- to one. For comparison we also include one example from the greedy and the Traveling Salesperson Problem approaches as discussed in Section 3.2.

The experiments were conducted on simulated, rather than real data. According to the model of Section 2.1, we generated placements for n unit-length clones whose left ends were uniformly and independently distributed over the interval $[0, N-1]$ (thus, the original chromosome is represented by the interval $[0, N]$.) To assign m probes to the n clones according to the Poisson distribution of rate λ, we calculated the lengths of the

$2n-1$ atomic intervals. Then for each atomic interval a of length ℓ, and each probe p, we assigned p to a (and to all clones containing a) with probability $1-e^{-\lambda\ell}$. Labeling clones from left to right by 1,2, ..., n, the true permutation of clones was always $\tau = 12\cdots n$ or the reverse permutation $n\cdots 21$. (As usual, we did not distinguish between a permutation and its reverse). This labeling makes it convenient to compare the true permutation to the permutations generated by our algorithms.

In each experiment we started with the greedy permutation γ, which was generated by the greedy algorithm. We then used γ as the starting point of the 2-opt local search algorithm with the $c^+ + c^-$ objective function.

We generated data for two values of λ, namely $-\ln(0.75) \approx 0.29$ and $-\ln(0.5) \approx 0.69$ corresponding to probabilities 25% and 50%, respectively, that a given probe hybridizes to a clone of unit length. For each λ we generated data for two values of coverage, $c = 5$ and $c = 10$. Throughout we fixed the number of clones at $n = 100$, hence, altogether we experimented on four different placements. For each placement, we varied the number of probes over $\{10, 20, 30, 50, 70\}$. The permutations σ generated by the 2-opt heuristic are shown in Figures 3.10 through 3.13. In each graph the the x-axis corresponds to i and the y-axis corresponds to $\sigma(i)$. Since the true permutation $\tau = 12\cdots n$ (or its reverse), the perfect solution would be represented by either a 45 degree line through the origin or a -45 degree line from point $(1, n)$ to point $(n, 1)$. Therefore, in these graphs long stretches of ± 45 degree line segments indicate the success of the 2-opt heuristic in zeroing in on the true permutation, and scattered short line segments mean a less successful attempt.

Overall, $c^+ + c^-$ seems to be a good objective function, except for small values of m. In Table 3.1 the $c^+ + c^-$ value of the permutation generated by the 2-opt heuristic (c_2) is compared with the $c^+ + c^-$ values of the greedy (c_1) and the true permutations (c_3). Some entries under c_3 columns are marked by "*". This indicates that in those cases, the $c^+ + c^-$ value of 2-opt permutation is smaller than that of the true permutation.

In Figure 3.9 the results of greedy and the Traveling Salesperson Problem algorithms are compared with the $c^+ + c^-$ 2-opt local optimum. For the Traveling Salesperson Problem, a local search based on the Lin-Kernighan heuristic was used. In this single experiment we used 200 clones, 40 probes, Poisson rate $-\ln(0.50)$, and coverage 5. Clearly, for this experiment $c^+ + c^-$ is a superior objective function. However, we have also observed that for large m and higher coverage, the TSP permutation and sometimes even the greedy permutation are already very close to the true permutation. On the other hand for small

		$m = 20$			$m = 30$			$m = 50$			$m = 70$		
λ	c	c_1	c_2	c_3	c_1	c_2	c_3	c_1	c_2	c_3	c_1	c_2	c_3
0.29	2.5	739	630	672*	1224	994	1040*	2184	1756	1793*	3127	2506	2552*
	5.0	528	434	419	710	625	631*	1280	1066	1066	1799	1543	1543
	10.0	280	234	234	393	342	342	705	597	597	920	823	823
0.69	2.5	774	623	713*	1214	1014	1077*	2130	1778	1828*	3223	2560	2587*
	5.0	564	469	432	860	672	669	1226	1131	1133*	1689	1671	1627
	10.0	338	254	254	516	394	394	735	712	674	1022	984	935

Table 3.1: Comparing $f_D(I)$ for Greedy, 2-opt, and the True Permuation. $c_1 = f_D(\text{Greedy})$, $c_2 = f_D(\text{2-opt})$, and $c_3 = f_D(\text{True})$.

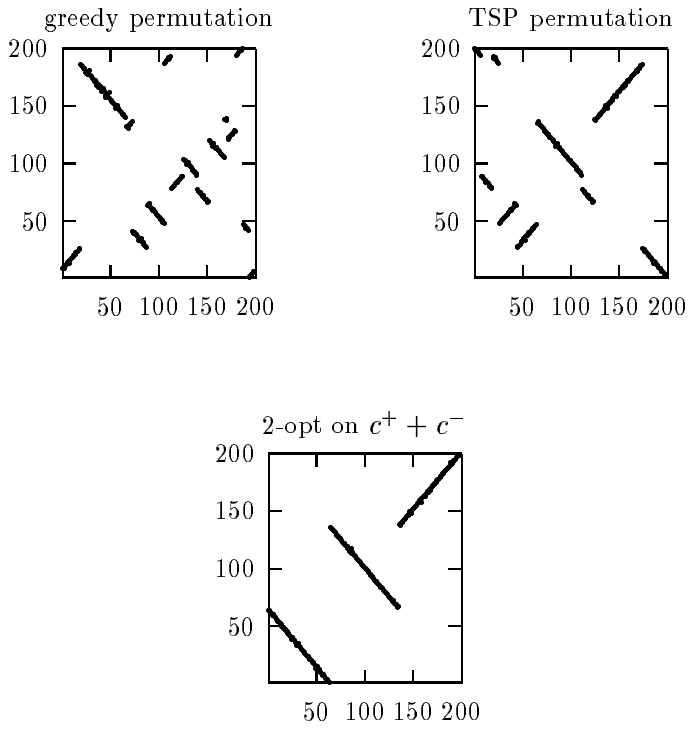

Figure 3.9: Pictorial Comparison of Greedy, TSP, and 2-opt with $f_D(I)$. Run with $\lambda = 0.69$, $c = 5.0$, $n = 200$, and $m = 40$

values of m and low coverage all three permutations are fairly poor.

Our experiments hint at the following phenomenon. For small values of m, there is simply not enough information in the data to reveal the placement of clones, and therefore, all algorithms are expected to perform poorly. For very large m, the argument of Section 3.2 shows that even the most simple-minded method such as the greedy algorithm is likely to find the true permutation. However, it seems that there is a range of values for m, depending on the underlying placement, where the result of the $c^+ + c^-$ objective is superior to those of the greedy or the Traveling Salesperson Problem heuristics.

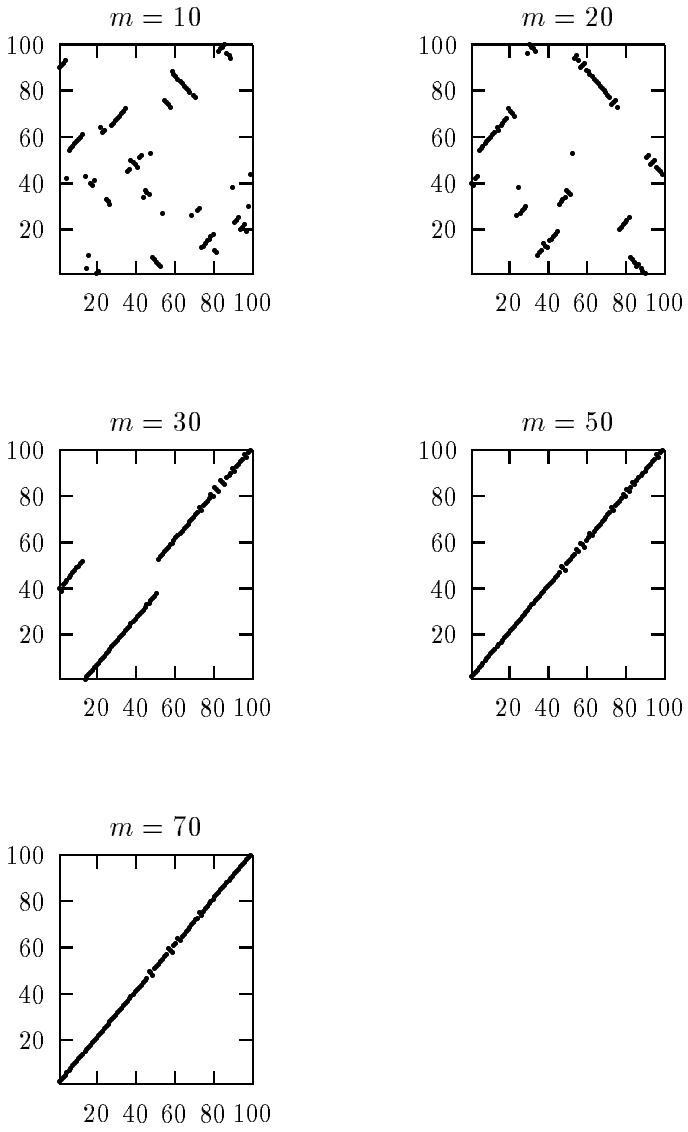

Figure 3.10: Performance of 2-opt with $\lambda = 0.29$, $n = 100$, $c = 5$

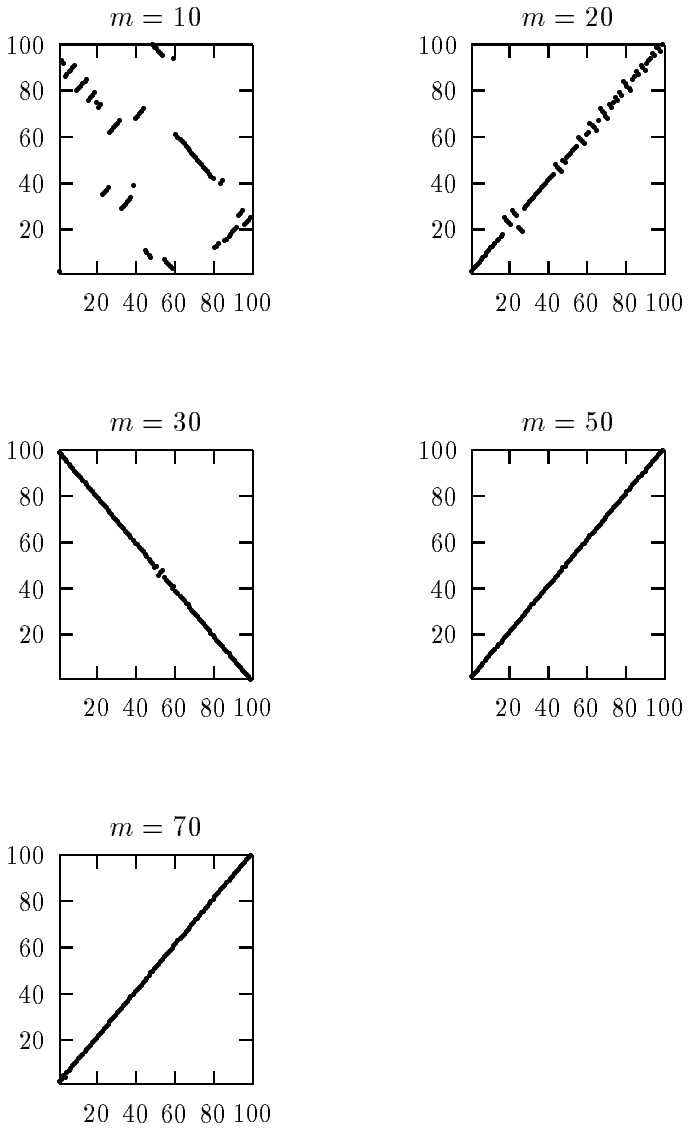

Figure 3.11: Performance of 2-opt with $\lambda = 0.29$, $n = 100$, c=10

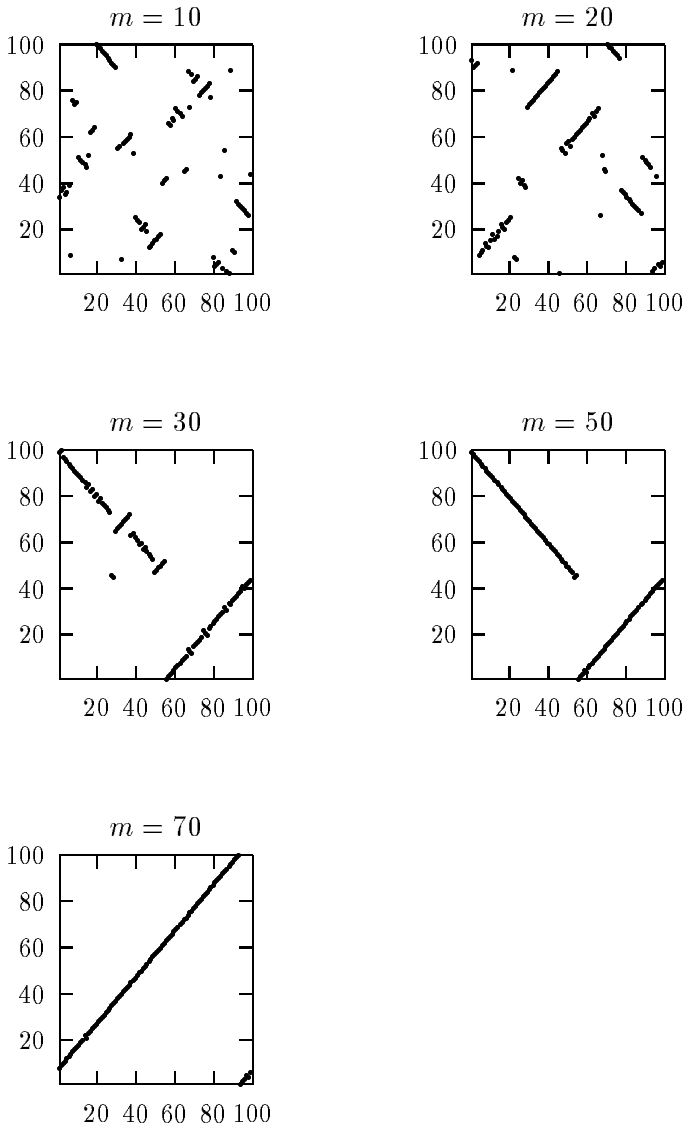

Figure 3.12: Performance of 2-opt with $\lambda = 0.69$, $n = 100$, $c = 5$

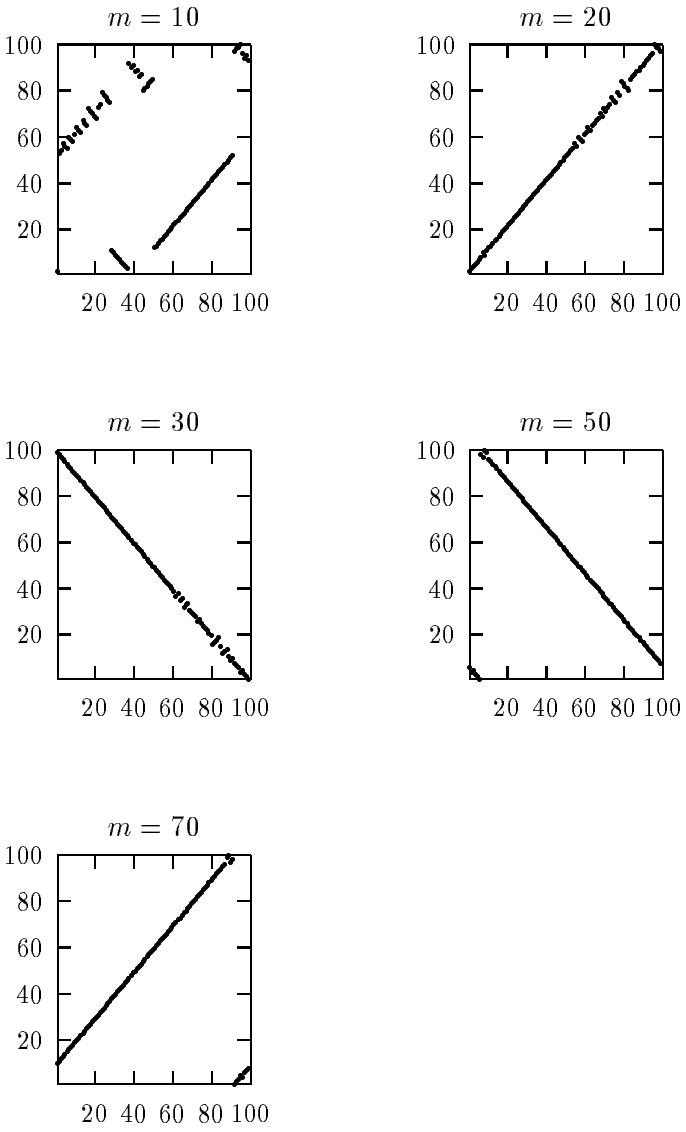

Figure 3.13: Performance of 2-opt with $\lambda = 0.69$, $n = 100$, $c = 10$

Chapter 4

Minimizing Errors

(This material will also appear as [New94a].)

In this chapter we give an algorithm for evaluating a permutation of clones based upon a minimum-error heuristic. Unlike the algorithms of Chapter 3 this algorithm can handle more than trivial amounts of experimental error. It is most appropriate when experimental errors are significant and the statistical model of Chapter 2 does not apply.

4.1 Approach

The approach makes use of the fact that the typical atomic interval is contained by several clones. In a library with coverage c, the expected number of clones which contain a randomly chosen point of the DNA is c. The expected number of clones which contain a randomly chosen atomic interval is approximately c. Thus, in the absence of experimental error a mark which occurs in a typical atomic interval will mark approximately c consecutive clones. This is true except near the ends of the DNA where the number of active clones tapers off; the number of consecutive clones marked by a probe in a run which includes the first or last clone may be significantly less than c.

A probe's data arranged in the order of the permutation of the clones is called its **signature**. The algorithm evaluates a permutation of clones by looking at each probe's signature separately. For each probe a cost is computed and this cost is summed over all the probes. The lower the total cost is, the better the permutation is. When all positive runs for a given probe either contain the first or last clone or contain at least c clones, the probe will be called **proper** and it will contribute a cost of zero to the permutation. That

is, a probe will be proper if its signature satisfies the regular expression

$$S \stackrel{\text{def}}{=} 1^*(0^*1^c1^*0^*)^*1^*$$

where α^* means zero or more occurrences of a regular expression α and β^c means exactly c occurrences of a regular expression β. (See, for instance, [HU79] for a discussion of regular expressions.)

A probe which has an improper signature σ will contribute a positive cost in the evaluation of a permutation of clones. It is postulated that σ should have been proper and that its failure be so must be the result of false positives and/or false negatives in the hybridization data. We assess a penalty θ for each false positive and a penalty ϕ for each false negative. Both θ and ϕ are nonnegative; at least one is positive. The cost to transform σ to a proper signature τ is defined to be

$$cost_\tau(\sigma) \stackrel{\text{def}}{=} \theta P_\tau(\sigma) + \phi N_\tau(\sigma)$$

where $P_\tau(\sigma)$ is the number of 1's in σ that are 0's in τ (i.e., the number of false positives in σ) and $N_\tau(\sigma)$ is the number of 0's in σ that are 1's in τ (i.e., the number of false negatives in σ). The cost that σ contributes to the permutation of clones is

$$cost(\sigma) \stackrel{\text{def}}{=} \min_{\tau \in \{\text{proper signatures}\}} cost_\tau(\sigma).$$

That is, the cost of σ is the "distance" to a "nearest" proper signature.

4.2 Algorithm

The cost of a signature can be computed using dynamic programming in time $O(nc)$ where n is the number of clones and c is the coverage specified in the definition of S. The computation proceeds via a 2-dimensional array *run*.

For a signature σ, let $\sigma(i)$ be the datum for the ith clone and let σ_i be the prefix which is the signature for the first i clones. The 2-dimensional array *run* contains some cost-like values for each of the prefixes $\{\sigma_i : 0 \leq i \leq n\}$ of a signature σ. The values are cost-like in that each is the "distance" of a σ_i to a "nearest" of a restricted set of proper signatures.

More precisely (in the general case when $i > 0$ and $j \leq i$) the definition of $\mathrm{run}(i,j)$ is

$$\mathrm{run}(i,j) \stackrel{\mathrm{def}}{=} \begin{cases} \text{The cost for } \sigma_i \text{ minimized over all proper} \\ \tau_i \text{ ending with one or more zeros} & \text{if } j = 0, \\ \text{The cost for } \sigma_i \text{ minimized over all proper} \\ \tau_i \text{ ending with exactly } j \text{ ones} & \text{if } 0 < j < c, \text{ and} \\ \text{The cost for } \sigma_i \text{ minimized over all proper} \\ \tau_i \text{ ending with } c \text{ or more ones} & \text{if } j = c. \end{cases}$$

These values are useful for two reasons. Firstly, $cost(\sigma)$ is a minimum over all τ regardless of how they end and thus it is easy to compute; $cost(\sigma) = \min_{0 \leq j \leq c} \mathrm{run}(n,j)$. Secondly, as it will be shown, it is easy to calculate each $\mathrm{run}(i,\cdot)$ value from the $\mathrm{run}(i-1,\cdot)$ values.

When $\sigma(i) = 0$ (i.e., the probe does not hybridize to the ith clone) it must be considered either a true negative or a false negative. A true negative arises when $\tau(i) = 0$. In such a case, since τ is proper, τ_{i-1} must end with a 0 or c or more 1's. A false negative arises when $\tau(i) = 1$. In such a case a penalty must be paid and τ_i ends with one more 1 than τ_{i-1} does. In summary, when $\sigma(i) = 0$

$$\mathrm{run}(i,j) = \begin{cases} \min(\mathrm{run}(i-1,0), \mathrm{run}(i-1,c)) & \text{if } j = 0, \\ \mathrm{run}(i-1,j-1) + \phi & \text{if } 0 < j < c, \\ \min(\mathrm{run}(i-1,c-1), \mathrm{run}(i-1,c)) + \phi & \text{if } j = c. \end{cases}$$

Similarly, when the probe does hybridize to the ith clone (i.e., when $\sigma(i) = 1$), $\mathrm{run}(i,j)$ satisfies

$$\mathrm{run}(i,j) = \begin{cases} \min(\mathrm{run}(i-1,0), \mathrm{run}(i-1,c)) + \theta & \text{if } j = 0, \\ \mathrm{run}(i-1,j-1) & \text{if } 0 < j < c, \\ \min(\mathrm{run}(i-1,c-1), \mathrm{run}(i-1,c)) & \text{if } j = c. \end{cases}$$

Because a proper τ may start with any number of 1's, in the degenerate cases $\mathrm{run}(i,j)$ is defined by:

$$\mathrm{run}(i,j) \stackrel{\mathrm{def}}{=} \begin{cases} 0 & \text{if } i = 0 \text{ and} \\ \mathrm{run}(i,i) & \text{if } j > i > 0. \end{cases}$$

An implementation of the algorithm is given in Figure 4.1. Note that moving the memory allocation outside of this subroutine will be more efficient if the algorithm is called more than once. Also note that the clones are numbered from 0 to $n-1$ rather than from 1 to n.

```
double signature_cost(
   int      n,                          /* The number of clones */
   int      c,                          /* The coverage of the clones */
   double   pospenalty,                 /* The penalty for a false positive */
   double   negpenalty,                 /* The penalty for a false negative */
   int      signature[ ])               /* The probe's signature */
{
/*
run[j] tells how much it would cost to have j 1's in a row up to the
current clone. When j == c it means j or more. When j == 0 it
tell's the cost for one or more 0's in a row up to the current clone.
*/
   double *run;
   double   runZero;                    /* Temporary storage for run[0] */
   int      i;                          /* Index into the signature array */
   int      j;                          /* Index into the run array */
   double   cost;                       /* The cost of the signature */

   /* Allocate and initialize memory for the run[0..c] array */
   run = (double *) malloc((c + 1) * sizeof(double));
   for (j = 0; j <= c; j++)
        run[j] = 0.0;

   /* Run through the signature computing the new run[ ] values */
   for (i = 0; i < n; i++) {
      if (signature[i] == 0) {          /* no hybridization to this clone*/
         runZero = min(run[0], run[c]);
         run[c] = min(run[c], run[c - 1]) + negpenalty;
         for (j = c - 1; j > 0; j--)
              run[j] = run[j - 1] + negpenalty;
         run[0] = runZero;
      }
      else {                            /* hybridization to this clone */
         runZero = min(run[0], run[c]) + pospenalty;
         run[c] = min(run[c], run[c - 1]);
         for (j = c - 1; j > 0; j--)
              run[j] = run[j - 1];
         run[0] = runZero;
      }
   }

   /* The cost of the signature is the smallest cost in run[ ] */
   cost = run[0];
   for (j = 1; j <= c; j++)
        if (run[j] < cost)
             cost = run[j];

   free(run);                           /* Deallocate the allocated memory */
   return cost;
}
```

Figure 4.1: Dynamic Programming Algorithm for Computing the Cost of a Probe Signature. Note that the clones are numbered from 0 to $n - 1$.

4.3 Discussion

This algorithm was implemented to evaluate permutations of clones. The greedy permutation (described in Section 3.2) was used as a starting point and the 2-opt heuristic for the Traveling Salesperson Problem Problem was used (see Section 3.1.6) to search through the space of permutations for a best permutation. For comparison purposes the $(c^+ + c^-)$-based algorithm is also shown. It too used the greedy permutation as a starting point.

The input data was simulated. It had $n = 50$ clones and $m = 60$ probes placed according to the stochastic model presented in Chapter 2. The Poisson rate for the probes was 0.69314718 and the library coverage was 10.

The results are presented in Figures 4.2 and 4.3. The individual graphs show the algorithm used and the fraction of errors in the data. Both false positives and false negatives occurred at the stated rate and the penalties used in the Minimum-Error algorithm for false positives and false negatives were both one. The coordinate pair (x, y) is plotted if in position x of the final permutation appears the clone which should have appeared in the yth position. As discussed in Section 3.3, ± 45 degree line segments indicate the success in zeroing in on the true permutation, and scattered short line segments mean a less successful attempt.

The $c^+ + c^-$ algorithm does not perform well in the presence of more than a trivial amount of error. Observe that the graphs for 2% error or more do not show coherent ± 45 degree straight lines.

The Minimum-Error approach performs well. It is able to handle more error than the simplistic greedy approach. At the 1%-error level the Minimum-Error algorithm is able to clean up a pair of clones transposed in the greedy solution. At the 8%-, 10%-, and even at the 15%-error levels the Minimum-Error algorithm produces more coherent straight lines at ± 45 degrees.

The Minimum-Error approach is simple and quick and should prove to be a valuable tool for finding clone orderings.

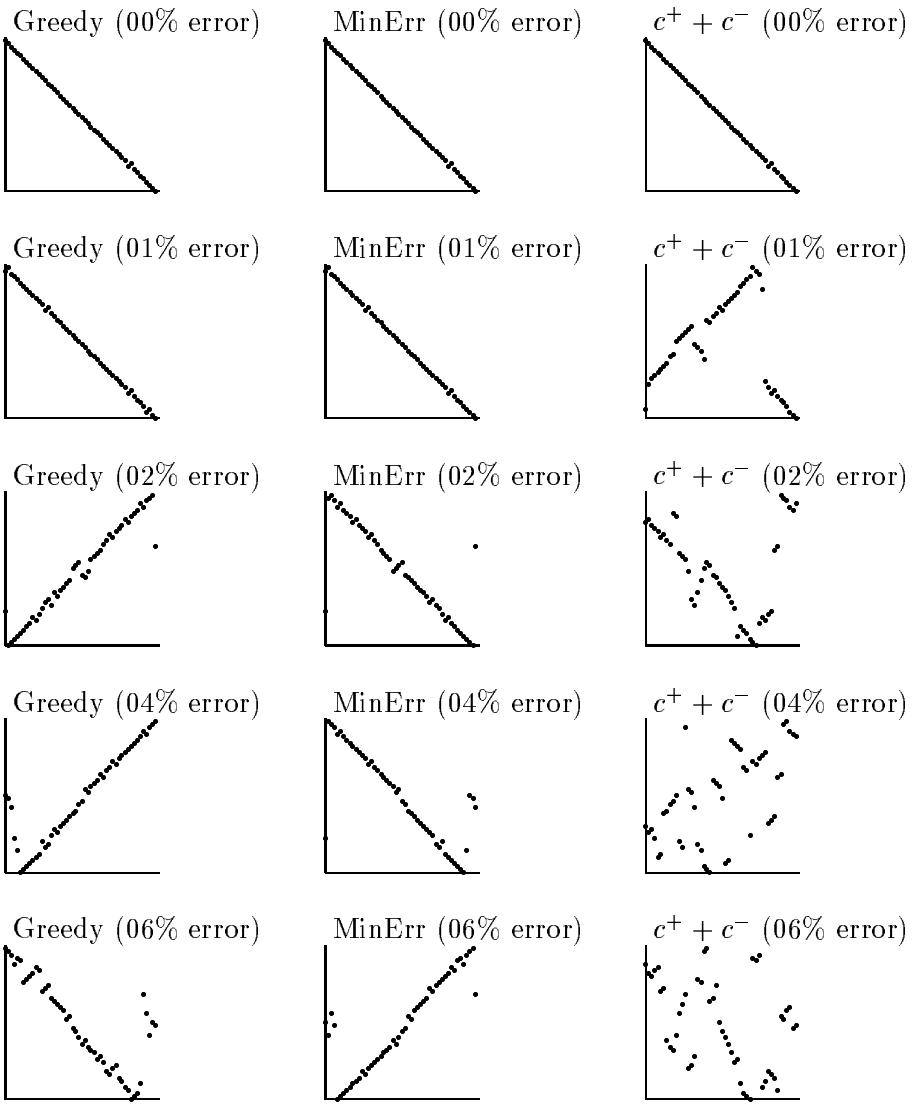

Figure 4.2: The Heuristics on Data with Few Errors

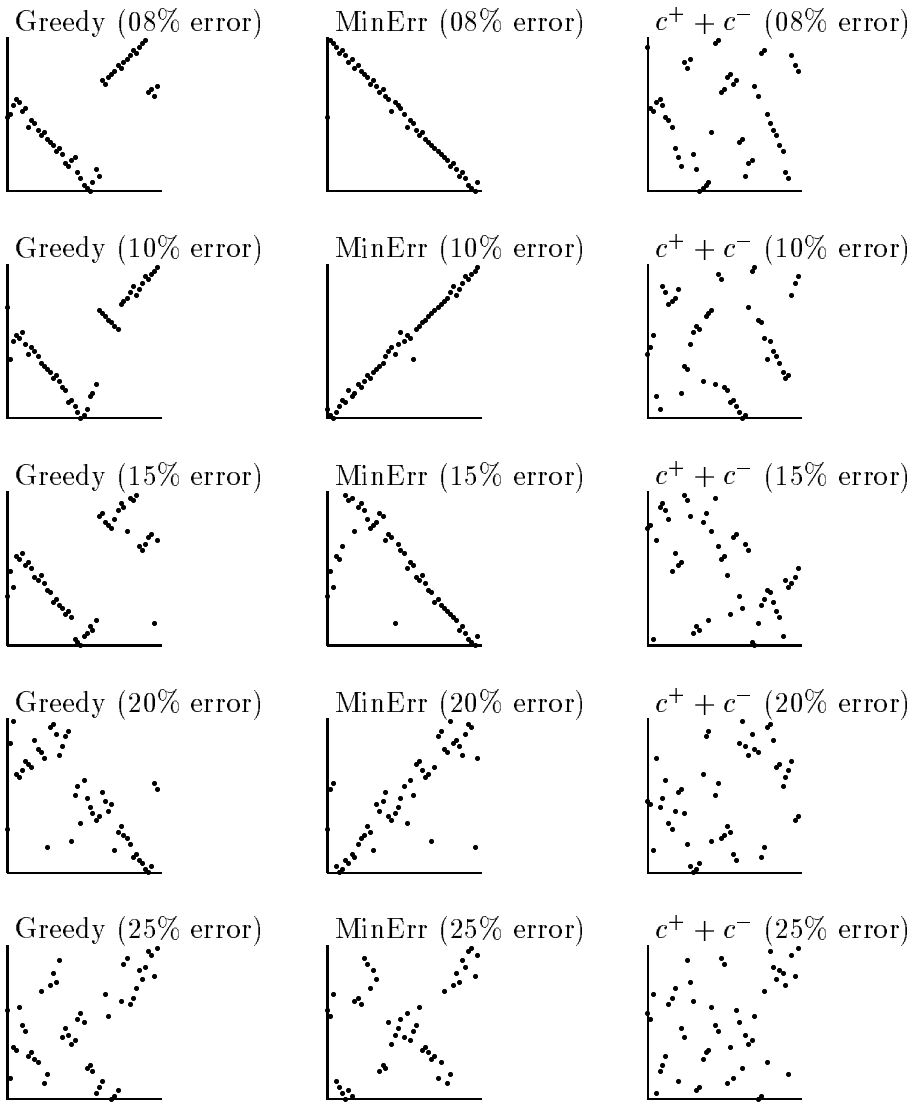

Figure 4.3: The Heuristics on Data with Many Errors

Chapter 5

Counting Clone Orderings

(This material will also appear as [New94c].)

For several computational biology problems, the number of possible solutions is known. Specifically, [SSL90] discusses the number of possible solutions to the Partial Digest Problem, [NN93] discusses it for the Probed Partial Digest Problem, and [SW91] discusses the Double Digest Problem. This article discusses the number of possible clone orderings that might arise in the Clone Ordering Problem.

Previous theoretical work on the Clone Ordering Problem can be found in [LW88, ALTW91]. Algorithms for solving the Clone Ordering Problem can be found in [AKNW93, CAT93, CNH$^+$90, EL89].

Besides being of combinatorial interest, the value of $c(n)$ is useful in the calculation of the the entropy of the Clone Ordering Problem, [Spe91]. In this situation, entropy describes how many binary questions on average must be asked to determine the clone ordering for n clones. The most straightforward approach to this calculation involves integration in n-dimensional space of a piecewise analytic function. The number of pieces is precisely the number of clone orderings.

5.1 What To Count

A clone ordering constructed using this experiment may not be unique. There is no way to know the relative order along the DNA of the islands. Also, there is no way to tell whether a given island is present as shown or if it is present as its left-right (biologists say **sense-antisense**) reflection.

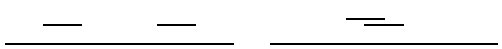

Figure 5.1: The Two Overlap Possibilities for Two Clones — the two clones either overlap or don't overlap.

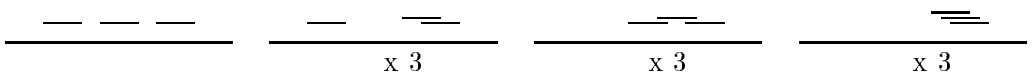

Figure 5.2: The Ten Possibilities For Three Clones. The second, third, and fourth diagrams each have 3 distinguishable ways to label the clones.

We say that two interleavings are **topologically similar** if one can be transformed into the other by permuting the islands and/or reflecting some of the islands. We will denote by $c(n)$ the number of topologically distinct interleavings for n clones under the assumption that no clone includes another. For the remainder of this chapter we will use **interleaving** to mean topologically distinct interleaving.

For small values of n we can enumerate the solutions by hand. For only one clone, there is just one interleaving. With two clones there are two possibilities. The two clones either overlap or they do not overlap. See Figure 5.1.

For three clones there are 10 possibilities. One possibility is that the three clones are mutually nonoverlapping. A second possibility is that all three clones overlap but clone #1 is between clones #2 and #3. A third possibility is that all three clones overlap but clone #2 is between clones #1 and #3, and so on. See Figure 5.2 for all 10 possibilities.

The value of $c(n)$ for the first few values of n is given in Table 5.1. These values were computed using the values for $c(1), c(2),$ and $c(3)$ and the recurrence relation we derive,

$$c(n) = (4n-5)c(n-1) - (4n-7)c(n-2) + (n-2)c(n-3), \qquad n \geq 4.$$

We define an exponential generating function $C(x) = \sum_{n=0}^{\infty} c(n)\frac{x^n}{n!}$ and show that

$$C(x) = \exp\left(\frac{1+2x-\sqrt{1-4x}}{4}\right).$$

n	$c(n)$
0	1
1	1
2	2
3	10
4	94
5	1,286
6	22,876
7	499,612
8	12,925,340
9	386,356,924
10	13,099,953,016
11	496,719,289,496
12	20,825,694,943,912
13	956,599,393,819,720
14	47,772,070,664,027,984
15	2,577,034,852,683,364,816
16	149,335,440,671,982,405,136
17	9,251,650,217,381,166,689,552
18	610,194,993,478,502,245,703,200
19	42,688,019,374,465,782,644,235,424
20	3,157,223,748,264,915,895,381,735,136

Table 5.1: The Number of Topologically Distinct Solutions to the Clone Ordering Problem for n distinguishable equal-length clones with non-coinciding endpoints.

Using Darboux's lemma, $C(x)$ gives us the asymptotic growth of $c(n)$. We show that

$$c(n) \sim \frac{e^{3/8}\sqrt{2}}{8n}\left(\frac{4n}{e}\right)^n.$$

5.2 Combinatorics

We will derive the results using a three step process. In the first step we will assume that the clones are indistinguishable and form one island, and we will ignore the reflection symmetry. In the second step we will correct for distinguishability and the reflection symmetry. In the third step we will allow more than one island.

5.2.1 Step 1

For this first step, assume that the clones are indistinguishable. Also for this step, ignore the reflection symmetry for islands and assume that the clones form one island.

As we did in Section 3.1.3 we will use lattice paths to represent interleavings. Label the clones 1 to n from left to right based upon the relative order of their left endpoints. Label the $2n$ clone endpoints $e_1 \leq \ldots \leq e_{2n}$ from left to right. These endpoints define $2n-1$ atomic intervals $(e_1, e_2), (e_2, e_3), \ldots, (e_{2n-1}, e_{2n})$.

Lemma 5.1 *For any k, the set of clones containing the atomic interval (e_{k-1}, e_k) is $\{i, i+1, \ldots, j\}$ for some $1 \leq i \leq j \leq n$. Futhermore, the set of clones containing (e_k, e_{k+1}) is either $\{i, \ldots, j+1\}$ or $\{i+1, \ldots, j\}$.*

Proof: Because the clones are labeled by the relative order of their left endpoints, the set of clones with left endpoints before (e_{k-1}, e_k) is $\{1, \ldots, j\}$ for some $1 \leq j \leq n$. Because the clones are equilength, the order of their right endpoints is the same as the order of their left endpoints. Thus, the set of clones with right endpoints before (e_{k-1}, e_k) is $\{1, \ldots, i-1\}$ for some $0 \leq i-1 \leq j$ where $i = 1$ corresponds to the empty set. Because the island is connected, $i-1$ must be strictly less than j.

The endpoint e_k is either a left endpoint or a right endpoint, hence the set of clones containing (e_k, e_{k+1}) is either $\{i, \ldots, j+1\}$ or $\{i+1, \ldots, j\}$. ■

The path corresponding to an interleaving starts at $(1,1)$ at time $= 1$. At time k, the path is at (i, j) with $i \leq j$, where $\{i, \ldots, j\}$ are the clones which cover the atomic interval (e_k, e_{k+1}). Each step is either to the right (for a clone left endpoint) or a step down

(for a clone right endpoint). The path ends at (n, n). This function from interleavings to lattice paths is a bijection.

Lemma 5.2 *If L is a lattice path starting at $(1, 1)$, ending at (n, n), always stepping right or down, and only passing through cells (i, j) with $i \leq j$, then there exists a unique interleaving that gives rise to L under the above mapping.*

Proof: Omitted. ■

Thus we can count interleavings by counting the lattice paths. Let $a(n)$ be the number of such paths. It is a Catalan number (see [Lov79, Cam84, EG88, Gar76, Bro65]) and is given by

$$a(n) = \frac{1}{n} \begin{pmatrix} 2n-2 \\ n-1 \end{pmatrix}.$$

It will be convenient to define a generating function (see [Wil94, GK90, GKP89, DRS72]) for $a(n)$. Let

$$A(x) = \sum_{n=1}^{\infty} a(n) x^n$$

be a formal power series in x. $A(x)$ has the advantage that it can be simplified:

$$A(x) = \sum_{n=1}^{\infty} a(n) x^n = \sum_{n=1}^{\infty} \frac{1}{n} \begin{pmatrix} 2n-2 \\ n-1 \end{pmatrix} x^n = \frac{1 - \sqrt{1-4x}}{2}.$$

This identity is not hard to verify using repeated differentiation. Also see [Kla70].

5.2.2 Step 2

Our next step is the return of distinguishability to the clones. Also, we will now properly account for the reflection symmetry. However, we will still require that the clones form one island.

Lemma 5.3 *Let $b(n)$ be the number of ways that n distinguishable equal-length clones can be interleaved to form one island. Let*

$$B(x) = \sum_{n=0}^{\infty} b(n) \frac{x^n}{n!}$$

be the exponential generating function for $b(n)$. Then,

$$b(n) = \begin{cases} n & \text{if } n \leq 1 \\ a(n)\frac{n!}{2} & \text{otherwise.} \end{cases}$$

Furthermore,

$$B(x) = \frac{1 + 2x - \sqrt{1 - 4x}}{4}.$$

Notice that $B(x)$, unlike $A(x)$, is a formal *exponential* series in that each $b(n)$ is multiplied by $x^n/n!$, rather than just x^n, before summing. This is why $B(x)$ simplifies to an expression quite similar to that for $A(x)$.

Proof: An island is symmetric if the sequence of the heights of its atomic intervals is a palindrome. Consider an island composed of n clones. The clones in an asymmetric island can be labeled in $n!$ ways. If the island is symmetric and $n > 1$ then the clones can be labeled in only $n!/2$ topologically distinct ways because of the reflection symmetry. Notice that because we ignored the reflection symmetry previously, $a(n)$ double-counts asymmetric islands (of indistinguishable clones) but correctly counts symmetric ones.

Thus when $n > 1$, we have that $a(n)n!/2$ is the number of ways that n distinguishable equal-length clones can be interleaved to form one island. When $n = 1$ there is only one interleaving possible. It is impossible to form an island with no clones so $b(0) = 0$. Thus,

$$B(x) = \sum_{n=0}^{\infty} b(n)\frac{x^n}{n!} = x + \sum_{n=2}^{\infty} \frac{a(n)n!}{2}\frac{x^n}{n!} = \frac{1 + 2x - \sqrt{1 - 4x}}{4}.$$

■

5.2.3 Step 3

We are now ready to address the real problem. We wish to count how many interleavings are possible when we do not restrict the number of connected components (i.e. islands) that the n clones form.

Lemma 5.4 *Let $c(n)$ be the number of interleavings (involving any number of islands) for n clones. Let $C(x)$ be the exponential generating function*

$$C(x) = \sum_{n=0}^{\infty} c(n)\frac{x^n}{n!}.$$

Then,

$$C(x) = e^{B(x)} = \exp\left(\frac{1 + 2x - \sqrt{1 - 4x}}{4}\right).$$

Proof: Consider the exponential generating function $\exp(B(x))$:

$$\begin{aligned}
\exp(B(x)) &= \sum_{k=0}^{\infty} \frac{1}{k!} \prod_{i=1}^{k} \left(\sum_{n_k=0}^{\infty} b(n_k) \frac{x^{n_k}}{n_k!} \right) \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{n=0}^{\infty} \left\{ \sum_{\left(\sum_{i=1}^{k} n_i\right)=n} \binom{n}{n_1 \quad n_2 \quad \ldots \quad n_k} \left(\prod_{i=1}^{k} b(n_i) \right) \frac{x^n}{n!} \right\} \\
&= \sum_{n=0}^{\infty} \left\{ \sum_{k=0}^{\infty} \frac{1}{k!} \left[\sum_{\left(\sum_{i=1}^{k} n_i\right)=n} \binom{n}{n_1 \quad n_2 \quad \ldots \quad n_k} \left(\prod_{i=1}^{k} b(n_i) \right) \right] \right\} \frac{x^n}{n!}
\end{aligned}$$

where $\binom{n}{n_1 \quad n_2 \quad \ldots \quad n_k}$ is the multinomial coefficient $\frac{n!}{\prod_{i=1}^{k} n_i!}$.

The coefficient of $\frac{x^n}{n!}$ in a single term of the innermost summation is of the form

$$\binom{n}{n_1 \quad n_2 \quad \ldots \quad n_k} \left(\prod_{i=1}^{k} b(n_i) \right).$$

The multinomial coefficient counts the number of ways to allocate n clones among k islands where the ith island from the left gets n_i clones. Conditioned upon this distribution, each $b(n_i)$ factor counts how many ways the clones allocated to the ith island can be arranged within that island. Thus, this term counts the number of ways n clones can be distributed within k ordered islands with sizes $n_1, \ldots, n_k$. The summantion over all positive n_i subject to $(\sum n_i) = n$ gives a count of the number of ways n clones can be distributed within k ordered islands, regardless of their sizes. The $k!$ divisor cancels the overcounting we have introduced by ignoring the equivalence of interleavings in which the islands are permuted. (Notice that the $k!$ divisor is appropriate even if two or more of the k islands have the same size because these islands are distinguished by the clones they contain.) The summation over all k gives the number of ways n clones can be distributed within any number of unordered islands. That is, it is the number of interleavings for n clones. ■

The use of exponentials in exponential generating functions is described in [DRS72, Example 5.5, p. 287].

5.3 Recurrence Relation and Asymptotic Growth

5.3.1 Recurrence Relation

Because of its complexity we cannot solve explicitly for the power series of $C(x)$. However, we can find an easy recurrence relation by using the fact that $C(x)$ satisfies a simple differential equation.

Lemma 5.5

$$c(n) = (4n-5)c(n-1) - (4n-7)c(n-2) + (n-2)c(n-3)$$

for $n \geq 3$.

Proof: The first two derivatives of $C(x)$ are

$$\begin{aligned}
\frac{dC(x)}{dx} &= \left(\frac{1}{2} - \frac{1}{2\sqrt{1-4x}}\right) C(x) \\
\frac{d^2C(x)}{dx^2} &= \left(\frac{1}{4} + \frac{1}{2\sqrt{1-4x}} + \frac{1}{4(1-4x)} + \frac{1}{(1-4x)\sqrt{1-4x}}\right) C(x)
\end{aligned}$$

The factors of $\sqrt{1-4x}$ are troublesome so we solve for $C(x)/\sqrt{1-4x}$ in the first equation and substitute into the second equation. We see that $C(x)$ satisfies the differential equation

$$(1-4x)\frac{d^2C}{dx^2} + (4x-3)\frac{dC}{dx} + (1-x)C = 0.$$

Substituting in the definition for $C(x)$ we get

$$(1-4x)\sum_{n=2}^{\infty} c(n)\frac{x^{n-2}}{(n-2)!} + (4x-3)\sum_{n=1}^{\infty} c(n)\frac{x^{n-1}}{(n-1)!} + (1-x)\sum_{n=0}^{\infty} c(n)\frac{x^n}{n!} = 0.$$

A power series can only be zero if the coefficient of every term is zero. This fact and the application of a little algebra to the above formula gives the desired result. ■

This relation provides a way to compute $c(n)$ using $\Theta(n)$ arithmetic operations. The first few values of $c(n)$ can be found in Table 5.1. Note that this sequence is *not* in Sloane's *Handbook of Integer Sequences,* [Slo73].

5.3.2 Asymptotic Growth

We can calculate the asymptotic growth of $c(n)$ via Darboux's lemma (see [Wil94, p. 148]) applied to $C(x)$.

Lemma 5.6

$$c(n) \sim \frac{e^{3/8}\sqrt{2}}{8n}\left(\frac{4n}{e}\right)^n$$

Proof: The singularity of $C(x)$ nearest to the origin of the complex plane (and, in fact, the only singularity) is at $x = 1/4$. We will express $C(x)$ in terms of functions with a singularity at $x = 1/4$, but simpler, and functions with a radius of convergence about the origin larger than $1/4$. We decompose $C(x)$ as follows:

$$\begin{aligned} C(x) &= \exp\left(\frac{1+2x}{4}\right)\exp\left(-\frac{\sqrt{1-4x}}{4}\right) \\ &= \exp\left(\frac{1+2x}{4}\right)\left[\cosh\left(\frac{\sqrt{1-4x}}{4}\right) - \frac{\sqrt{1-4x}}{4}\frac{\sinh\left(\frac{\sqrt{1-4x}}{4}\right)}{\sqrt{1-4x}/4}\right] \\ &= \exp\left(\frac{1+2x}{4}\right)\left[\cosh\left(\frac{\sqrt{1-4x}}{4}\right) - \frac{\sqrt{1-4x}}{4}H(\frac{\sqrt{1-4x}}{4})\right] \end{aligned}$$

where $H(y)$ is defined to be $\sinh(y)/y$. Notice that both $\cosh(\cdot)$ and $H(\cdot)$ are even functions and "cancel out" the square-roots in their argument. The functions $\exp(\frac{1+2x}{4})$, $\cosh(\frac{\sqrt{1-4x}}{4})$, and $H(\frac{\sqrt{1-4x}}{4})$ are entire (i.e., analytic over the entire complex plane) functions of x.

According to Darboux's lemma the asymptotic growth of $c(n)$ will be the same as the asymptotic growth indicated by the simplified generating function which is obtained by replacing those functions which are analytic over a radius greater than $1/4$ by their values at the singularity $x = 1/4$. Thus, $c(n)$ is asymptotic to the coefficient of $x^n/n!$ in the generating function

$$e^{3/8}\left(1 - \frac{\sqrt{1-4x}}{4}\right)$$

where we have used the fact that $\lim_{x\to 0}\sinh(x)/x = 1$. The exponential generating function

$$1 - \frac{\sqrt{1-4x}}{4}$$

defines $b(n)$ for $n > 1$, thus we conclude that

$$\begin{aligned} c(n) &\sim e^{3/8}b(n) \\ &\sim e^{3/8}\frac{(2n-2)!}{2(n-1)!} \\ &\sim \frac{e^{3/8}\sqrt{2}}{8n}\left(\frac{4n}{e}\right)^n \end{aligned}$$

where we have used Stirling's formula, $n! \sim \sqrt{2\pi n}n^n e^{-n}$. ■

Part II

Probed Partial Digests

Chapter 6

A Branch-And-Bound Algorithm

(This material will also appear as [KN95].)

Under certain experimental conditions a restriction enzyme cuts at some but not all sites where the DNA matches the pattern. This process is called **partial digestion**. (See [KAI87].) Many copies of a clone are partially digested at once. In (unprobed) partial digestion, for each pair of cut sites a fragment is obtained which is a copy of the DNA between those two cut sites. However, it is not known which pair of cut sites produces which fragment. Using **gel electrophoresis** the lengths of these fragments are measured. From these lengths, the original locations of the cut sites can be reconstructed. The algorithms for this task vary in efficiency and in ability to handle errors in measurements. See [CGL+89, SSL90, SS94].

In **probed partial digestion**, a probe is chosen which hybridizes to the uncut clone in a unique location. The copies of the clone are partially digested, but the only lengths measured are those of fragments which hybridize to the radioactively or fluorescently labeled probe. (See [SES+87].)

In this chapter we give an efficient algorithm for interpreting the data from a probed partial digestion experiment. The algorithm produces one or more candidate solutions, each of which designates the locations of the cut sites and specifies the end points of each fragment. If necessary, further experiments can then be designed to select the most likely solution from this small set of candidates. Such a solution is useful in DNA mapping for it provides a fingerprint of the clone. Two clones can be hypothesized to overlap if the right part of the restriction map for one clone agrees with the left part of the restriction map for the other clone.

M	=	the experimentally measured fragment lengths
N	=	fragments with neither x_0 nor y_0 as an endpoint
X	=	fragments with x_0 but not y_0 as an endpoint
Y	=	fragments with y_0 but not x_0 as an endpoint
B	=	the fragment that has both x_0 and y_0 as endpoints
C	=	$(X \cup B) + (Y \cup B) - B$, the computed fragment lengths
Z	=	the fragment lengths not yet assigned to N, X, Y, or B.

Table 6.1: Multisets Used by the Probed Partial Digest Algorithm

The algorithm works well even when there are errors in the length measurements. The resilience is derived, in part, from the algorithm's heavier reliance on the lengths of the shorter fragments, which are measured more accurately in the experiment. The algorithm cannot handle the cases in which some measurements are completely lost. In many cases there may be more than one mathematically correct solution (see Chapter 7 and [NN93]). The algorithm handles this gracefully in that it need not be terminated when the first solution is produced. It will generate all solutions, in descending order of quality, until terminated by the user.

A solution to the **Probed Partial Digest Mapping Problem** is in the form of a **complete assignment** of the fragment lengths to four multisets. (A **multiset** is a set in which elements may appear more than once.) Let x_0 be the nearest cut site on one side of the probe and let y_0 be the nearest cut site on the other side of the probe. Each fragment that includes the probe site is assigned to one of four multisets; this assignment depends on how the endpoints of the fragment compare to $\{x_0, y_0\}$. The shortest fragment, which must represent the interval that has *both* x_0 and y_0 as endpoints, is assigned to the one-element multiset B. Fragments with one endpoint at x_0 and the other not at y_0 are assigned to the multiset X. Fragments with one endpoint at y_0 and the other not at x_0 are assigned to the multiset Y. All other fragments are assigned to the multiset N. See Table 6.1 and Figure 6.1.

As well as acting as a fingerprint, the collection of fragments whose lengths are assigned to X or Y can be used to locate markers on the clone. Each atomic interval is contained by a unique subset of this collection. Hence, knowledge of which fragments contain the marker is sufficient for determining the atomic interval in which it lies. For instance, any marker present on all X fragments but the shortest and absent from all other

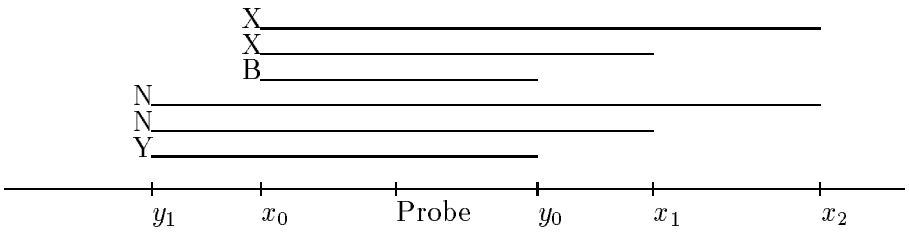

Figure 6.1: The Fragments Produced by Probed Partial Digestion

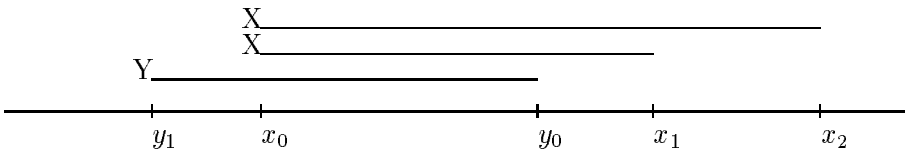

Figure 6.2: The Fragments Found by the Algorithm

fragments must lie in the atomic interval between x_1 and x_2 shown in Figure 6.2.

Given a particular complete assignment it is easy to compute, from the lengths in X, Y, and B, what the lengths in N ought to be. Using multiset addition and subtraction notation we write

$$X + Y - B \stackrel{\text{def}}{=} \{x + y - b : x \in X, y \in Y, b \in B\}$$

for these lengths. Since B has one element, the size of this multiset is $|X| \cdot |Y|$. The union of this multiset with the multisets X, Y, and B is the multiset of **computed lengths**, C. (Note that, with multisets, the number of occurrences of an element in a union of multisets is the sum of the numbers of occurrences of that element in those multisets.) The computed multiset can also be written as $C = (X \cup B) + (Y \cup B) - B$ and we see immediately that the number of elements in the computed multiset is

$$|C| = (|X| + 1)(|Y| + 1).$$

Example: From $X = \{6, 8, 9\}$, $Y = \{7, 9, 12, 14\}$, and $B = \{5\}$ we compute $C = X + Y - B = \{5, 6, 7, 8, 8, 9, 9, 10, 10, 11, 12, 12, 13, 13, 14, 15, 15, 16, 17, 18\}$.

If the complete assignment is correct then, in the absence of experimental error, the multisets C and M should be identical. A comparison of C against the measured fragment lengths M gives a measure of the quality of the complete assignment; the more similar C and M are, the more likely it is that a hypothesized complete assignment is the biologically correct answer. To ascertain the similarity, the lengths in C are paired up with those in M. A positive **cost** (i.e., penalty) is associated with those pairs in which the paired lengths differ. The cost as a function of the two lengths is determined by the error model for the experiment and the cost of a complete assignment is the minimum, over all possible pairings, of the sum of the costs of the pairs. The goal of the algorithm is to find the complete assignment(s) with the smallest cost.

The algorithm considers the elements of M in ascending order, and assigns each element to one of the sets B, X, Y and N. Until all the elements are assigned, an assignment is called **partial**. A data structure called a *priority queue* (see, for instance, [CLR90]) is created to contain a collection of partial assignments. The priority queue efficiently supports two operations: the *insertion* of a partial assignments and the *removal* of a partial assignment of minimum cost. The *Branch and Bound Technique* (see [CLR90]) is applied to speed the exponential search through the universe of assignments. In a general step, a lowest cost partial assignment is removed from the priority queue and replaced by several, more complete, partial assignments. These are generated by choosing an assignment for the first unassigned element of M.

As with a complete assignment, the **cost** of a partial assignment is evaluated by checking the fragment lengths it implies against the input M. The elements already assigned imply that certain lengths (plus or minus experimental error) ought to belong to M. Furthermore, there is a lower bound on the length of any fragment that might be implied by a future assignment. As described later, the cost of a partial assignment is computed using this information. The cost function has the property that an extended assignment always costs at least as much as the partial assignment from which it was extended.

6.1 Algorithm

6.1.1 Branch and Bound

The elements of M are assigned in ascending order. The smallest element must be the distance from x_0 to y_0. This element will be the element assigned to B. Without loss of generality, the second smallest element of M is assigned to the multiset X. This assignment of the first two elements of M is the first partial assignment to be placed in the priority queue.

In the general step, a partial assignment of minimum cost is removed from the priority queue. If the partial assignment is complete, (i.e., all elements of M have been assigned) it is given as output. Otherwise three partial assignments are inserted into the priority queue. These are created by assigning a least unassigned value of M to X, Y, or N. Because the cost function has the property that an extended assignment always costs at least as much as the partial assignment from which it was extended, the first complete assignment produced by the algorithm will be of minimum cost. If the algorithm is not terminated when it produces the first minimum cost complete assignment, it will continue to produce complete assignments in nondecreasing order of cost.

The need for a priority queue can be eliminated if a different form of Branch and Bound is used. This may be advantageous under circumstances where space usage (i.e., the amount of computer memory required) is more critical than running time. In this alternate implementation a *Depth-First Search* is used. (See [CLR90], for instance, for a description of depth-first search, rooted trees, etc.) The initial priority queue entry is the root of a tree of partial assignments. Every partial assignment is represented by an internal node of the tree and has a directed edge pointing to each of the three assignments that can be formed by assigning the least unassigned length. The complete assignments are the leaves of the tree. The goal, to find the cheapest complete assignment(s), is accomplished via standard Depth-First Search except that if an encountered partial assignment is found with cost exceeding some fixed bound then the edges of the tree leading from that assignment are not explored. Any leaves reached by the search are complete solutions to be output.

Care must be taken in chosing a cost bound. A bound too low may result in the discovery of fewer low cost complete assignments than desired. If this occurs, the bound should be increased and the search should be restarted. A bound too high may cause the algorithm to spend large amounts of time exploring assignments that are of no interest. If

during the search the algorithm has found an excessive number of solutions and/or explores an excessive number of assignments then the search should be restarted with a lower cost bound.

6.1.2 The Cost of a Partial Assignment

For a partial assignment let N, X, Y, and B be the multisets of already assigned lengths and let Z be those lengths not yet assigned.

If the partial assignment (N, X, Y, B, Z) is to lead to a good complete assignment, the multiset M must contain data similar to that in the multiset of computed fragment lengths C (see Table 6.1). In particular, if $|C| > |M|$ then the partial assignment cannot lead to a complete solution and is given infinite cost. On the flip side, the partial assignment is also given infinite cost if every complete solution to which it may lead has a computed fragment length multiset with size strictly smaller than $|M|$. This maximum obtainable size is not hard to compute: Since $|C| = (|X|+1)(|Y|+1)$, the size of C will be maximized if all the elements of Z are assigned to X or Y (but not N) in such a way that $|X|$ and $|Y|$ are as nearly equal as possible.

If a partial assignment passes these two size tests, it is checked for compatibility with M. It is assigned a nonnegative cost which is low if the compatibility is good and high if it is not. The compatibility test involves a matching which pairs elements of C with elements of M. Since all solutions extended from this partial assignment will have computed fragment length multisets which contain C, for every length in C there should be a corresponding length in M. (The matched lengths should be roughly equal, though the possibility of experimental error dictates that we cannot insist upon strict equality.) However, since the computed fragment length multiset may grow with future assignments not every element of M need have a match in C. Thus, for the purpose of matching, a special element *cutoff* is added to C. It will be matched with every element of M not matched to an ordinary element of C. In the event that Z is empty, $|C| = |M|$ by the size tests and *cutoff* is not used.

A **matching** is a function $\mu : M \to C$ which takes an element of M to its match in C. The function is bijective (i.e., one-to-one and onto) except that more than one element of M may map to the *cutoff* element of C. The cost of the entire matching is defined by

$$Cost_{\mu}(M, C) \stackrel{\text{def}}{=} \sum_{m \in M} cost(m, \mu(m))$$

where $cost(m, \mu(m))$ measures the quality of the match, $m \leftrightarrow \mu(m)$. The quality of the partial assignment is given by the cost of a matching which minimizes $Cost_\mu(M, C)$.

6.1.3 The Cost of a Pair

The nonnegative cost function $cost(m, c)$ should reflect the error model. For instance, if one assumes that the error in the measurement of the length of a fragment obeys a Gaussian distribution with standard deviation proportional to the fragment's length then one should use $cost(m, c) = |\log(m) - \log(c)|^2$. The logarithms change the scale so that the distribution of error is independent of fragment length. The quantity is squared so that it can be combined by summing with other independent pairings that obey a Gaussian distribution.

If one assumes that the error is Gaussian with standard deviation independent of the the fragment length then one should use $cost(m, c) = |m - c|^2$. We tested the algorithm under the assumption that the distance a fragment travels in a gel is inversely proportional to its length and that the error in measuring gel position is Gaussian with standard deviation independent of gel position. Hence we use

$$cost(m, c) = \left| \frac{1}{m} - \frac{1}{c} \right|^2 .$$

With regard to the special element *cutoff*, *cost* must satisfy

$$cost(m, cutoff) = \begin{cases} 0 & \text{if } m \geq \min Z, \\ cost(m, \min Z) & \text{otherwise.} \end{cases}$$

This reflects the fact that, as the result of future assignments, the smallest element that might be added to X or Y and hence to C is $\min Z$. This requirement incorporates into $Cost_\mu(M, C)$, a lower bound on the cost of future assignments. Hence it gives a lower bound to the cost of all complete solutions to which the partial assignment may lead.

For any of these three choices *cost* satisfies the following regularity property. For all $m \leq m'$ and all $c \leq c'$,

$$cost(m, c) + cost(m', c') \leq cost(m, c') + cost(m', c).$$

(In fact, if $cost(m, c) = |f(m) - f(c)|^2$ for any monotonic function $f()$ then cost will satisfy this regularity property.) For any $cost(m, c)$ satisfying this regularity property it is never worse to match the elements in order than out of order. For equal size multisets, M, C, the

property implies that a best matching is one in which the smallest element of M is matched to the smallest element of C, the second smallest element of M is matched to the second smallest element of C, etc.

6.1.4 Finding the Cost of an Optimal Matching

The value $Cost(M, C)$, the minimum over all matchings of $Cost_\mu(M, C)$, can be found efficiently using dynamic programming. Write c_i for the ith lowest element of C and write C_i for the multiset $\{c_1, \ldots, c_i\}$. Define m_i and M_i similarly. For $i \geq j \geq 0$, let $Cost(M_i, C_j)$ be the minimum cost of a matching between M_i and $C_j \cup \{cutoff\}$. In the degenerate cases, define

$$Cost(M_i, C_j) = \begin{cases} 0 & \text{if } i = j = 0, \\ +\infty & \text{if } i < 0,\ j < 0, \text{ or } i < j. \end{cases}$$

Because of the regularity restriction on the function *cost*, in a best matching those elements of M_i not matched to *cutoff* will be matched in ascending order to the elements of C_j in ascending order. In particular, when computing $Cost(M_i, C_j)$, m_i will be matched to c_j or *cutoff*. Thus, for $i \geq j \geq 0$, $Cost(M_i, C_j)$ satisfies the recursive relation:

$$\begin{aligned} Cost(M_i, C_j) \quad = \min\{ \quad & cost(m_i, c_j) + Cost(M_{i-1}, C_{j-1}), \\ & cost(m_i, cutoff) + Cost(M_{i-1}, C_j)\}. \end{aligned}$$

That is, m_i must be matched to c_j or *cutoff*, and once it is, the problem is reduced to a smaller version of the problem. Thus, standard dynamic programming techniques can be used to find $Cost(M, C)$; Figure 6.3 contains a sample program. Note that the occurrence of $m - c$ as the upper bound of the inner loop keeps the algorithm from computing values of $Cost(M_i, C_j)$ that cannot contribute to the return value $Cost(M, C)$.

As an example, suppose the probe is at position 10 and the cut sites are at 2, 3, 8, 12, and 19. If there is no error, the probed partial digestion experiment will produce the lengths $M = \{4, 9, 10, 11, 16, 17\}$. Our goal is to correctly classify these lengths to B, X, Y, and N. A biologically correct solution is to assign the lengths, in order, to B, X, X, Y, N and N.

Suppose we wish to evaluate the partial assignment $B = \{4\}, X = \{9\}, Y = \{10\}, N = \emptyset, Z = \{11, 16, 17\}$. We compute $C = (X \cup B) + (Y \cup B) - B = \{4, 9\} + \{4, 10\} - \{4\} = \{4, 9, 10, 15\}$ and $\min Z = 11$. The dynamic programming algorithm optimally

```
double cost(
  double  m,                          /* A measured length */
  double  c)                          /* A computed length */
{
  if (c <= 0.0) {                     /* If c <= 0.0 then it is the negative of a cutoff */
    c = -c;
    if (m >= c) {
      return 0.0;}}                   /* m is above the cutoff */

  return (1/m - 1/c) * (1/m - 1/c);
}

double best_match(
  int       m,                        /* The number of measured lengths */
  double  measured[ ],                /* The measured lengths, sorted */
  int       c,                        /* The number of computed lengths */
  double  computed[ ],                /* The computed lengths, sorted */
  double  cutoff)                     /* The shortest unassigned length */
{
  double *value;                      /* value[i] = Cost(M[j+i], C[j]) for the implied j */
  int       i;                        /* Index into the measured[ ] array */
  int       j;                        /* Index into the computed[ ] array */

  /* Allocate memory for value[0..m-c] */
  value = (double *) malloc((m - c + 1) * sizeof(double));

  /* Fill in value[ ] for j = 0 */
  value[0] = 0.0;
  for (i = 1; i <= m - c; i++) {
    value[i] = cost(measured[i], -cutoff) + value[i - 1];}

  /* Repeat for each computed length */
  for (j = 1; j <= c; j++) {
    value[0] = cost(measured[j], computed[j]) + value[0];
    for (i = 1; i <= m - c; i++) {
      value[i] = min(cost(measured[j + i], -cutoff) + value[i - 1],
                     cost(measured[j + i], computed[j]) + value[i]);}}

  free(value);                        /* Deallocate the allocated memory */

  return value[m - c];
}
```

Figure 6.3: Dynamic Programming Algorithm for Computing the Cost of an Optimal Matching

	j	c_j							
i			0	1	2	3	4	5	6
m_i				4	9	10	11	16	17
	0		.000000	.025310	.025718	.025801	.025801	.025801	.025801
	1	4	$+\infty$	.000000	.000408	.000491	.000491	.000491	.000491
	2	9	$+\infty$	$+\infty$	.000000	.000083	.000083	.000083	.000083
	3	10	$+\infty$	$+\infty$	$+\infty$	.000000	.000000	.000000	.000000
	4	15	$+\infty$	$+\infty$	$+\infty$	$+\infty$	.000588	.000017	.000017

Table 6.2: An Example of the Cost Computation for a Partial Assignment Using $cost(m, c) = |1/m - 1/c|^2$, $cutoff = 11$. All values are shown although only those on the diagonal from the upper left corner and the adjacent two diagonals to its right are needed for the computation of $Cost(M, C) = Cost(M_6, C_4)$.

matches $C \cup \{cutoff\}$ with M. For $cost(m, c) = |1/m - 1/c|^2$, the values of $Cost(M_i, C_j)$ are in Table 6.2. The cost of an optimal matching is $Cost(M_6, C_4) = .000017$. In this case, the cost of the partial assignment comes from an optimal matching which is unique. It matches the elements of M, in order, to $4, 9, 10, cutoff, 15, cutoff$.

6.2 Discussion

Let $N = |M|$ be the number of fragment lengths given as input to the algorithm. The dynamic programming step of the algorithm requires a constant amount of time to compute each $Cost(M_i, C_j)$ from previous values. Since for a given partial assignment there may be $O(N^2)$ values of $Cost(M_i, C_j)$ to compute, the evaluation of a partial assignment requires $O(N^2)$ time.

If a priority queue is used, each step of the Branch and Bound portion of the algorithm requires $O(\log N)$ time for the priority queue operations. However, this is dominated by the time to evaluate the cost of the three extended partial assignments. Thus the total running time of the algorithm is $O(iN^2)$ where i is the number of Branch and Bound iterations required. In theory, i can be as large as 3^{N-2} but in practice it is much smaller.

6.2.1 Using a Priority Queue

The algorithm using a priority queue was tested on several different sets of randomly generated data. The sets are characterized by **size** and **error level**. Sizes are written $x \times y$ to indicate how many cut sites are before and after the probe site. The product xy is the number of distinct fragments produced by the experiment.

The error level describes the error in a measurement of a fragment of unit length. It is assumed that the distance a fragment travels in a gel is inversely proportional to its length and that the uncertainty in measuring its gel position is a Gaussian with a standard deviation which is independent of that position. Under such circumstances, a typical error in a value obtained for a fragment length (as opposed to a value for gel position) will be proportional to the square of the fragment length. With the clone length normalized to unity, an error level of ϵ implies that the measurement of a fragment of length m will have a standard deviation of ϵm^2.

The error level is based upon the error in measuring a single fragment length *independent* of the other fragment lengths. For instance, a gel inaccuracy that causes all fragments to appear approximately 5% too long will not hinder the performance of the algorithm and is not included in the definition of error level. If for a particular experiment, a fragment of unit length would be typically measured with error around 1% *even when all other fragments were measured accurately* then an error level of 1% is appropriate.

For size $x \times y$ the $x + y$ cut sites are chosen uniformly at random on a clone with length normalized to unity. From these cut sites the xy exact fragment lengths are calculated. In the cases of non-zero error level each fragment length is perturbed by an amount which is generated from a Gaussian distribution with the appropriate standard deviation. Note, however, that the perturbation is not allowed to change the order of the fragments' lengths. If the perturbation results in a change in the order of the fragments' lengths, the perturbed lengths are sorted in ascending order and are matched with the exact lengths in ascending order.

For each set, 100 instances were run. The results are in Tables 6.3 and 6.4. Note that every data instance M produced a **trivial solution**: $B = \{\min M\}$, $X = M \backslash \{\min M\}$, $Y = \emptyset$, $N = \emptyset$. It corresponds to a restriction map in which the probe site separates one cut site from all the others. The lengths measured are those from the one cut site to each of the other cut sites. Although mathematically correct, this zero-cost solution is uninteresting

	0% Error		1% Error		3% Error		5% Error	
Rank	Deletes	Correct	Deletes	Correct	Deletes	Correct	Deletes	Correct
trivial	55.28	0%	29.00	0%	29.00	0%	29.00	0%
1	56.28	100%	157.28	60%	667.64	30%	1821.43	9%
2	100.89	0%	197.36	21%	705.71	16%	1869.68	10%
3	157.95	0%	257.14	4%	784.14	10%	2021.64	3%
4	214.79	0%	300.72	8%	902.08	5%	2096.37	4%
5	302.37	0%	376.72	2%	962.59	3%	2206.25	3%
6	339.17	0%	436.25	0%	1002.08	2%	2320.30	3%
7	417.45	0%	532.37	0%	1082.09	3%	2423.18	0%
8	535.87	0%	601.68	0%	1126.65	0%	2469.01	1%
9	597.14	0%	675.51	2%	1219.02	1%	2561.39	2%
10	645.86	0%	738.58	0%	1265.37	3%	2623.78	1%
> 10		0%		3%		27%		64%

Table 6.3: The Average Number of Priority Queue Deletions and the Number of Correct Solutions Appearing at Each Rank for Problems of Size 6×5.

biologically. The correct solution may be ranked first, second, etc. among the non-trivial solutions produced. The tables give how many correct complete assignments appeared at each rank and the average number of priority queue deletions (i.e., Branch and Bound iterations) required to produce all solutions up to that rank.

The data in Table 6.3 is intended to be typical of existing technology. The measurement of 30 fragments at an error level of 3% – 5% is feasible. The results are good; at the 3% error level, the correct assignment appeared as one of the first three complete assignments found in over half of the randomly generated instances, and appeared among the top ten in almost three-quarters of the instances. By observing how the top solutions differ and running experiments to test for these differences one can easily eliminate the spurious solutions.

The data in Table 6.4 gives a glimpse of the future. Because of the high number of fragments, measurements must have an error level not much more than 0.3% if the correct solution is to be found (using *any* algorithm) among those complete assignments with the smallest cost. Once the technology has improved to the point that the correct solution is among the assignments of lowest cost, the algorithm presented here will narrow the problem to that of finding the correct solution among a handful of complete assignments.

	0.0% Error		0.1% Error		0.2% Error		0.3% Error	
Rank	Deletes	Correct	Deletes	Correct	Deletes	Correct	Deletes	Correct
trivial	234.98	0%	119.00	0%	119.00	0%	119.00	0%
1	235.98	100%	426.37	64%	876.51	51%	2359.14	34%
2	330.70	0%	491.58	24%	959.72	17%	2441.74	24%
3	438.68	0%	585.06	4%	1091.53	10%	2737.44	12%
4	512.65	0%	657.49	4%	1177.66	7%	2846.59	3%
5	617.10	0%	771.27	1%	1413.03	4%	3018.66	7%
6	684.71	0%	842.52	2%	1538.11	2%	3106.61	4%
7	778.01	0%	956.32	0%	2158.15	3%	3260.89	5%
8	858.58	0%	1015.67	0%	2263.24	1%	3336.99	1%
9	951.20	0%	1139.09	0%	2516.03	0%	3494.91	0%
10	1020.34	0%	1200.84	0%	2630.78	0%	3569.40	1%
> 10		0%		1%		5%		9%

Table 6.4: The Average Number of Priority Queue Deletions and the Number of Correct Solutions Appearing at Each Rank for Problems of Size 12×10.

6.2.2 Using Depth-First Search

For the space-saving Depth-First Search version of the algorithm it is important that the initial cost bound be less than or equal to the cost bound that would produce the number of correct solutions. Even when the initial cost bound is too small the amount of time wasted in finding too few solutions is small compared to the amount of time that will be needed to find sufficiently many solutions. A bound that is too large even by a little can be disastrous since running time is likely to be exponential as a function of the bound.

We have found that the value $b = 1/N^2\ell^2$, where N is the number of fragments and ℓ is the length of the clone (or of the longest fragment) makes a good initial cost bound for the cost function $cost(m, c) = |1/m - 1/c|^2$. If the bound proves to be too low we use a bound of $2b$ for the second attempt. If the cost bound is still too low after two or more attempts we fit $w(b)$ the amount of work (i.e., the number the nodes explored) for the two most recent runs to an exponential with two parameters: $w(b) = c_1 e^{c_2 b}$. The new bound is chosen so that the amount of work it will require (as predicted by the exponential) will be twice the amount of work done for the most recent bound. If the predictions are accurate this guarantees that running time for the last cost bound used will be at least half of the total running time. Furthermore, the last bound's running time will be at most twice the

running time of a perfectly guessed cost bound. Thus the the total running time of the algorithm will be at most four times that which would have been needed if we were able to perfectly guess a cost bound that produced the correct number of solutions.

Letting b' be the most recent bound and b'' be the second-most recent bound, the new bound b is given by the formula

$$b = b' + \frac{b' - b''}{\log_2 w(b')/w(b'')}.$$

As a practical matter, to prevent huge changes in the cost bound, if $\log_2 w(b')/w(b'')$ is less than $1/4$ it is replaced by $1/4$ in the above formula.

Chapter 7

Counting Probed Partial Digest Maps

(This material also appeared as [NN93].)

The probed partial digest (or PPD) mapping scheme is used to generate restriction maps of cloned DNA fragments. The objective: to reconstruct the linear order of the restriction sites using the lengths of subfragments that hybridize to a probe. (The experiment is introduced in Chapter 6).

The computational problem of reconstructing the linear order is a difficult one. One of the difficulties is the multiplicity of solutions. That is, in many cases, more than one underlying linear ordering is consistent with a multiset of measured lengths, although in reality there is only one true linear order for the restriction enzyme cutting sites. Since, without additional information, there is no way to distinguish between the true linear order and any other consistent order, the objective is to reconstruct *all* possible solutions and leave it to the biologists to decide which solution is most likely to be the right one.

The fact that a given multiset of measured lengths may have multiple solutions has two implications. First, by the discussion above, it affects the efficiency of any algorithm that reconstructs the restriction enzyme map. It also reflects on the power of the mapping scheme to resolve ambiguities among maps; that is, a mapping scheme that yields many consistent maps may not be adequate. The question of multiplicity of solutions for other types of DNA mapping strategies has been previously addressed. It [GW87] it is shown that if the enzyme sites are modeled as a Poisson process, then the Double Digest

Mapping Problem can attain as many as an exponential number of solutions in the limit; these solutions have been further characterized in [SW91]. The status of the question for (unprobed) Partial Digest mapping is summarized later in this article.

We show in Section 7.1 that PPD is equivalent to the following multiset-addition problem: Let X and Y denote sets of nonnegative integers. Let S be a multiset of nonnegative integers containing exactly one zero. Given S find all unordered pairs $\{X, Y\}$ such that

$$S = X + Y = \{x + y : x \in X, y \in Y\}.$$

For instance, for $S = \{0, 1, 1, 2, 2, 3, 3, 4, 5\}$ the pair $X = \{0, 1, 2\}$, $Y = \{0, 1, 3\}$ satisfies $S = X + Y$. In this article we discuss how many solutions this multiset-addition problem, and therefore the Probed Partial Digest Mapping Problem, might have for a given input multiset S.

A related problem, the (unprobed) **Partial Digest Mapping Problem**, is well studied and is discussed in [AY88, Bel88, DK88, RS82, SSL90, Ste78, TDMH88]. This problem is equivalent to the following multiset-subtraction problem: Given a multiset S of positive integers, find a set X such that

$$S = \{a - b : a, b \in X, a > b\}.$$

In [SSL90] it is shown that there are at most $0.5N^{0.61624135}$ solutions to this multiset-subtraction problem for any multiset S of size N.

The PPD Mapping Problem can have more solutions. Define $\#\text{ppd}(S)$ to be the number of solutions for a multiset S that contains exactly one zero. It is shown in [Nao90] that there are infinitely many values N for which there exists a multiset S, of size N, with $\#\text{ppd}(S) \geq 0.5N$. This chapter generalizes and improves this bound. Define

$$\#\text{PPD}(N) = \sup\{\#\text{ppd}(S) : |S| = N\}.$$

We show that

$$\limsup_{N \to \infty} \frac{\#\text{PPD}(N)}{N^t} = \infty$$

for all $t < \zeta^{-1}(2)$ where $\zeta(t)$ is the Riemann Zeta Function and $\zeta^{-1}(2) \approx 1.73$.

Biologists may find this result disenheartening for it implies that the information derived from a PPD experiment is not sufficient to determine the linear order of the restriction enzyme cutting sites in all cases. Even though there is only one real linear ordering of

the restriction enzyme cutting sites, there may be many linear orderings which are consistent with the fragment lengths measured by experiment. Furthermore, it is not yet known how bad this phenomenon can get. We have a lower bound, but no upper bound, on the number of solutions in the worst case.

However, it is still left to be determined how bad this phenomenon is in the average case. The PPD experiment is still viable, if there is only one solution (or very few solutions) on average. Another experimental technique should be employed in those few cases where the number of PPD solutions is excessive.

In fact, we simulated the PPD experiment on two real DNA vectors (circular DNA strands), the M13 and the Colicin E1 vectors, which are 6407 and 6576 nucleotides (characters) long respectively. We generated 45 multisets for the M13 and 69 multisets for the Colicin with different restriction enzymes. (This was done by obtaining the cutting sites of many restriction enzymes along these vectors; then, for each restriction enzyme we generated a random location for the probe and produced the multiset of lengths implied by the PPD experiment with this pair of enzyme and probe). The number of cutting sites in each instance varied from 5 to 55, and the size of the multisets ranged from 10 to about 800. Each multiset had one solution, namely the true linear order it was derived from. None of these multisets produced any other solution. This simulation suggests that the number of solutions to a typical multiset that is obtained from real, biological, data is very likely to be one.

Remark - The lengths of DNA fragments are measured with error. Furthermore, it is often not possible to detect how many fragments are of a given length. Thus, in general, the multiset of lengths is fraught with errors in measurement and with loss of multiplicity information. We do not consider these two complicating factors. However, we should note that under such circumstances the number of solutions to a given set of measured lengths will behave differently than under the assumption of exact measurements.

7.1 The Multiset-Addition Problem

The PPD Mapping Problem can easily be stated as a multiset-addition problem. The clone is represented by an interval on the real line, and the integers along this line represent the nucleotide positions. The probe hybridizes at zero and the restriction enzyme cut sites are represented as a set, C, of integers. Let $Y' = \{c : c \in C, c \geq 0\}$ and let

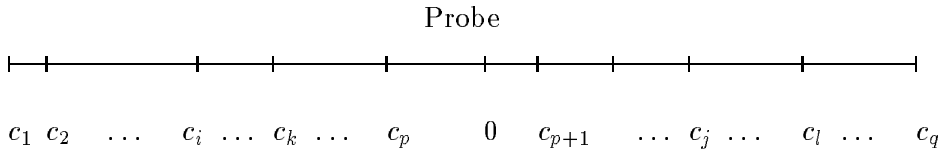

Figure 7.1: Linear Order of the Cutting Sites and the Probe

$X' = \{|c| : c \in C, c < 0\}$. The result of the partial digestion experiment is the multiset $S' = X' + Y'$. The task of the biologist is to find sets of nonnegative integers, X' and Y', from S'.

Notice that the results of the experiment would not be different if the probe were anywhere in the interval $[-\min X', \min Y']$. Equivalently, for any integer p in the interval $[-\min X', \min Y']$, the pair $\{\{x + p : x \in X'\}, \{y - p : y \in Y'\}\}$ would also be a solution to S'. These solutions are called **congruent**. We wish to count each group of congruent solutions exactly once.

The number of integers in $[-\min X', \min Y']$ and, hence, the number of pairs in a congruency class is $1 + \min S'$. In particular, if $\min S'$ were zero there would be only one solution pair $\{X', Y'\}$ per congruency class. With that in mind, we define $S = \{s - \min S' : s \in S'\}$. S is a multiset with exactly one zero. The solutions $\{X', Y'\}$ for S' map $(1+\min S')$-to-1 to the solutions $\{X, Y\}$ for S. The mapping is:

$$\begin{aligned} X' &\longrightarrow X = \{x - \min X' : x \in X'\} \\ Y' &\longrightarrow Y = \{y - \min Y' : y \in Y'\} \end{aligned}$$

Counting all the solutions for S is the same as counting congruency classes of solutions for S'. Hence, the number of unordered pairs $\{X, Y\}$ which solve S is the number of noncongruent solutions to the PPD Mapping Problem.

Thus, we have transformed the PPD Mapping Problem to the multiset-addition problem stated earlier.

7.2 Mixed Radix Representation

Our lower bound on the number of solutions to the multiset-addition problem uses a **mixed radix** representation of the integers. This is a generalization of the decimal representation, which goes back to 1869, when it was first obtained by Cantor [Can69]. For completeness, we describe it in this section.

Lemma 7.1 *Let N be a positive integer and fix a tuple of positive integers*

$$(n_1, n_2, \ldots, n_\ell)$$

such that $\prod_{i=1}^{\ell} n_i = N$. Let $N_1 = 1$ and for $1 < j \leq \ell$ define $N_j = \prod_{i=1}^{j-1} n_i$. Any integer $x \in \{0, \ldots, N-1\}$ can be written uniquely as

$$x = \sum_{i=1}^{\ell} a_i N_i$$

where we require $a_i \in \{0, \ldots, n_i - 1\}$. Furthermore, if $x \notin \{0, \ldots, N-1\}$ then it cannot be written in this way.

For a given integer x, we say that the tuple $(a_1, \ldots, a_\ell)$ is its **representation** in the **basis** which is defined by the tuple $(n_1, \ldots, n_\ell)$. Notice that the representation of x depends not only on the values $\{n_1, \ldots, n_\ell\}$ but also on their order.

Proof: The proof is by induction on ℓ, the number of factors.
Base case $\ell = 1$: Obvious.
Inductive Step $\ell > 1$: Assume the theorem is true for fewer than ℓ factors. First, we show the existence of a representation for $x \in \{0, \ldots, N-1\}$. Let $a_\ell = \lfloor \frac{x}{N_\ell} \rfloor$ and let $x_\ell = x \bmod N_\ell$. We have that $x = a_\ell N_\ell + x_\ell$. Furthermore, since $x_\ell < N_\ell$ and since $N_\ell = \prod_{i=1}^{\ell-1} n_i$ is the product of $\ell - 1$ factors the induction hypothesis gives a representation

$$x_\ell = \sum_{i=1}^{\ell-1} a_i N_i.$$

Thus,

$$x = \sum_{i=1}^{\ell} a_i N_i.$$

Uniqueness also follows from the induction hypothesis applied to $N_\ell = \prod_{i=1}^{\ell-1} n_i$. The only integers that can be represented by $\sum_{i=1}^{\ell-1} a_i N_i$ are those in $\{0, \ldots, N_\ell - 1\}$. Thus,

if we chose a_ℓ to be anything other than $\lfloor \frac{x}{N_\ell} \rfloor$ it would be impossible to write $x = \sum_{i=1}^{\ell} a_i N_i$. For $i < \ell$, a_i is unique by the induction hypothesis applied to x_ℓ in the $N_\ell = \prod_{i=1}^{\ell-1} n_i$ basis.

Lastly, we must show that if $x \notin \{0, \ldots, N-1\}$ then it cannot be represented as above. This follows immediately from the fact that $a_\ell N_\ell \in \{0, N_\ell, 2N_\ell, \ldots, (n_\ell - 1)N_\ell\}$ and by the induction hypothesis that $\sum_{i=1}^{\ell-1} a_i N_i \in \{0, \ldots, N_\ell - 1\}$. ■

7.3 Decomposing $\{0, \ldots, N-1\}$

For any $N > 0$ consider the multiset $S_N = \{0, \ldots, N-1\}$. We will show that

$$\limsup_{N \to \infty} \frac{\#\mathrm{ppd}(S_N)}{N^t} = \infty$$

for all $t < \zeta^{-1}(2)$.

For each ordered factoring $\prod_{i=1}^{\ell} n_i$ of N into positive integers we can construct sets X and Y satisfying $X + Y = S_N$. Let $N_j = \prod_{i=1}^{j-1} n_i$ as before. Define X to be all those integers x that can be written as

$$x = \sum_i a_{2i+1} N_{2i+1}$$

where, as before, $a_{2i+1} \in \{0, \ldots, n_{2i+1} - 1\}$. That is, X is all those integers whose representation has zeros for the a's with even subscripts.

Similarly, define Y to be all those integers y that can be written as

$$y = \sum_i a_{2i} N_{2i}$$

where $a_{2i} \in \{0, \ldots, n_{2i} - 1\}$.

Lemma 7.2 *For every element $s \in S_N$ there is a unique $x \in X$ and a unique $y \in Y$ such that $s = x + y$. For any $s \notin S_N$ there is no $x \in X$ and $y \in Y$ such that $s = x + y$. That is, $S_N = X + Y$.*

Proof: Let $s \in S_N$. Let $(a_1, \ldots, a_\ell)$ be the representation for s in the mixed radix basis $(n_1, \ldots, n_\ell)$. Define:

$$\begin{aligned} x &= \sum_i a_{2i+1} N_{2i+1} \\ y &= \sum_i a_{2i} N_{2i}. \end{aligned}$$

Then $x \in X$, $y \in Y$, and $s = x + y$. If $s = x + y$ for a different $x \in X$ or $y \in Y$ then there would be a another representation $(a_1, \ldots, a_\ell)$ for s. However, by Lemma 7.1 the representation for s is unique. Thus, these are the only $x \in X$ and $y \in Y$ satisfying $s = x + y$.

Lemma 7.1 also implies that any $s \notin S_N$ cannot be written as the sum of an $x \in X$ and a $y \in Y$. ■

We wish to show that there are many unordered pairs $\{X, Y\}$ such that $S_N = X + Y$. In Lemma 7.3 we show that every ordered factoring of N into integers, where each integer is greater than one, gives a unique unordered pair $\{X, Y\}$.

Lemma 7.3 *Let $N > 1$ be an integer. Let $\prod_{i=1}^{\ell} n_i$ and $\prod_{i=1}^{\ell'} n'_i$ be distinct ordered factorings of N into integers, where each integer is greater than one. Let X and Y be the sets constructed from the first factoring and let X' and Y' be the sets constructed from the second factoring. We have that*

$$\{X, Y\} \neq \{X', Y'\}.$$

Proof: It is sufficient to prove $X \neq Y'$ and $X \neq X'$. Observe that $n_1 > 1$ implies that $1 \in X$. It is represented by the tuple $(a_1 = 1, a_2 = 0, \ldots, a_\ell = 0)$. Also, $n'_1 > 1$ implies that $1 \notin Y'$ because 1 is represented by the tuple $(a'_1 = 1, a'_2 = 0, \ldots, a'_{\ell'} = 0)$. Thus $X \neq Y'$.

Let m be the smallest positive integer for which $n_m \neq n'_m$. Without loss of generality assume that $n'_m > n_m$. Since $\prod_{i=1}^{\ell} n_i = \prod_{i=1}^{\ell'} n'_i$, we have that m is strictly less than ℓ.

The integer z represented by the tuple

$$(a_1 = 0, \ldots, a_m = 0, a_{m+1} = 1, a_{m+2} = 0, \ldots, a_\ell = 0)$$

in the first basis is represented by the tuple

$$(a'_1 = 0, \ldots, a'_{m-1} = 0, a'_m = n_m, a'_{m+1} = 0, \ldots, a'_\ell = 0)$$

in the second basis. If m is odd then $z \in X'$ but $z \notin X$. If m is even then $z \in X$ but $z \notin X'$. Thus, in either case, $X \neq X'$. ■

Define $H(N)$ to be the number of ways to factor N into the product of integers, each greater than one, where the order of factors is significant. The above discussion indicates that

$$\#\mathrm{PPD}(N) \geq \#\mathrm{ppd}(S_N) \geq H(N).$$

7.4 Hille's Result

In [Hil36] it is shown that

$$\limsup_{N\to\infty} \frac{H(N)}{N^t} = \infty$$

for all $t < \zeta^{-1}(2)$. The value is not infinite for all $t \geq \zeta^{-1}(2)$. This proves the result:

Theorem 7.1 *For any multiset S containing exactly one zero, let $\#\mathrm{ppd}(S)$ be the number of unordered pairs $\{X, Y\}$ of sets such that $X + Y = S$. Let $\#\mathrm{PPD}(N) = \sup\{\#\mathrm{ppd}(S) : S$ is a multiset of size N containing exactly one zero$\}$. We have that*

$$\limsup_{N\to\infty} \frac{\#\mathrm{PPD}(N)}{N^t} = \infty$$

for all $t < \zeta^{-1}(2)$ where $\zeta(t)$ is the Riemann Zeta Function and $\zeta^{-1}(2) \approx 1.7286472390$.

Corollary 7.1 *For worst case data sets, the Probed Partial Digest Mapping Problem with an input of N fragment lengths will have $\Omega(N^t)$ solutions for any $t < \zeta^{-1}(2)$.*

7.5 Examples

For every value of $t < \zeta^{-1}(2)$ there must *exist* an infinite sequence $\{N_1, N_2, \ldots\}$ satisfying

$$\lim_{i\to\infty} \frac{\#\mathrm{ppd}(S_{N_i})}{N_i^t} = \infty.$$

We do not have an algorithm to *construct* such a sequence for arbitrary values of t. However, we can construct some sequences $\{N_1, N_2, \ldots\}$ for which the sequence

$$\{\#\mathrm{ppd}(S_{N_1}), \#\mathrm{ppd}(S_{N_2}), \ldots\}$$

grows quickly.

Lemma 7.4 *For $N_i = 2^i$, $H(N_i) = N_i/2 = 2^{i-1}$.*

Proof: This is shown in [Nao90]. The proof is by induction on i.
Base case $i = 1$: $N_1 = 2$ has only one ordered factoring into factors greater than one.
Inductive Step $i > 1$: Assume the theorem is true for values less than i. There is one ordered factoring of N_i with just one factor. For factorings of N_i into two or more factors

the first factor must be in $\{2^1, 2^2, \ldots, 2^{i-1}\}$. If the first factor is 2^j the product of the remaining factors will be 2^{i-j}. The number of ordered factorings of 2^{i-j} is 2^{i-j-1} by the induction hypothesis. $H(2^i)$ is the sum, over every choice of a first factor, of the number of ways to factor the rest. Thus,

$$H(2^i) = 1 + \sum_{j=1}^{i-1} H(2^{i-j}) = 1 + \sum_{j=1}^{i-1} 2^{i-j-1} = 2^{i-1}.$$

■

Lemma 7.5 *For $N_i = 3 \cdot 2^i$, $H(N_i) = \frac{N_i \lg(4N_i/3)}{6} = (i+2)2^{i-1}$.*

Proof: Proof by induction on i.

Base case $i = 0$: $N_0 = 3$ has only one ordered factoring into factors greater than one.

Inductive Step $i > 1$: Assume the theorem is true for values less than i. The first factor of N_i is either N_i, an element of $\{3 \cdot 2^0, \ldots, 3 \cdot 2^j, \ldots, 3 \cdot 2^{i-1}\}$, or an element of $\{2^1, \ldots, 2^k, \ldots, 2^i\}$. Thus,

$$\begin{aligned}
H(3 \cdot 2^i) &= 1 + \sum_{j=0}^{i-1} H(\frac{N_i}{3 \cdot 2^j}) + \sum_{k=1}^{i} H(\frac{N_i}{2^k}) \\
&= 1 + \sum_{j=0}^{i-1} H(2^{i-j}) + \sum_{k=1}^{i} H(3 \cdot 2^{i-k}) \\
&= 1 + \sum_{j=0}^{i-1} 2^{i-j-1} + \sum_{k=1}^{i} ((i-k)+2)2^{i-k-1} \\
&= 1 + \sum_{\ell=0}^{i-1} 2^{\ell} + \sum_{\ell=0}^{i-1} \ell 2^{\ell-1} + \sum_{\ell=0}^{i-1} 2^{\ell} \\
&= 1 + (2^i - 1) + (i2^{i-1} - 2^i + 1) + (2^i - 1) \\
&= (i+2)2^{i-1}
\end{aligned}$$

■

Lemma 7.6 *For any t satisfying $\frac{6^t}{2} - 3^t - 2^t + 1 < 0$ (i.e., $t < 1.4352791$) there exists infinitely many values of N of the form $N = 2^i \cdot 3^j$ satisfying $H(N) = \Omega(N^t)$.*

Proof: For convenience, define $H(1) = 1$. When $N > 1$, we have

$$2H(N) = \sum_{d|N} H(d).$$

There is a factor of two on the left-hand side because $H(N)$ is included on the right-hand side. When $N = 1$, we have $2H(N) - 1 = \sum_{d|N} H(d)$. Define generating functions

$$\begin{aligned} G(s) &= \sum_{\substack{N=2^m \cdot 3^n, \\ m,n \geq 0}} \frac{H(N)}{N^s} \\ Z(s) &= \sum_{\substack{N=2^m \cdot 3^n, \\ m,n \geq 0}} \frac{1}{N^s} \end{aligned}$$

The first equations imply the formal identity, $2G(s) - 1/1^s = G(s)Z(s)$, which implies

$$G(s) = \frac{1}{2 - Z(s)}.$$

This formula implies that the sum defining $G(s)$ must diverge when s satisfies $Z(s) = 2$. Now $Z(s) = (\sum_{m \geq 0} 1/2^{ms})(\sum_{n \geq 0} 1/3^{ns})$, thus $Z(s) = 2$ when $\frac{6^s}{2} - 3^s - 2^s + 1 = 0$. If $H(N)$ were $O(N^t)$ for some $t < s$ then the definition of $G(\cdot)$ would imply that $G(s)$ converges. Thus, $H(N)$ is not $O(N^t)$ for any $t < Z^{-1}(2)$. ■

Note that in the above, $G(t)$ and $Z(t)$ are defined as a sum over all values of N whose prime factorization includes only the first two primes. If instead we use those N whose prime factorization includes only the first k primes we get a tighter lower bound to $H(N)$. If we let $k \to \infty$ we find that we have rederived Hille's result.

Bibliography

[ABK+89] Frederick M. Ausubel, Roger Brent, Robert E. Kingston, David D. Moore, J. G. Seidman, John A. Smith, and Kevin Struhl, editors. *Current Protocols in Molecular Biology.* Green Publishing Associates and Wiley-Interscience, New York, 1989.

[AKNW93] Farid Alizadeh, Richard M. Karp, Lee A. Newberg, and Deborah K. Weisser. Physical mapping of chromosomes: A combinatorial problem in molecular biology. In *Proceedings of The Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, Austin, TX, January 1993. ACM Press.

[AKNW95] Farid Alizadeh, Richard M. Karp, Lee A. Newberg, and Deborah K. Weisser. Physical mapping of chromosomes: A combinatorial problem in molecular biology. *Algorithmica*, 13(1–2):52–76, January–February 1995.

[ALTW91] R. Arratia, E. S. Lander, S. Tavare, and M. S Waterman. Genomic mapping by anchoring random clones — a mathematical analysis. *Genomics*, 11(4):806–827, December 1991.

[AY88] L. Allison and C. N. Yee. Restriction site mapping is in separation theory. *Computer Applications in the Biosciences*, 4(1):97–101, March 1988.

[Bel88] Bernard Bellon. Construction of restriction maps. *Computer Applications in the Biosciences*, 4(1):111–115, March 1988.

[Bev69] Philip R. Bevington. *Data Reduction and Error Analysis for the Physical Sciences.* McGraw-Hill, New York, 1st edition, 1969.

[BJL+94] Avrim Blum, Tao Jiang, Ming Li, John Tromp, and Mihalis Yannakakis. Linear approximation of shortest superstrings. *Journal of the Association for Computing Machinery*, 41(4):630–647, July 1994.

[Bro65] William G. Brown. Historical note on a recurrent combinatorial problem. *American Mathematical Monthly*, 72(9):973–977, November 1965.

[BSP+90] E. Branscomb, T. Slezak, R. Pae, D. Galas, A. V. Carrano, and M. Waterman. Optimizing restriction fragment fingerprinting methods for ordering large genomic libraries. *Genomics*, 8(2):351–366, October 1990.

[Cam84] Douglas M. Campbell. The computation of Catalan numbers. *Mathematics Magazine*, 57(4):195–208, September 1984.

[Can69] G. Cantor. Über die einfachen zahlensysteme. *Z. Math und Physik*, 14:121–128, 1869.

[CAT93] A. J. Cuticchia, J. Arnold, and W. E. Timberlake. ODS: Ordering DNA sequences — a physical mapping algorithm based on simulated annealing. *Computer Applications in the Biosciences*, 9(2):215–219, April 1993.

[CdJB+89] A. V. Carrano, P. J. de Jong, E. Branscomb, T. Slezak, and B. Watkins. Constructing chromosome- and region-specific cosmid maps of the human genome. *Genome*, 31(2):1059–1065, 1989.

[CGL+89] William Chang, Dan Gusfield, Eugene Lawler, Dalit Naor, and Frank Olken. Mapping algorithms for DNA partial digestion: A survey. Technical report, Information and Computing Sciences Division, Lawrence Berkeley Laboratory, May 1989.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, Cambridge, Massachusetts, 1990.

[CNH+90] A. G. Craig, D. Nizetic, J. D. Hoheisel, G. Zehetner, and H. Lehrach. Ordering of cosmid clones covering the Herpes simplex virus type-I (HSV-I) genome — a test case for fingerprinting by hybridisation. *Nucleic Acids Research*, 18(9):2653–2660, May 11 1990.

[DDL+92] Radoje Drmanac, Snezana Drmanac, Ivan Labat, Radomir Crkvenjakov, Aleksandra Vicentic, and Anne Gemmell. Sequencing by hybridization — towards an automated sequencing of one million M13 clones arrayed on membranes. *Electrophoresis*, 13(8):566–573, August 1992.

[DDS+93] R. Drmanac, S. Drmanac, Z. Strezoska, T. Paunesku, I. Labat, M. Zeremski, J. Snoddy, W. K. Funkhouser, B. Koop, L. Hood, and R. Crkvenjakov. DNA sequence determination by hybridization: A strategy for efficient large-scale sequencing. *Science*, 260(5114):1649–1652, June 11 1993.

[DK88] Trevor I. Dix and Dorota H. Kieronska. Errors between sites in restriction site mapping. *Computer Applications in the Biosciences*, 4(1):117–123, March 1988.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B (Methodological)*, 39(1):1–38, 1977.

[DRS72] Peter Doubilet, Gian-Carlo Rota, and Richard Stanley. On the foundations of combinatorial theory (vi): The idea of generating function. *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability*, 6(2):267–318, 1972.

[DSL+90] R. Drmanac, Z. Strezoska, I. Labat, S. Drmanac, and R. Crkvenjakov. Reliable hybridization of oligonucleotides as short as 6 nucleotides. *DNA and Cell Biology*, 9(7):527–534, September 1990.

[EG88] R. B. Eggleton and R. K. Guy. Catalan strikes again! How likely is a function to be convex? *Mathematics Magazine*, 61(4):211–219, October 1988.

[EL89] G. A. Evans and K. A. Lewis. Physical mapping of complex genomes by cosmid multiplex analysis. *Proceedings of the National Academy of Sciences, USA*, 86(13):5030–5034, July 1989.

[Gar76] Martin Gardner. Mathematical games. *Scientific American*, 234(6):120–125, June 1976.

[GK90] Daniel H. Greene and Donald Ervin Knuth. *Mathematics for the Analysis of Algorithms*, volume 1 of *Progress in Computer Science and Applied Logic*. Birkhauser, Boston, 3rd edition, 1990.

[GKP89] Ronald L. Graham, Donald Ervin Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Reading, Massachusetts, 1989.

[GW87] Larry Goldstein and Michael S. Waterman. Mapping DNA by stochastic relaxation. *Advances in Applied Mathematics*, 8:194–207, 1987.

[Hil36] Einar Hille. A problem in "factorisatio numerorum". *Acta Arithmetica*, 2(1):134–144, 1936.

[HSB89] Ute Hochgeschwender, J. Gregor Sutcliffe, and Miles B. Brennan. Construction and screening of a genomic library specific for mouse chromosome 16. *Proceedings of the National Academy of Sciences, USA*, 86(21):8482–8486, November 1989.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA., 1979.

[JM91] J. Jurka and A. Milosavljević. Reconstruction and analysis of human ALU genes. *Journal of Molecular Evolution*, 32(2):105–121, February 1991.

[JWM92] J. Jurka, J. Walichiewicz, and A. Milosavljević. Prototypic sequences for human repetitive DNA. *Journal of Molecular Evolution*, 35(4):286–291, October 1992.

[KAI87] Yuji Kohara, Kiyotaka Akiyama, and Katsumi Isono. The physical map of the whole E. Coli chromosome: Application of a new strategy for rapid analysis and sorting of a large genomic library. *Cell*, 50:495–508, July 31 1987.

[Kar93] Richard M. Karp. Mapping the genome: Some combinatorial problems arising in molecular biology. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, San Diego, CA, May 1993. ACM Press.

[Kla70] David A Klarner. Correspondence between plane trees and binary sequences. *Journal of Combinatorial Theory*, 9(4):401–411, December 1970.

[KN95] Richard M. Karp and Lee A. Newberg. An algorithm for analyzing probed partial digestion experiments. To appear in *Computer Applications in the Biosciences*, 1995.

[LGA+87] Eric S. Lander, Philip Green, Jeff Abrahamson, Aaron A. Barlow, Mark J. Daly, Stephen E. Lincoln, and Lee A. Newberg. *MAPMAKER*: An interactive computer package for constructing primary genetic linkage maps of experimental and natural populations. *Genomics*, 1(2):174–181, October 1987.

[LK73] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2), March-April 1973.

[Lov79] Laszlo Lovasz. *Combinatorial Problems and Exercises*. North-Holland Publishing Company, New York, 1979.

[LW88] Eric S. Lander and Michael S. Waterman. Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics*, 2(3):231–239, April 1988.

[Nao90] Dalit Naor. A note on the number of distinct solutions to the probed partial digestion reconstruction problem. Technical Report CSE - 90 - 40, Division of Computer Science, University of California, Davis, 1990.

[New94a] Lee Aaron Newberg. Finding a minimum-error clone ordering. In preparation, 1994.

[New94b] Lee Aaron Newberg. Finding a most likely clone ordering from oligonucleotide hybridization data. *Genomics*, 21(3):602–611, June 1994.

[New94c] Lee Aaron Newberg. The number of clone orderings. Submitted to *Discrete Applied Mathematics*, 1994.

[NN93] Lee Aaron Newberg and Dalit Naor. A lower bound on the number of solutions to the probed partial digest problem. *Advances in Applied Mathematics*, 14(2):172–183, June 1993.

[Rab89] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, February 1989.

[Ric88] John A. Rice. *Mathematical Statistics and Data Analysis.* Statistics/Probability Series. Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, CA, 1988.

[RS82] J. Rosenblatt and P. D. Seymour. The structure of homometric sets. *Siam Journal of Algorithms and Discrete Mathematics*, 3(3):343–350, 1982.

[SES+87] Cassandra L. Smith, Jason G. Econome, Andrew Schutt, Stephanie Klco, and Charles R. Cantor. A physical map of the Escherichia Coli K12 genome. *Science*, 236:1448–1453, June 12 1987.

[Slo73] Neil James Alexander Sloane. *A Handbook of Integer Sequences.* Academic Press, New York, 1973.

[Spe91] Terence P. Speed. Personal Communication, 1991.

[SS94] Steven S. Skiena and Gopalakrishnan Sundaram. A partial digest approach to restriction site mapping. *Bulletin of Mathematical Biology*, 56(2):275–294, March 1994.

[SSL90] Steven S. Skiena, Warren D. Smith, and Paul Lemke. Reconstructing sets from interpoint distances. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, pages 332–339, Berkeley, CA, June 6-8 1990. ACM Press.

[Ste78] M. Stefix. Inferring DNA structures from segmentation data. *Artificial Intelligence*, 11:85–114, 1978.

[SW91] William Schmitt and Michael S. Waterman. Multiple solutions of DNA restriction mapping problems. *Advances in Applied Mathematics*, 12(4):412–427, December 1991.

[TDMH88] Pierre Tuffery, Philippe Dessen, Claude Mugnier, and Serge Hazout. Restriction map construction using a complete sentences compatibility algorithm. *Computer Applications in the Biosciences*, 4(1):103–110, March 1988.

[TF92] George B. Thomas, Jr. and Ross L. Finney. *Calculus and Analytic Geometry.* Addison-Wesley, Reading, MA, 8th edition, 1992.

[Tur89] J. S. Turner. Approximation algorithms for the shortest common superstring problem. *Information and Computation*, 83(1):1–20, October 1989.

[Wil91] Christopher Wills. *Exons, Introns, and Talking Genes: The Science Behind The Human Genome Project.* BasicBooks, New York, 1991.

[Wil94] Herbert S. Wilf. *Generatingfunctionology.* Academic Press, San Diego, 2 edition, 1994.