Search Algorithms for Biosequences Using Random Projection

Jeremy Buhler

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2001

Program Authorized to Offer Degree: Computer Science and Engineering

University of Washington

Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Jeremy Buhler

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of Supervisory Committee:

_____

Martin Tompa

Reading Committee:

_____

Phil Green

_____

Larry Ruzzo

_____

Martin Tompa

Date: _____

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Bell and Howell Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Search Algorithms for Biosequences Using Random Projection

by Jeremy Buhler

Chair of Supervisory Committee:

Professor Martin Tompa
Computer Science and Engineering

The recent explosion in the availability of long contiguous genomic sequences, including the complete genomes of several organisms, poses substantial challenges for bioinformatics. In particular, algorithms must be developed for annotating biologically meaningful features in multimegabase DNA sequences either by observing their similarity to known genes, regulatory sites, and other features or to conserved copies of the same features in an equally long sequence from another organism. Annotation on such a large scale must be both computationally efficient and sensitive enough to recover subtle but significant features that would otherwise be lost in a mass of unannotated, and hence undifferentiated, sequence.

This work explores algorithms for biosequence annotation that use *random projection*, a technique borrowed from high-dimensional computational geometry. Random projection reduces computationally challenging problems of inexact string matching to a series of more tractable exact matching problems in exchange for a formally quantifiable and practically small loss in sensitivity. Applied to biosequences, the technique permits efficient comparison of very long sequences to discover local alignments corresponding to meaningful features, including some that are practically inaccessible to existing annotation tools. Specific applications that benefit from random projection's increased sensitivity and/or efficiency include comparisons of long orthologous sequences, whole-genome repeat finding, and discovery of regulatory motifs.

Highlights of the thesis include: the LSH-ALL-PAIRS algorithm for discovering all high-scoring ungapped local alignments between pairs of substrings of one or more long sequences, *without* relying on the presence of long exact matches in these alignments; the PROJECTION motif finding algorithm, which extends random projection to a multiple alignment context and discovers motifs that are inaccessible to existing motif finders; and the development of *score simulation*, a theory for building sparse indices or fingerprints of a biosequence such that the probability that two sequences' fingerprints match increases with their similarity under a given alignment score function.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

K-MER: a sequence of length k. For small values of k, Greek prefixes are used, e.g. dimer, trimer, tetramer, pentamer, etc.

AFFINE: for alignment score functions, penalizing gaps of length ℓ with a penalty $-c\ell - d$. d and c are respectively the "gap opening" and "gap extension" penalties.

ALIGNMENT: a correspondence between two or more sequences. Each column of an alignment matches up corresponding bases from each participating sequence. If a base in one sequence does not match any base in the other sequence (because of, e.g., a historical insertion or deletion), that base is matched to a gap character "–".

ALIGNMENT SCORE FUNCTION: a real-valued function $\mathcal{F}$ on sequence alignments that measures the similarity between aligned sequences. A higher score indicates greater similarity.

ALL-PAIRS LOCAL ALIGNMENT: the problem of finding all pairs of sufficiently similar substrings in a collection C of one or more sequences.

AMINO ACID: the building block of protein sequence. Different amino acids are distinguished by their *side chains*, chemical groups attached to the central carbon atom that give the amino acid its properties.

ANNOTATION: the process of finding biologically meaningful features in genomic sequence.

ANTIDIAGONAL: in gapped alignment, a set of cells (i, j) of the dynamic programming matrix for which $i + j$ is constant. An antidiagonal cuts across every diagonal of the matrix and so is orthogonal to an ungapped alignment.

ASSEMBLY ALGORITHM: any algorithm designed to reconstruct a sequence from a collection of overlapping substrings. Assembly is crucial to reconstructing long genomic sequences from the fragments of roughly 700 bases produced by laboratory sequencing machines.

BACKGROUND SEQUENCE: genomic sequence lacking any biologically meaningful feature, usually assumed to be under no selective pressure and, in sufficiently diverged organisms, to be completely uncorrelated.

BANDED SMITH-WATERMAN: a variant of the Smith-Waterman local alignment algorithm in which an alignment may span no more than a fixed number d of diagonals. The limit d is called the *bandwidth*.

BASE: the building block of DNA and RNA sequence; see also *nucleotide*. DNA bases include *A*denine, *C*ytosine, *G*uanine, and *T*hymine; in RNA, thymine is replaced by *U*racil. In genomic sequence, DNA bases occur in complementary pairs, with A pairing with T and C pairing with G.

BINDING SITE: a short DNA or RNA sequence to which a molecule specifically binds. In DNA, the molecule is often a *transcription factor*.

CANDIDATE PAIR: in all-pairs local alignment and related algorithms, a pair of substrings that have passed an initial filtering stage and must be checked to determine if they exhibit high similarity.

CANONICAL: in LSH-ALL-PAIRS, a term describing an ℓ-mer pair that begins with a matching base pair and is immediately preceded by a mismatched base pair.

CHROMOSOME: a large (multi-megabase) discrete piece of DNA, forming part of an organism's genome. In *diploid* organisms, chromosomes occur in homologous pairs which recombine during the formation of *gametes*, or eggs and sperm.

CIS-REGULATORY LOGIC: a set of regulatory sites in genomic sequence that collectively determine a gene's level of expression. A complete functional description of a gene's cis-regulatory logic is rare in the literature; for an example, see Yuh et al.'s description of the *endo16* pathway in sea urchin [106].

CODON: a triplet of bases in DNA or RNA that encodes a single amino acid of a protein. The mapping from codons to amino acids is a function but is not 1:1, since the 64 codons of nuclear DNA encode only 20 amino acids (plus three *stop codons* that act as signals to end translation).

COMPARATIVE ANNOTATION: the process of finding features in genomic sequence by detecting regions of local similarity, which may indicate an evolutionarily conserved function.

COMPLEMENTARY STRANDS: the two polymeric strands of the DNA double helix, held together by hydrogen bonds between complementary A-T and C-G base pairs. Because of base pair complementarity, the sequence of one strand predicts the sequence of the other.

COMPUTATIONAL GENOMICS: an academic discipline at the interface of computer science and molecular biology, devoted to automated analysis and annotation of large amounts of genomic sequence.

CONSENSUS SEQUENCE: in multiple alignments and motifs, a description of aligned sequences by a single sequence χ whose jth position contains that base occurring most frequently in the jth column of the alignment or motif.

CONSERVATION: the maintenance of a sequence with few or no changes over time because of evolutionary pressure, particularly selective pressure against mutations deleterious to its function. Conserved sequences in two organisms or two parts of the same organism can often be identified by their similarity.

DNA: DeoxyriboNucleic Acid, the molecule constituting the genomes of prokaryotes, eukaryotes, and DNA viruses. DNA occurs in cells as two long polymers of covalently linked *nucleotides*, twisted into a double helix and held together by hydrogen bonding between complementary *base pairs*.

DNA TRANSPOSON: an autonomously replicating piece of DNA that can excise itself from the genome and reinsert elsewhere; one of several sources of interspersed repeats.

DIAGONAL: in gapped alignment, a set of cells (i, j) of the dynamic programming matrix for which $i - j$ is constant. An ungapped alignment is confined to a signal diagonal, while gapped alignments can span multiple diagonals.

DOT PLOT: a graphical representation of similarity between two sequences. A diagonal line on the plot represents an ungapped alignment between the sequences, with the extent of the line on the x and y axes respectively indicating the aligned interval in each sequence.

DUPLICATION: a repeated sequence in a genome, probably arising from historical errors during the genome's replication. Duplications include, e.g., tandem repeats and duplicate genes but not transposable elements; contrast with *interspersed repeat*.

ENHANCER: a genomic sequence element that acts to increase transcription of a gene, usually by attracting a transcription factor that directly or indirectly promotes recruitment of the polymerase complex; contrast *repressor*.

ENHANCER REGION: the DNA sequence roughly 200 to 1000 bases upstream of a gene's transcription start site; a common location for transcriptional enhancer and repressor elements other than the core promoter.

EST MATCHING: the process of identifying coding sequence in a genome by its similarity to an *expressed sequence tag*.

EUKARYOTIC: of cells, having a nuclear envelope. Animals, plants, and fungi all possess eukaryotic cells, while bacteria and archaea do not; contrast *prokaryotic*.

EXCLUSION METHODS: a class of similarity search algorithms characterized by deterministic guarantees of sensitivity, e.g. finding every ungapped alignment of 69-mers with at most 23 substitutions. Examples include the Chang-Lawler algorithm [25] and Pevzner and Waterman's double filtration [77]. Contrast randomized algorithms, such as LSH-ALL-PAIRS, and *word matching*.

EXON: contiguous portion of a protein-coding gene's sequence that remains in the mRNA after splicing; contrast *intron*.

EXPRESSED: of a gene, transcribed (and possibly translated) into its active end product.

EXPRESSED SEQUENCE TAG (EST): a short sequence derived by capturing and sequencing a cell's transcribed mRNA. EST's can be useful for identifying unknown genes in genomic sequence as well as for seeing which genes are transcribed in a given cell type.

FALSE NEGATIVE ERROR: in filtering approaches to pairwise similarity search, a significant similarity that produces no candidate pair. In annotation generally, a conserved sequence feature that produces no similarity (e.g. many RNA genes).

FALSE POSITIVE ERROR: in filtering approaches to pairwise similarity search, a candidate pair that does not lead to a significant similarity. In annotation generally, a

similarity not corresponding to any conserved sequence feature.

GAMETES: eggs and sperm; the sex cells of diploid organisms.

GAP: in an alignment, a run of one or more columns in which bases of one sequence are not aligned to any base of another.

GENE: the functional unit of the genome; a DNA sequence encoding an expressed product, usually a protein but occasionally a specialized RNA. Expression of genes is controlled by, among other things, their *cis-regulatory logic*.

GENE PREDICTOR: a program that identifies the locations and extents of genes in unannotated genomic sequence.

GENOME: the complete collection of an organism's DNA; all the sequence that must be copied when a cell replicates.

HOMOLOGOUS: arising from a common ancestral sequence. See *orthologous* and *paralogous*.

INDEL: an mutation in which a base is **in**serted or **del**eted from a sequence.

INTERSPERSED REPEAT: a repeated sequence in a genome arising from a transposable element. Interspersed repeats may be dead or incomplete copies of these elements; contrast *duplication*.

INTRON: contiguous portion of a protein-coding gene's sequence that is removed from its pre-mRNA during splicing; contrast *exon*.

KILOBASE: one thousand bases; abbreviated "kb."

LINE: Long Interspersed Nuclear Element; one of a family of interspersed repeats several kilobases in length arising from autonomously replicating retrotransposons. Common LINE families in human include LINE1 and LINE2.

LINEAR: of alignment score functions, penalizing gaps of length ℓ with a penalty $-c\ell$, that is, a penalty linearly proportional to the gap length. Contrast *affine*.

LOCAL ALIGNMENT: alignment in which only a substring of each sequence, rather than the entire sequence, participates in the alignment.

LOCUS CONTROL REGION: a regulatory region, not in the promoter or enhancer region of a particular gene, that may control the expression of several genes up to tens of kilobases away. See e.g. [35].

MEGABASE: one million bases; abbreviated "Mb."

MESSENGER RNA (MRNA): the sequence of a protein-coding gene, expressed as RNA and destined to be translated into protein by the ribosome. In higher eukaryotes, most newly transcribed mRNAs contain introns that must be removed before translation.

MRNA SPLICING: the process by which the spliceosome removes introns from the pre-mRNA molecule.

MOTIF: a pattern appearing (perhaps with small differences) in multiple sequences. Motifs in genomic sequence often derive from conserved transcription factor binding sites.

MOTIF FINDING: the process of locating the occurrences of a hidden motif in one or more genomic sequences.

MUTATION: a change in a sequence, usually caused on a small scale by insertion, deletion, or replacement of a base.

NUCLEOTIDE: often used synonymously with *base*; technically refers to both the base and the attached sugar molecule that forms a link in the DNA or RNA sugar-phosphate backbone.

ORTHOLOGOUS: of sequences in two or more organisms, deriving from the same sequence in the organisms' evolutionary common ancestor. Contrast *paralogous*.

PARALOGOUS: of sequences in a single organism, deriving from a single ancestral sequence by duplication, as with families of duplicated genes. Contrast *orthologous*.

PERCENT IDENTITY: in a pairwise alignment, the percentage of columns that contain a pair of matching bases, rather than a mismatch or gap.

POSITIONAL CANDIDATE: a gene that is believed to cause or contribute to a certain phenotype (e.g. a heritable disease) based on genetic linkage studies that correlate the phenotype's presence with a particular sequence pattern in or near the gene.

PRE-MRNA: the transcribed RNA sequence of a protein-coding gene prior to splicing out of its introns and other processing that eventually forms a mature mRNA molecule.

PROCESSED PSEUDOGENE: the inactive remnant of a gene's mature mRNA, with introns removed, that has been reverse-transcribed back into the genome.

PROKARYOTIC: of cells, lacking a nuclear envelope. Bacteria and archaea are prokaryotic cells; contrast *eukaryotic*.

PROMOTER: the basal attachment site for the RNA polymerase occurring just upstream of a gene's transcription start site.

PROMOTER REGION: the sequence between 0 and about 200 bases upstream of a gene's transcription start site, containing the basal promoter and possibly one or more regulatory sites.

PROTEIN: a long polymer of covalently linked amino acids. Proteins perform almost all enzymatic and most structural and regulatory functions in living cells; their function is determined by their three-dimensional folded shape.

PSEUDOGENE: an inactive copy (full or partial) of a gene. Types of pseudogene include *processed pseudogenes* arising from reverse-transcribed mRNA, damaged or incompletely duplicated genes that produce only truncated products, and previously active genes that have become inactive through mutation of their regulatory sites.

PURINE: an A or G base, denoted R in sequences.

PYRIMIDINE: a C or T base denoted Y in sequences.

REGULATORY MECHANISM: any means by which a cell controls the expression of its genes. Common regulatory mechanisms include control of transcription, selective degradation of RNA and protein, and RNA structures such as hairpins that affect the rate of translation.

REGULATORY SITE: any site in genomic sequence that affects the expression of a nearby gene. Regulatory sites are typically *binding sites* for transcription factors or other proteins.

REPETITIVE ELEMENT (REPEAT): any sequence that occurs multiple times in a genome. Types include *duplications* and *interspersed repeats*.

REPRESSOR: a genomic sequence element that acts to decrease transcription of a gene, usually by attracting a transcription factor that directly or indirectly hinders recruitment of the polymerase complex; contrast *enhancer*.

RETROTRANSPOSON: a transposable element that replicates by transcribing itself into RNA that is later reverse transcribed back into DNA elsewhere in the genome.

REVERSE TRANSCRIPTASE: an enzyme that reads an RNA sequence and constructs the complementary DNA sequence; used by retroviruses and retrotransposons for replication.

RIBOSOME: a macromolecular complex that translates mRNA into protein.

RIBOSOMAL RNA (RRNA): one of several RNA molecules that form part of the structure of the ribosome.

RNA: RiboNucleic Acid, the molecule constituting the intermediate between a gene's DNA sequence and its protein product, as well as certain special purpose molecules such as tRNAs and rRNAs. RNA occurs in cells as single-stranded polymers of covalently linked *nucleotides*.

RNA POLYMERASE: a macromolecular complex that transcribes DNA into RNA.

SEQUENCE FEATURE: a substring of a genomic sequence with an identifiable present or past function.

SIMILAR: possessing approximately the same sequence. Similar genomic sequences may or may not be *conserved*; similarity is a property of strings in general, while conservation implies an evolutionary process at work.

SINE: Short Interspersed Nuclear Element; one of a family of interspersed repeats a few hundred bases in length arising from short retrotransposons that copy themselves with the help of a LINE. Common SINE families in human include the ubiquitous *Alu* and the less common MIR.

SPLICEOSOME: a macromolecular complex that splices introns out of the pre-mRNA.

STOP CODON: one of three codons that signal the end of an mRNA's coding sequence to the ribosome, causing it to stop translation.

SUBSTITUTION: a mutation in which one base changes into another.

SUBTLE: of a motif, having mutations distributed uniformly across all positions, rather than concentrated in a few highly variable positions. Subtle motifs are generally not found by existing motif finders.

SYNONYMOUS: of DNA or RNA codons, encoding the same amino acid. Amino acids have anywhere from one to six synonymous codons.

TANDEM DUPLICATION: two or more copies of the same genomic sequence occurring one after the other; thought to arise when the DNA polymerase "slips" during replication of the genome.

TRANSCRIBED: of genomic sequence, being copied as RNA by the RNA polymerase complex.

TRANSCRIPTION FACTOR: a protein that modifies the rate at which a gene is transcribed. Transcription factors often work by directly or indirectly contacting the RNA polymerase complex, but they may also work in other ways, e.g. by temporarily bending the DNA near a gene.

TRANSFER RNA (TRNA): a specialized non-coding RNA that transports amino acids to the ribosome and matches them to their corresponding codons in the mRNA during translation.

TRANSITION: a substitution mutation in genomic sequence that changes one purine to another or one pyrimidine to another; contrast *transversion*.

TRANSLATED: of mRNA, being decoded into protein by the ribosome.

TRANSPOSABLE ELEMENT (TRANSPOSON): a DNA sequence that can copy or move itself elsewhere in the genome, either autonomously or with the aid of another transposon. Types include *retrotransposons* and *DNA transposons*.

TRANSVERSION: a substitution mutation in genomic sequence that changes a purine to a pyrimidine or vice versa; contrast *transition*.

UNGAPPED IDENTITY: *percent identity* when the alignment is ungapped.

UNTRANSLATED REGION (UTR): a transcribed, exonic portion of a protein-coding gene that is not part of the coding sequence. Most genes have two UTR's, one upstream ($5'$) and one downstream ($3'$) of the coding sequence.

WORD MATCH: a contiguous run of matching bases between two sequences, with no intervening substitutions or indels.

ACKNOWLEDGMENTS

This version of the thesis with corrected errata was generated on August 28th, 2001.

The author would like to thank his reading committee for proofreading above and beyond the call of duty, as well as his cat, Figaro, for moral support throughout the thesis-writing process. Thanks also to Lee Hood and the Institute for Systems Biology for providing much needed computing cycles.

This work was supported in part by a Fannie and John Hertz Fellowship.

DEDICATION

To my parents, who don't care whether I mention them in my dissertation... really!

Chapter 1

BACKGROUND: EXPLAINING THE GENOME

As the 21st century begins, large-scale genomics has become a fundamental tool for understanding an organism's biology. Access to multiple complete genomic sequences helps biologists to formulate and test hypotheses about how genomes are organized, how that organization evolved, and how a genome encodes the observed properties of a living organism. Key questions being pursued include: what parts of our genome encode the mechanisms for major cellular functions like metabolism, differentiation, proliferation, and programmed death? How do multiple genes act in concert to carry out these and other, more specialized functions? How is our non-protein-coding DNA organized, and what parts of it are functionally important? How do selective pressures act on the random processes of gene duplication and mutation to give rise to complex constructs like eyes, wings, and brains? Why do humans appear so different from worms and flies, despite sharing so many of the same genes?

Until the 1990's, molecular biologists could pursue questions about the content and function of genomes only indirectly, or else at great cost. Indirect techniques such as Giemsa staining and CoT-based measurement of repetitive content [97] provided such global information as was available about a genome. Full sequence was available for only a few short regions found to be functionally significant, usually after a long and expensive process of localization by (e.g.) linkage mapping, followed by cloning out and finally sequencing a minimal region of interest. The cost and time required to sequence DNA made sequencing a tool to be applied only at particular points, and only once a region was shown to be important by other means.

More recently, high-throughput DNA sequencing has enabled a direct approach to study-

ing genomes. Using this new technology, biologists have obtained progressively larger complete genomic sequences, from viruses [84] to prokaryotes [34] to single-celled [65] and multicellular [24] eukaryotes. Available genomes today include those of several higher metazoans, including the fruit fly *Drosophila melanogaster* [2], the flowering plant *Arabidopsis thaliana* [7], and, of course, *Homo sapiens* [101, 37]. Armed with substantially complete euchromatic sequence from these organisms, we can now directly interrogate global properties like base frequencies and repetitive content, obtain immediately the sequence of any potentially interesting region, and – perhaps most exciting – compare corresponding long stretches of genomic DNA in two or more organisms. Such analyses encompass massive amounts of sequence, on a scale requiring computation that defies manual analysis. The need to automate analysis of long or numerous genomic sequences gives rise to the field of *computational genomics*.

In this work, we address a particular problem of computational genomics: how to discover which parts of a long DNA sequence encode particular biological features, such as genes. Even when the whole sequence is available for inspection, finding these features reliably can be surprisingly difficult. If we know little about the features being sought, or their presence leaves only a weak imprint on the underlying sequence, finding them may be theoretically intractable or practically beyond our limited budget of computing time and space. This work focuses specifically on new techniques to find features that are difficult to find in theory or simply intractable to existing search algorithms.

The algorithms that we introduce in this thesis are founded on a common technique, *random projection*, which exploits the intuition that two nearly identical biosequences cannot differ in many of their characters, regardless of where the differences occur. We show how to apply this intuition to lower the cost barriers that previously made certain types of sequence features difficult to find efficiently. As a result, we can more readily identify more interesting features and ultimately provide more complete information to the working biologist.

1.1 Road Map to the Thesis

We begin by providing the reader with a brief guide to the content of the thesis. Some readers may find the biological terms used in subsequent sections and chapters unfamiliar; hereafter, we will both define such terms at their point of first use and provide a glossary of terms to collect the important definitions in one place.

- Chapter 1 is devoted to background and significance. We first review the nature of genomic DNA and the kinds of meaningful features it contains, then discuss the fundamental problem of discovering, or *annotating*, these features. We then describe the idea of comparative annotation based on sequence similarity, discuss its strengths and limitations, and finally focus on the computational problems that arise from it.

- Chapter 2 addresses the *all-pairs local alignment* problem for long genomic sequences. After stating the formal problem and some of its uses, we introduce the basic theory of random projection that we use to address it. Using this theory, we develop the LSH-ALL-PAIRS algorithm, which efficiently solves the problem for the case of ungapped local alignments even with quite low similarity. We then show how to implement the algorithm in practice and validate our implementation on a collection of interesting biosequences.

- Chapter 3 undertakes a more ambitious multiple alignment problem, that of discovering conserved regulatory motifs. We first establish a model for *subtle* motifs that are intractable to standard search algorithms, then introduce the PROJECTION motif finding algorithm to find such motifs. We then validate our algorithm and illustrate how it improves on existing heuristic methods for motif finding.

- Chapter 4 describes ways to extend matching techniques based on random projection to find similar sequences under more general criteria than a simple count of matching characters.

1.2 Biological Background: DNA and Sequence Features

The first prerequisite to developing algorithms to find features in genomic sequence is to understand *what* we are looking for and why. We therefore begin with a brief review of genomic DNA and its major features. Readers seeking more background on genomic DNA or on molecular biology in general may wish to consult the standard text by Lewin [60] or the gentler introduction by Gonick and Wheelis [39].

1.2.1 DNA and Genomic Sequence

The fundamental substrate for genome analysis is *DNA*, or *DeoxyriboNucleic Acid*. DNA is a polymeric molecule composed of a sequence of *nucleotides*, also called *bases*[1]. Four such bases – A, C, G, and T– form the alphabet[2] from which all natural DNA is constructed. Abstractly, a *DNA sequence* is simply a string over the alphabet $\{A, C, G, T\}$. We will use the terms "string" and "sequence" interchangeably.

DNA normally exists as two *complementary strands* made up of a sequence of *base pairs*. The pairing is deterministic: A always pairs with T, while C pairs with G. Thus, the sequence of one strand determines the sequence of its complement, and we can describe a DNA sequence uniquely by only one of its strands. Because of this pairing, bases are sometimes classed as "weak" (A/T, joined by two hydrogen bonds) or "strong" (C/G, joined by three hydrogen bonds). Another common classification of bases, this time by chemical structure, is as *purine* (A/G) or *pyrimidine* (C/T). An unspecified purine or pyrimidine is denoted by the characters R and Y respectively.

DNA molecules are found in the cytoplasm of *prokaryotic* cells (the bacteria and archaea) and in the nuclei of *eukaryotic* cells (everything else). A cell's total complement of DNA sequence is its *genome*; in multicellular organisms, essentially all cells contain the same genome, with all except the *gametes* (eggs and sperm) carrying two copies. The differences in genomic sequence from one organism to another within a species are quite small compared

[1] Abstractly, "base" and "nucleotide" are used synonymously to refer to characters of the DNA alphabet. Chemically, however, a DNA nucleotide is actually a base joined to a 2-deoxyribose sugar molecule that forms part of the backbone of the double helix.

[2] The letters are drawn from the chemical names of the bases: adenine, cytosine, guanine, and thymine.

Table 1.1: Genome sizes of model organisms commonly studied by biologists. For eukaryotes, chromosome number and size are given for the haploid genome.

Organism	Common Name	# Chromosomes (haploid)	Genome Size (megabases)
E. coli	(a bacterium)	1	4.4
S. cerevisiae	baker's yeast	16	12
C. elegans	(a roundworm)	6	100
D. melanogaster	fruit fly	4	120 (euchromatic)
A. thaliana	thale cress	5	125
M. musculus	mouse	20	∼3000
H. sapiens	human	23	∼3000

to the differences between species, so it makes sense to talk about an entire species' genome. For example, the human genome, which is 3×10^9 base pairs in length, is 99.9% similar between individuals, while the genome of our closest relative, the chimpanzee, is only 98–99% similar to ours [26].

An organism's genome is organized into a small number of discrete DNA molecules, called *chromosomes*. Bacteria typically have a single, circular chromosome a few million bases in length, while eukaryotic species have anywhere from three to over 100 linear chromosomes of total length ranging from tens of millions up to billions of bases. Table 1.1 lists the genome sizes of some model organisms commonly studied by biologists. A genome's division into chromosomes does not vary within a species, so it makes sense to number chromosomes, say in decreasing order by length, and then to speak of e.g. "human chromosome one."

An essential feature of DNA is that it is not static over time. Chemicals, radiation, and copying errors can all cause a DNA sequence to *mutate*. Biologically common types of mutation include *substitutions*, in which one base is replaced by another[3], and *indels* (in-

[3] "Substitution" is sometimes used in biology to describe *any* uncorrected mutation. In this work, it refers specifically to replacement of one base by another.

sertions and deletions), in which bases are added to or removed from a sequence[4]. Different types of mutation happen at different rates; for example, *transition* substitutions – those that replace A with G or C with T and vice versa – are roughly twice as common [96] as other substitutions, which are called *transversions*.

1.2.2 The DNA Zoo: Features Found in Genomic Sequence

While we can think of a genome abstractly as a long string of bases, such an abstraction is not very useful for thinking about its functional role in the cell. A better model might be a series of beads on a string. Each bead represents a *sequence feature*, a short substring of the genome to which we can ascribe some biological interpretation or function. Not every base in a genome falls within the boundary of a feature; we will refer to those parts of a sequence that form the "string" separating features as *background sequence*.

The sequence features we will consider fall broadly into three categories: *genes*, which encode the active molecules that carry out the cell's business; *regulatory sites*, which control the behavior of genes; and *repetitive elements*.

Genes

Genes, the primary functional component of genomic sequence, encode instructions for building other polymeric molecular species. A gene's basic function is to have its DNA sequence *transcribed* into a corresponding (single-stranded) polymer of *RNA*, or *RiboNucleic Acid*. The sequence of an RNA molecule is identical to that of its originating gene, except that T bases are mapped not to T but rather to a different base, U (for uracil). As shown in Figure 1.1A, genes are transcribed by a large molecular complex called an *RNA polymerase* that sequentially reads the DNA sequence and strings together free-floating RNA nucleotides in the cell to reproduce that sequence as RNA.

Most genes produce *messenger RNAs* (*mRNAs* for short) that are destined to be *translated* into *protein*, a polymer over an alphabet of twenty *amino acids*. Translation into protein is carried out by another large molecular complex, the *ribosome*. Unlike DNA-to-

[4]We ignore large-scale shuffling and duplication of blocks of the type considered in genome rearrangement.

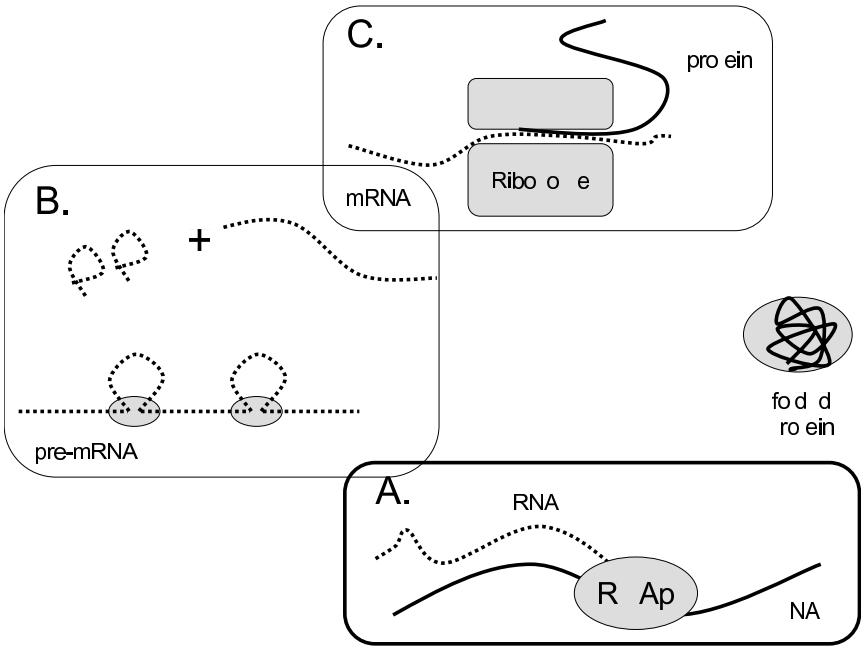

Figure 1.1: processing of genomic coding sequence into protein. (A) gene sequence is transcribed by RNA polymerase (RNAp) to form pre-mRNA; (B) introns are spliced out of pre-mRNA by spliceosome, producing a processed mRNA; (C) final mRNA is translated by the ribosome to produce a protein, which folds into its active configuration. DNA and protein are shown as solid lines; RNA, as dotted lines.

RNA transcription, which is a 1:1 mapping, translation maps successive groups of three bases – called *codons* – to a single amino acid, as shown in Figure 1.1C. The latter mapping is degenerate, with some amino acids encoded by as many as six different but *synonymous* codons. The sequence of a protein determines its ultimate three-dimensional shape, which in turn determines its activity in the cell.

Not every base of an mRNA is interpreted as coding for protein. The mRNA typically starts well before its first codon and extends past its last codon. Within this sequence, the coding region is delimited by the occurrence of particular codons, usually AUG at the beginning and one of three *stop codons* at the end. Moreover, eukaryotic genes are frequently discontiguous: they exist in the genome as a series of *exons* that are separated by intervening *introns*. The entire sequence of the gene, both introns and exons, is transcribed into a *pre-mRNA*, after which the introns in the RNA are removed and the remaining exons joined to form the processed mRNA as shown in Figure 1.1B. This process of *mRNA splicing* is carried out by yet another large molecular complex, the *spliceosome*, which determines where to cut the pre-mRNA by recognizing particular sequence patterns that serve as signals for the splicing boundaries between intron and exon.

A few genes' RNAs are never translated into protein; instead, they are biologically active molecules by themselves. Such RNAs include the *transfer RNAs* (*tRNAs* for short), which match specific amino acids to their RNA codons during translation, as well as various RNA components of the ribosome and other molecular complexes in the cell. In rare cases, even mRNAs can actually be reverse-transcribed back into the genome as DNA, littering the genome with (generally nonfunctional) artifacts called *processed pseudogenes*.

Returning to the primary genomic DNA, it is clear that genes must be highly distinctive sequence features. Coding exons in particular have a biased sequence composition compared to the background sequence: their composition depends on the usage frequencies of amino acids in proteins, and they cannot contain internal stop codons. Moreover, some codons for a given amino acid are used much more often than others, further biasing the observed composition. Most genes also exhibit a number of distinctive signal sequences, including the aforementioned splicing signals at exon boundaries and signals that indicate where the RNA polymerase should start and stop transcribing the gene. These distinctive

properties of genes are exploited by computational *gene predictors*, such as Green and Wilson's GeneFinder [103] and Burge and Karlin's GenScan [23], to identify genes in raw DNA sequence.

Finally, we note that for all their functional importance, genes constitute only a small part of the genomic sequence of higher eukaryotes. In particular, coding sequences make up only about 1.4% of the human genome [101, Table 11].

Regulatory Sites

An organism does not use the product of every gene in its genome at all times or in all its various tissues. For instance, every nucleated cell in the human body contains the gene for making insulin, but (unfortunately for sufferers of type I diabetes) only a few cells in the pancreas actually produce it. Similarly, the genes for proteins involved in cell division are always present, but their products are only needed during the brief time in the cell cycle when division actually occurs. Cells therefore have *regulatory mechanisms* for controlling when and where genes are *expressed* to produce their products.

In this work, we focus on one particular form of regulation: control of gene transcription by a class of proteins called *transcription factors*. These proteins adhere to genomic DNA at *binding sites*, regions up to a few tens of bases in length that contain factor-specific signal sequences. Transcription factors often bind at sites within a few hundred bases of the start of a gene, its so-called *promoter* and *enhancer regions*, where they influence how frequently the RNA polymerase complex initiates transcription of that gene. If a transcription factor causes the gene to be expressed at a higher level, it is said to be an *enhancer*; if it causes a lower level of expression, it is a *repressor*.

Transcription factors are often activated in response to changes in the cell's environment, especially changes in the amounts of various chemicals (including other gene products) present. These proteins can therefore orchestrate the cell's transcriptional response to changing external conditions as well as carrying out "programs" such as cell division, differentiation, or death in response to particular chemical signals. The exact mechanism by which transcription factors transduce these changes varies. Many factors form (or block

formation of) protein complexes that contact the RNA polymerase directly, increasing or decreasing its affinity for binding to a gene's promoter and initiating transcription [60, Chapter 28]. Factors may also alter the conformation of the DNA to which they bind, again changing the binding affinity of the polymerase [86, 87].

Multiple transcription factors can act on a single gene, in which case several different binding sites may cluster near that gene. The factors' actions are not necessarily independent; in general, they may form a complex *cis-regulatory logic* that permits fine control over when and how strongly a gene is expressed. At this time, few examples of cis-regulatory logic have been worked out in detail; the work of Yuh et al. in sea urchin development [106] illustrates the complexity possible in such logic.

Transcription factor binding sites, while clearly important as sequence features, are unfortunately difficult to identify in raw genomic sequence. We know that sites are likely to occur in clusters in the promoter regions of genes, typically within a few hundred to a few thousand bases of the transcription start site. However, significant sites may be found elsewhere, including the introns of genes [72] and *locus control regions* that may be ten *kilobases* (ten thousand bases) or more away from the genes they regulate [35]. In general, we cannot assume much *a priori* about what binding sites look like – their sequence patterns are too dependent on the particular factor that they bind. Certain types of transcription factor may require binding sites with known structure, such as a DNA palindrome for some homodimeric factors, but such structures are far from universal.

Finally, we note that even if all the sites for a given transcription factor had identical sequence (which is not the case), the sequence pattern is usually short enough that it may occur purely by chance in the background sequence, at a place where no protein actually binds. Programs to find new transcription factor binding sites in genomic sequence are therefore challenged not only by a lack of identifying characteristics for these sites but also by confusion between true binding sites and chance occurrences of their sequence patterns.

Repetitive Elements

Over 90% of the sequence in most bacterial genomes is part of a gene or a regulatory site, but a full accounting of these features in the human genome, or even the genome of a fly or worm, would still leave most of the sequence unexplained. Much of the noncoding sequence in these organisms arises from a third kind of feature: *repetitive elements*, or *repeats* for short.

A repetitive sequence or element is any DNA sequence that occurs multiple times (perhaps with mutations) in a genome. Repetition is pervasive in the genomes of higher eukaryotes, and its causes are legion. We may broadly divide repetitive elements into *duplications*, including both functional (e.g. duplicate genes) and nonfunctional duplications, and transposon-derived *interspersed repeats*.

Duplications are a mixed bag of features, probably arising from historical errors in copying the genome. *Tandem duplications* occur when a piece of DNA is copied twice in a row, perhaps as a result of "stuttering" or "skipping" by a DNA polymerase. They include short tandem repeats, in which a pattern like "CAT" is copied anywhere from a few to a few hundred times, as well as longer tandem repeats of just two or a few copies, each tens to hundreds of bases in length. Larger, non-tandem duplications are also known. For example, human chromosome 22 contains the LCR-22 repeat family [33], each copy of which is tens of kilobases long, while human chromosome 21 is known to share duplicated regions of 50 to 100 kb with chromosomes 4, 7, 16, 20, and 22 [43].

Duplication events that include gene sequences are a source of pseudogenes, but they can also create new active genes whose functions diverge from the originals. Some important genes (both active and pseudo) arising from historical duplications include vertebrate T-cell receptors [20] and MHC genes; in both of these gene families, historical duplication followed by mutation created groups of diverse receptor molecules that let the immune system recognize a wide variety of pathogens.

Interspersed repeats arise by a much different process than duplications. They are the detritus of *transposable elements*, DNA sequences that propagate themselves autonomously through the genome [90, 91]. These elements include the *DNA transposons*, which can

excise and reinsert themselves into the genome directly, and the *retrotransposons*, which copy themselves via an RNA intermediate using the enzyme *reverse transcriptase*. Most interspersed repeats are "dead" transposable elements that either were truncated during insertion into the genome or have been rendered inactive by mutations.

While there exist numerous families of transposable elements, we will mention only two types of retrotransposon that are well-known from the human genome and are responsible for most of the interspersed repeats found there. *Long interspersed nuclear elements*, or *LINEs*, are retrotransposons several kilobases in length that replicate autonomously, producing their own reverse transcriptase protein. LINE families occurring in human include the LINE1s and the older LINE2s. The evolutionary origins of both families predate the mammalian radiation, but only the LINE1 appears to have any active members remaining; the LINE2s have all suffered fatal mutations. Each of the LINE families has a corresponding *SINE (short interspersed nuclear element)* – a much shorter, non-autonomous element that uses the LINE's machinery to copy itself[5]. The human SINE corresponding to LINE1 is the *Alu*, easily the most prolific repeat in the human genome at hundreds of thousands of copies[6]. The SINE corresponding to LINE2 is the MIR element, inactive now that its LINE is dead.

Overall, repetitive elements, and interspersed repeats in particular, account for a large part of the sequence of eukaryotic genomes – more than 40% of the noncoding sequence in human [91]. Some of these repeats are recently copied and so are easy to identify, but older repeats that have diverged by many mutations are more challenging. Today, the primary strategy used to find interspersed repeats is first to locate several copies by their similarity to each other, then to build a *consensus sequence* from these copies that can be used to match other members of the same repeat family. Repeat finding tools, including Smit and Green's RepeatMasker [92] and Bedell, Korf, and Gish's Maskeraid [13], use libraries of standard repeat sequences constructed in this way.

---

[5] As Jonathan Swift observed: "So, naturalists observe, a flea / Has smaller fleas that on him prey; / And these have smaller still to bite 'em; / And so proceed ad infinitum."

[6] The mouse genome also contains LINE1, but the corresponding SINE there is the B1 element. Both *Alu* and B1 are recent ($<$ 100 million years diverged) descendents of the same ancestral RNA gene.

1.3 Finding Sequence Features by Comparative Annotation

We now progress from describing the genome to the vital problem of *annotation* – how to identify features in raw DNA sequence. This work concentrates on algorithms for the *comparative* approach to annotation, which has substantial advantages over competing approaches in characterizing certain difficult-to-find features. In this section, we briefly outline the principle of comparative annotation and describe a formal basis for comparing DNA sequences, then proceed to an example that motivates a more detailed discussion of the merits and limitations of the comparative approach.

1.3.1 The Principle of Comparative Annotation

Comparative annotation identifies features on the basis of their *conservation*, or lack of change, over evolutionary time. Although all DNA sequences are subject to mutation, natural selection ensures that we observe today only those individuals whose ancestors' reproductive fitness was not limited by strongly deleterious mutations. Many mutations to genes or regulatory elements can render them dysfunctional, causing the organism carrying these mutations to die or to have fewer viable offspring. In contrast, mutations in nonfunctional sequence can accumulate freely with no effect on reproductive fitness. We therefore expect that the organisms we see today exhibit fewer mutations, or equivalently more conservation, in their functional sequences than in their background sequence. Similarly, many mutations can inactivate a transposon, preventing it from copying itself; hence, the most prolific interspersed repeats that we can see today were reproduced from transposons that accumulated relatively few mutations.

We say that two sequences are *similar* if they have few differences between them. Because sequence features are more strongly conserved than background sequence, two distinct copies of a feature are more likely to remain similar over time. For example, a comparison of sequences from two sufficiently diverged organisms, such as human and mouse, can pick out features such as genes and regulatory regions because the features appear more similar across organisms than the background. Similarly, a comparison between two sequences from mouse can identify recently inserted repetitive elements by their high similarity compared

to the background. **Searching for similar sequences is a useful way to detect and annotate biologically meaningful features in genomic DNA.**

Alignment is a Quantitative Measure of Similarity

If we are to take advantage of sequence conservation to find biologically meaningful features, we must define a precise, quantitative measure of similarity between sequences. Such a measure follows from the idea of a sequence *alignment*.

Suppose that some ancestral DNA sequence s_0 evolves by mutation along two separate lineages, creating present-day sequences s_1 and s_2. If we knew the entire mutation history of s_1 and s_2, we could match up those bases in each sequence that derive (perhaps with substitutions) from the same ancestral base of s_0. Figure 1.2 shows such a matching, or *alignment*, of two sequences, written as a series of columns in which bases deriving from the same ancestor appear in the same column. If, as in this example, the sequences are subject to indels, the alignment contains *gaps* (represented in the figure by columns containing dashes "–") where bases in one sequence do not correspond to any part of the other sequence.

The history of modern genomic sequence is unknown, so we cannot find a true alignment as described above. However, we can plausibly guess at the true matching of bases by finding a *parsimonious* alignment, i.e. one that explains the observed differences between sequences using the minimum number of mutation events. Because different types of mutation occur with different probabilities in nature, we evaluate an alignment not by strict parsimony but by an *alignment score function* that assigns differing costs to substitutions and indels.

An alignment score function is a mapping $\mathcal{F}$ from sequence alignments $\mathcal{A}$ to real numbers (in practice, to integers) that assigns higher scores to alignments requiring fewer or less expensive mutations. A *linear* score function is computed as a sum over the columns of an alignment of a column score function $\sigma(x, y)$, where x and y range over the characters of sequence together with the gap character "–". For example, a simple linear score function would assign a bonus of $+a$ for a match, a penalty of $-b$ for a mismatch, and a larger penalty $-c$ for a gap. In practice, score functions are more often *affine*, penalizing gaps of length ℓ with a penalty $-c\ell - d$. Affine score functions cannot in general be described as a

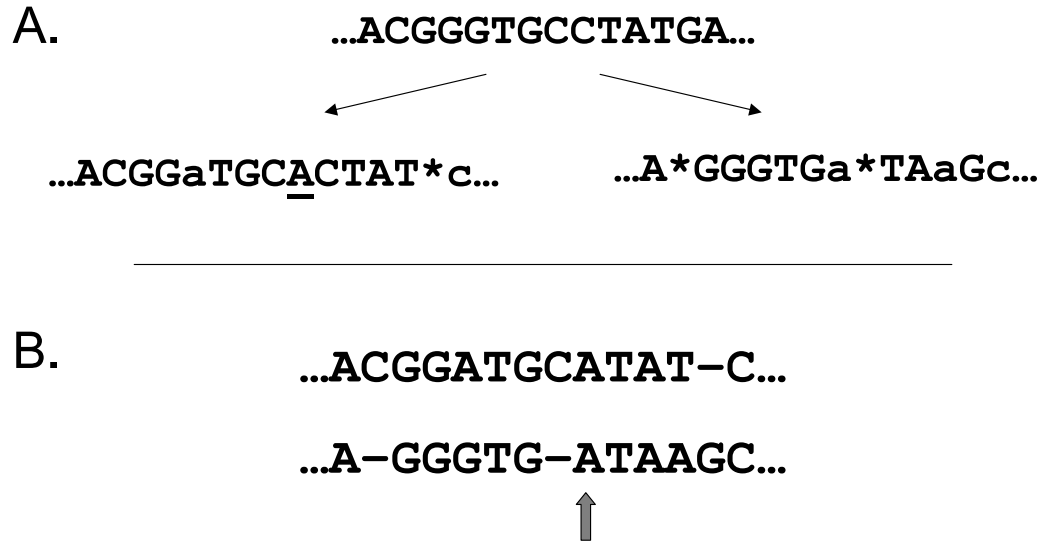

Figure 1.2: example of a parsimonious alignment between two DNA sequences with a common ancestor. (A) the true history of each modern sequence: lower-case letters indicate substitutions vs. the ancestral sequence, while underlined bases and "*" respectively indicate insertions and deletions. (B) a parsimonious alignment of the modern sequences. Note that the best alignment of the central part of the sequences is historically incorrect; the two *A* bases in column nine (indicated by arrow) do not derive from the same ancestral base.

simple sum of column scores but behave identically to linear score functions on alignments without gaps.

We can now use alignment to define similarity precisely. Let s_1 and s_2 be sequences as described above, and fix a score function $\mathcal{F}$. Let $\mathcal{A}$ be an *optimal* alignment (i.e. one of maximum score) between the two sequences. Then the *pairwise similarity of* s_1 *and* s_2 is defined to be $\mathcal{F}(\mathcal{A})$. For both linear and affine alignment score functions, an optimal alignment between two sequences can be computed using improved forms of the Needleman-Wunsch dynamic programming algorithm [69] in time proportional to the product of the sequence lengths. The optimal alignment score is thus a quantitative, as well as a biologically plausible, measure of similarity.

The alignment-based notion of similarity has one major flaw for our purposes: it is defined with respect to two *complete* sequences. In contrast, we have seen that long genomic sequences usually consist of short, well-conserved features in a background that is either wholly unrelated or so ill-conserved as to be unalignable. Because we are concerned only with features, not with background sequence, our similarity measure should ideally ignore the background sequence and measure only the similarity between features. Although we cannot separate the features from the background *a priori*, we can use *local alignment*, in particular the Smith-Waterman dynamic programming algorithm [94], to find the best-aligning pair of substrings between two sequences. Because background sequences are unlikely to align well, Smith-Waterman should exclude them from the optimal substring alignment, leaving only the better-conserved features' sequences to contribute to the alignment score.

Simplifications of the alignment score provide other computationally useful notions of similarity. One common measure is *percent identity*, defined as the percentage of columns in an optimal alignment that contain matching bases. This measure is typically applied to *ungapped* alignments, in which indels are not allowed; for such alignments, the *ungapped identity* score is equal to one minus the fraction of mismatched bases. We will make heavy use of ungapped identity in developing our new annotation algorithms.

1.3.2 An Example of Comparative Annotation

We now provide a concrete example to illustrate the utility of comparative annotation for finding sequence features. Figure 1.3 compares human and mouse genomic sequence at the Bruton's tyrosine kinase (BTK) locus, a stretch of roughly 80 kilobases known to contain several genes and studied in detail by Oeltjen et al.[72]. The human and murine lineages diverged 80-100 million years ago, at the time of the mammalian radiation, so roughly 2×10^8 years separate the individuals from whom these sequences were taken. We have removed *Alu*s from the human sequence using RepeatMasker to eliminate (uninteresting) matches between them and any mouse B1 repeats. The figure shows the sequence comparison in the form of a *dot plot*. Diagonal lines on the plot represent ungapped alignments between intervals of the two sequences; the extent of a line on the x and y axes respectively indicate the aligned intervals in the human and mouse sequences. The alignments shown here were produced by the LSH-ALL-PAIRS algorithm described in Chapter 2, using an empirically derived similarity measure described in [85].

The great majority of the local alignments shown in the figure occur between corresponding exons of four genes: FTP3, GLA (called AGS in mouse), L44L, and the BTK gene itself[7]. These genes are quite well conserved (81-92% base identity between corresponding RNAs), so it is not surprising that similarity search at this level found every known exon at the locus. The observed high similarity between exons frequently extends well past their ends into nearby intronic sequence. Some of these extended alignments probably reflect conserved signals recognized by the mammalian splicing machinery. However, the BTK gene in particular exhibits unusually large regions of similarity in its first, fourth, and fifth introns; Oeltjen et al. found that parts of these intronic regions appear to function in regulating the transcription of BTK. Other evidence of conservation in noncoding sequence includes substantial similarity in the promoter region of each gene, which includes stretches identifiable as known transcription factor binding sites.

The BTK locus example demonstrates the utility of sequence comparison for discovering

[7]The BTK gene gives its name to the locus because it is medically important – a defective BTK causes the immune disorder agammaglobulinemia. The locus was originally sequenced to study this gene.

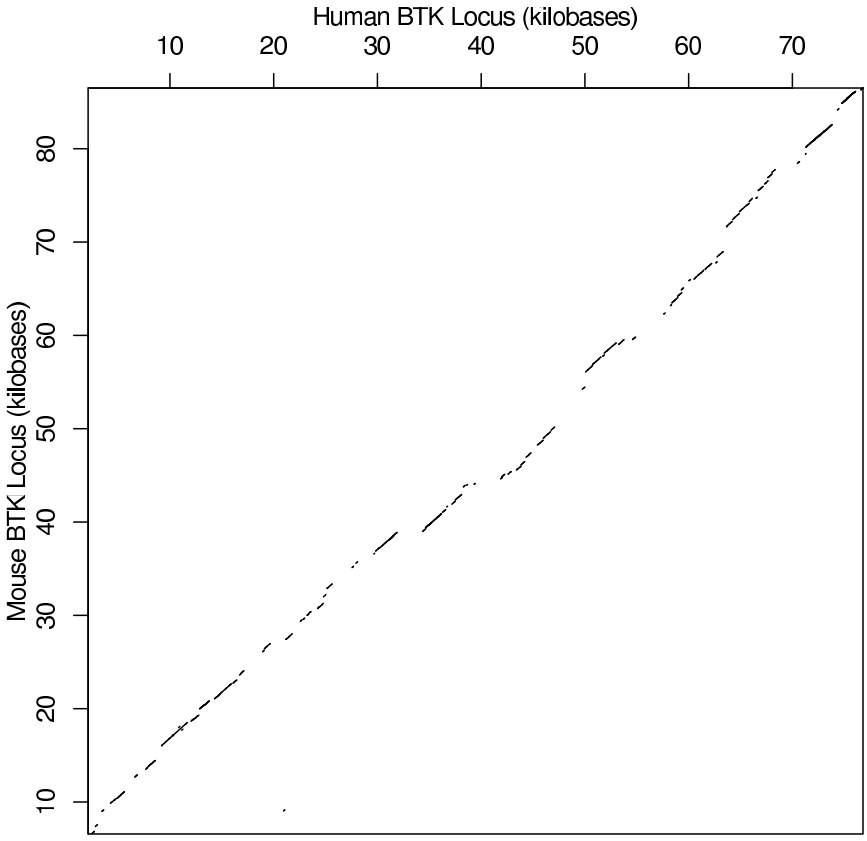

Figure 1.3: locations of significant local alignments between the human and mouse Bruton's tyrosine kinase (BTK) loci, shown in the form of a dot plot. Gapped alignments for this plot were produced with the LSH-ALL-PAIRS algorithm of Chapter 2 with a human/mouse-specific scoring function.

meaningful sequence features. Using similarity between human and mouse, we identified 100% of the exons of all genes at the locus, as well as many intronic and promoter sequences that are known or believed to function in gene regulation. Outside the genes' promoter regions, one stretch of intergenic sequence – a region at around 5 kb in the human sequence and 10 kb in the mouse – shows substantial (and to the best of our knowledge unexplained) conservation; otherwise, little conservation is observed in the intergenic portions of the sequence, which is consistent with our belief that these parts of the sequence share no common function in human and mouse.

1.3.3 Strengths of Comparative Annotation

The comparative approach to annotation performed well on the BTK locus, but have we made the annotation problem more difficult than needed? Certainly, this approach uses very limited prior information about the features of interest, particularly genes. However, comparative annotation's limited information requirements are its great strength, allowing this approach to find a wide variety of different genome features.

In the BTK example, we assumed only that the target features had a certain minimum length, and that they were well-enough conserved to exhibit the kind of ungapped alignments used to create the dot plot. Because we anticipated finding several genes at this locus, we could have increased our certainty of finding them by using a more informed comparison tool, such as `blastx` [5], that incorporates both a model of protein coding sequence and a description, such as PAM [29] or BLOSUM [45], of how it evolves over time. Indeed, we could have abandoned the comparative approach altogether by using a model-based gene finder such as GenScan [23] or GeneFinder [103]. These tools identify sequences that match a detailed probabilistic model of a gene, including both codon biases and signals such as splice junctions and the $TATA$ box.

We chose a less informed search technique to analyze the BTK locus because we sought not just genes but a *variety* of sequence features. Many of the features exposed by our search were noncoding regulatory elements, for which we have no general model that could form the basis of a more specific search tool, and one area of intergenic conservation that we

found as yet has *no* explanation. The sensitivity of comparative annotation to some known feature types, especially genes, cannot perhaps match that of more informed methods, but neither is the comparative approach specific to any one class of well-characterized features.

The most appropriate uses of comparative annotation take advantage of its generality and largely model-free nature. Besides the aforementioned task of finding regulatory elements, some other applications that exploit these strengths include:

- *finding repetitive elements:* Well-studied repeats like the human *Alu* and LINE1 families can be found by their similarity to a library of consensus sequences, while previously unknown repeat families can be identified directly by their similarity to each other. In principle, one could detect some types of repeats on the basis of known structure, such as the short inverted repeat structure of DNA transposons or the long terminal repeats of endogenous retroviruses. However, comparative annotation is the preferred method for finding these features because the known structural signals in inactive elements are rapidly obscured by random mutation, making them difficult to recover.

  Comparative annotation is particularly useful for "one-off" repeat finding, where the cost of building a model is unlikely to be amortized across additional searches. For example, the aforementioned LCR-22 chromosome-specific repeats are easily detectable but are few in number and are unlikely to occur outside human chromosome 22. The worm *C. elegans* has thousands of repeat families with similarly few members in each.

- *finding features for which a suitable model-based tool is not readily available:* Some types of feature can be found by model-based tools in practice, but the investigator simply may not have them in hand. In such cases, comparative annotation can often substitute for the specialized tool. We encountered such a situation while comparing the sequence of human chromosome 22 to itself [22]. Our analysis identified a number of approximate tandem duplications hundreds of bases in length. Although specialized tools exist for finding tandem duplications [83], we did not have them and, more importantly, had not planned to look specifically for this feature before we found it.

A comparative genomic approach may also be useful for finding pseudogene copies of known genes, which may be ignored by ordinary gene finders because they contain frameshift errors or internal stop codons.

Even active genes can sometimes be found more accurately with a little help from comparative annotation. Gene finders are prone to calling the extents of exons incorrectly, especially when their splice signals are of an unusual type, such as GC rather than GT for a splice donor site. One way to correct these mispredictions for coding exons is to observe that the exon sequence is generally better conserved (especially when translated to protein) than the intervening introns. Theoretical innovations such as paired hidden Markov models [73], or simply weighting possible choices of intron-exon boundaries by how well the exon sequences are conserved [12], can incorporate this extra comparative information into a standard gene finder.

- *discovering new and unexpected reasons for sequence conservation:* Only a small fraction of the noncoding similarities in BTK have actually been investigated experimentally and shown to be regulatory elements. The remaining similarities, especially in the intergenic regions, might conceivably have some other function entirely. Unlike comparative annotation, model-specific tools perform poorly at finding novel feature types that do not fit their models.

1.3.4 Limitations of Comparative Annotation

Comparative annotation is powerful because it relies on conservation alone to provide evidence that a sequence has biological significance. One price paid for the comparative approach's generality is that it cannot interpret the evidence it finds and therefore cannot distinguish interesting from uninteresting features. This lack of interpretation can be ameliorated in various ways, such as masking known and uninteresting features from the annotator's input (as we removed SINES in the BTK example) or identifying and removing known sequence features such as coding sequences from its output.

Comparative annotation is also limited by properties of real biosequences that are not accounted for in its simple model of conserved features embedded in uniformly uninteresting

background sequence. These limitations appear as a propensity to commit false positive and false negative errors, that is, to detect similarity where no feature exists or, conversely, to miss a feature whose sequence is insufficiently conserved. We now consider the causes of these errors and whether and how they may be remedied.

False Positives due to High Background Similarity

Similarity in the absence of conserved features can arise if the sequences being compared have not had sufficient time to diverge. Search algorithms must select a minimum degree of similarity worth reporting; for example, our analysis of the BTK locus reported only ungapped alignments above a certain length with at least 67% base identity. In choosing this similarity threshold, any search algorithm runs the risk that, even in the absence of a feature, the background portions of two closely related sequences may be similar enough to pass the threshold.

High background similarity is a problem when search tools do not accurately model a sequence's rate of neutral mutation. Search algorithms choose a similarity threshold determined by statistical considerations; in the BTK example, we chose the minimum length and percent identity to report such that at most one similarity with these properties is expected to occur by chance alone in the entire input. Karlin-Altschul theory [54] provides a basis for computing thresholds when more general score-based similarity measures are used. Both ways of setting thresholds can in principle be modified to accommodate background sequence at a known level of divergence: the probability that two bases match, or that they achieve a particular score, can be estimated under the assumption that the sequences being compared are *homologous* (that is, derived from a common ancestor) but have undergone a given amount of neutral mutation. More commonly, however, estimates of significance assume that the backgrounds being compared are *infinitely diverged*, so that the bases at corresponding positions in two homologous background sequences are effectively independent of each other. If this assumption fails, some apparently meaningful similarities may be chance occurrences, not evidence of sequence features.

Ideally, search algorithms should always choose similarity thresholds based on accurately

estimated neutral mutation rates. In practice, however, the historical rates of neutral mutation between two organisms may be difficult to estimate. Published mutation rates are often estimated from functionally important sequences, such as ribosomal RNA genes, that evolve at rates much different from nonfunctional background sequence. Neutral mutation rates can be estimated empirically from using sequence alignments [107], but a reliable estimate requires finding a representative set of homologous sequences that are believed not to be under selective pressure.

If accurate mutation rates, and hence accurate significance thresholds, cannot practically be obtained for homologous background sequences, it seems safest to apply comparative annotation to sequences from organisms for which the assumption of infinite background divergence approximately holds, or to choose a conservative threshold well above the background rate of similarity. The lack of widespread intergenic similarities in the BTK example suggests that 67% ungapped identity is a fairly safe threshold for human-mouse similarities, but more compelling evidence of noncoding conservation could be provided by comparing, e.g., human and fugu (both vertebrates, but 400 million years diverged), or even human and sea urchin (from distinct but closely related phyla). In contrast, the suspicion of false positives would taint any computational comparison, however suggestive, between the genomes of human and chimpanzee, which at roughly five million years' divergence remain over 98% identical.

False Positives due to Nonuniform Divergence

The previous discussion assumed that one could measure *the* rate of mutation for nonconserved genomic sequence in an organism. However, the rate of mutation could potentially be nonuniform even in background sequence, resulting in some background regions that diverge more slowly than others. Such slowly diverging regions, though not under selective pressure, could nonetheless be mistaken for conserved sequence features.

Nonuniform mutation in the absence of selective pressure has not been conclusively demonstrated over long time scales, and its possible causes remain poorly understood[8].

[8]In contrast, variation in, e.g., SNP frequency over short time scales is well known and can largely be explained by differing rates of recombination across the genome [14].

The primary causes of nonuniformity would likely be differing efficiency of mutation repair and physical exposure of the DNA to mutation, but just how these forces act on any given sequence is unclear. For example, the MIR repetitive element, present in numerous mammalian genomes, has been inactive since before the mammalian radiation, yet different parts of the repeat have inexplicably diverged at different rates [93]. Proximity to a functional feature might also affect the rate of divergence through some as yet unknown mechanism; in particular, a source of ongoing controversy [55, 72] in human-mouse comparisons like the BTK example is whether the majority of observed intronic similarities between these organisms reflect conserved regulatory sites or merely some form of protection against mutations in otherwise nonfunctional introns.

Although comparative annotation cannot easily separate sequence features from background regions of locally slow divergence, we take some consolation in the fact that the latter may still be considered interesting, if only for investigating the causes of nonuniform background mutation rates.

False Negatives due to Conservation of Function Without Conserved Sequence

Comparative annotation assumes that functional sequence features are conserved because their precise genomic sequence is important to their function. However, there exist features whose function is only weakly coupled to their genomic sequence. Such features are difficult or impossible to find by looking for conservation at the genomic level.

Protein coding genes are a good example of features whose genomic sequence has the potential to be only weakly conserved. Most amino acids can be encoded by any one of several synonymous codons, so the coding sequence of a gene can change substantially – up to 67% – without changing the encoded protein. Third base positions of codons are under particularly weak selective pressure – almost like background sequence – while synonymous changes in the second base position are common between organisms with different codon usage biases. The possibility of synonymous change suggests that similarity searches at the protein level, using tools like `blastx`, will likely be more sensitive than genomic sequence comparison for finding coding sequences.

A protein can remain functional even if its amino acid sequence changes, so long as its 3D structure is preserved. Over long periods, such as the divergence times between families of bacteria, the most effective way to identify a possible coding sequence may be to check not whether its sequence is recognizable but whether its product folds into a recognizable structure [18]. In some features, particularly RNA genes, neither structural *nor* sequence-based signals may be sufficiently conserved to provide strong evidence of functional conservation [80].

Although comparative annotation of genomic DNA is often effective for finding coding sequences in practice, as in the BTK example, the strong potential for false negatives when finding such features suggests that this approach is better suited to other applications. In particular, the comparative approach seems most appropriate for finding either regulatory regions such as transcription factor binding sites, whose function is closely tied to their genomic sequence, or recently duplicated features whose sequences have not had sufficient time to diverge.

1.4 The Sensitivity-Complexity Tradeoff

We have described several limitations of comparative annotation, all of which arise from fundamental problems with genomic sequence data: the background rate of neutral mutation is generally unknown; some background regions might mutate more slowly than others; and not all interesting features are well conserved at the genomic level. We accept or work around these limitations because the comparative approach excels as an easy technique for finding unknown or poorly modeled types of feature. However, there remains one major limitation to address: the high computational cost of comparative annotation.

The similarity search algorithms at the heart of comparative annotation, such as the Smith-Waterman algorithm, are designed for high sensitivity. Unfortunately, these methods also have high asymptotic complexity (with nontrivial constant factors) in the size of their input sequences. Smith-Waterman, for example, requires time proportional to the product of its input sequence lengths; search tools that use this algorithm directly [50] are practically limited to comparing only a few *megabases* (millions of bases) of sequence. Even more costly

are multiple alignment algorithms designed to find similarities in three or more sequences at once. Finding an optimal multiple alignment under a variety of scoring criteria, even in a gap-free mutation model, is known to be NP-hard [3], so the known exact algorithms for this problem scale exponentially in the size of their input.

The cost of highly sensitive alignment algorithms like Smith-Waterman seems unlikely to be significantly improved. For example, the largest speedup of the basic Smith-Waterman algorithm in the last thirty years improves its asymptotic running time by only a logarithmic factor [8]. Moreover, this algorithm runs in essentially the same time regardless of whether the input contains many or just a few significant similarities. Given today's immense growth in the amount of genomic sequence to be annotated and the observed sparsity of functional features in the genomes of higher eukaryotes, it seems desirable to develop similarity search algorithms whose cost is more sensitive to the (small) number of meaningful similarities in a sequence than to its (large) total size. We now turn to a class of methods with just this property: the *filtering heuristics*. These techniques greatly reduce search costs on average for real genomic sequences, at the price of limiting the search's sensitivity.

1.4.1 Reducing Complexity with Filtering Heuristics

The goal of filtering heuristics is to reduce the effective input size of expensive similarity search algorithms. The heuristics rapidly filter out all but those portions of the input sequences that are likely to contain an interesting similarity, producing a search problem only a fraction of the size of the original. Because the cost of algorithms like Smith-Waterman scales superlinearly with input size, reducing the effective problem size through filtration can dramatically speed up similarity search.

A simple but widely used filtering heuristic exploits the presence of *word matches*, or runs of matching bases, in an alignment. The general form of the heuristic is: *when searching for similarities between two sequences, apply the full Smith-Waterman algorithm only to portions of the sequences that contain a sufficiently long word match.* We ignore for now the question of how to apply Smith-Waterman only to parts of the sequences and focus instead on the behavior of the word match heuristic, which is parameterized by a minimum

threshold match length.

The word match filtering heuristic is appealing for two reasons. First, word matches, unlike more general similarities, are easy to find efficiently. Specifically, all the word matches above some threshold length can be enumerated in time proportional to their number plus the total length of the sequences being compared[9]. If the number of matches is small, this bound compares quite favorably with dynamic programming approaches that require time quadratic in the sequence length. Algorithmic techniques that achieve the linear cost bound for finding word matches include hashing, as used by e.g. NCBI's `blastn`, and suffix trees [42, Chapter 5], used in e.g. the MUMmer software of Delcher et al. [30].

Second, we can show that, given an appropriate choice of match length, the word match heuristic finds *all* alignments above a certain similarity threshold. The following lemma is drawn from [77] but was known long before it:

Lemma 1.1 (Pevzner and Waterman, 1995) Suppose two sequences have a pairwise local alignment of length at least ℓ (i.e., containing at least ℓ columns) with mutations (substitutions or indels) in at most d columns. Then the alignment contains a word match of length at least

$$\left\lfloor \frac{\ell}{d+1} \right\rfloor .$$

Proof: The alignment contains at least $\ell - d$ matching positions, divided into at most $d+1$ segments (some possibly of zero length) by intervening mutations. The average segment length is therefore at least

$$\frac{\ell - d}{d+1}.$$

Hence, there exists a segment with no mutations of length at least

$$\left\lceil \frac{\ell - d}{d+1} \right\rceil = \left\lfloor \frac{\ell}{d+1} \right\rfloor$$

which is the claimed word match. ∎

[9]The key property of word matches that makes them efficient is that equality of strings is an *equivalence relation*. Finding all word matches of a given length k is therefore the same as computing equivalence classes on the set of all length-k substrings of the input.

Corollary 1.2 Finding every word match of length at least $\lfloor \ell/(d+1) \rfloor$ will identify the locations of all alignments of length at least ℓ and percent identity at least $100(\ell - d)/\ell$.

The word match filtering heuristic uses an inexpensive preprocessing step to limit the application of the quadratic Smith-Waterman algorithm to regions of the input sequences that are provably "promising." In practical similarity search problems, interesting features rarely make up more than a small fraction of the total sequence length, so the total length of all truly significant sequences should be quite small compared to the the original input size. The cost of the algorithm is therefore largely determined by how *specific* the heuristic is, i.e. how much of the promising sequence is actually part of a significant similarity. We estimate this cost by asking, given a sequence model that looks approximately like real genomic DNA, how likely is a word match in the absence of a significant alignment?

For a rough estimate of the rate of word matches occurring by chance alone, assume that the sequences being compared are i.i.d. random with equal base frequencies. Then, for two sequences of length N, the expected number m of word matches between them of length k is given by

$$E[m] = \left(\frac{1}{4}\right)^k (N - k + 1)^2. \tag{1.1}$$

If, as in some similarity search tools, we do not fix a length but rather count non-overlapping *maximal* word matches, the expected number of matches remains about the same. A word match is maximal if it cannot be extended in either direction, that is, if (ignoring end-of-sequence effects) it is bounded by a mismatch on each side. The expected number of maximal matches of length at least k is therefore at least

$$\sum_{i=k}^{N-2} \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^2 (N - i - 1)^2$$

which for large N is about $3/4$ of the total number of k-mers. This estimate extends straightforwardly to i.i.d. sequence with more realistic unequal base frequencies: simply replace $1/4$ and $3/4$ in the above formulas by ϕ and $1 - \phi$, where ϕ is the probability that two randomly chosen bases match each other.

Given the provable bound on sensitivity and the estimated specificity of the word match filtering heuristic, we can plot the tradeoff between them. Figure 1.4 shows how specificity,

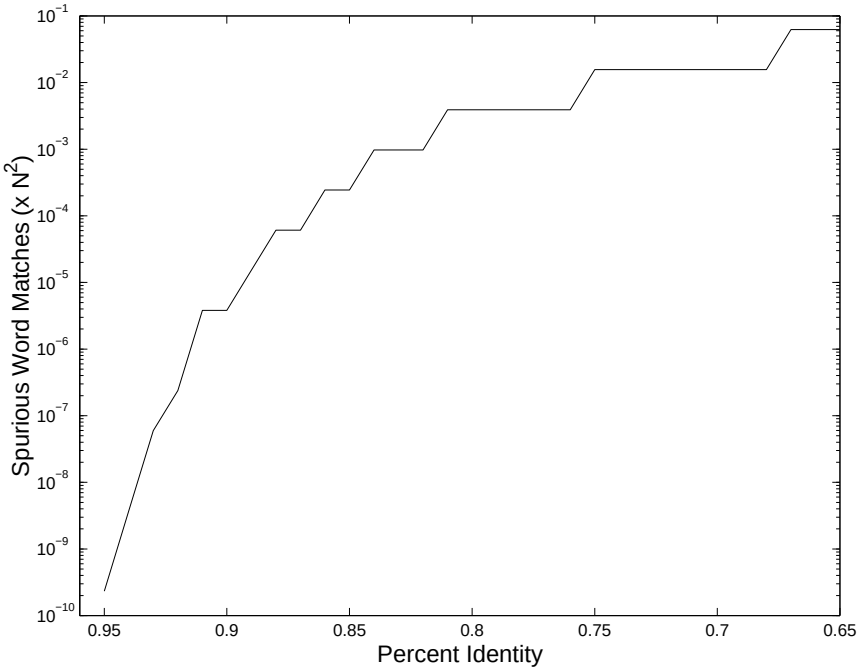

Figure 1.4: sensitivity and specificity of exact word matches, computed using the worst-case sensitivity estimate of Lemma 1.1 and the estimated specificity of Equation (1.1). The number of chance matches is given as a fraction of the sequence length N^2. Values were computed assuming an alignment of length $\ell = 100$.

as measured by the number of chance matches, degrades as projected sensitivity increases. At high levels of similarity, say 90%, the tradeoff permits relatively long match lengths and proves advantageous. As the threshold level of similarity decreases, or equivalently as sensitivity increases, specificity becomes so poor that the search spends most of its time chasing false leads and becomes computationally impractical. Thus, **the specificity of the search heuristic limits the sensitivity that can be achieved at reasonable cost.** Just where the limit falls depends on the cost of checking and discarding false positive matches, but it is clear that, for example, this heuristic is inappropriate for finding alignments with 67% identity – doing so would require checking all word matches of size two, which requires essentially N^2 pairwise comparisons.

The above heuristic is only one of numerous tricks used to trade off sensitivity for efficiency in similarity search, and indeed only one of several different ways to exploit word

matches for this purpose. Practical search tools like BLAST use word match lengths larger than the bound specified by Lemma 1.1, which are not guaranteed to find every desired alignment but do find most of the interesting ones in practice with a much lower rate of false positive word matches. The search heuristic can also be improved by choosing more specific "fingerprints" of similarity than word matches; advanced fingerprinting techniques include the Chang-Lawler algorithm [25] and Pevzner and Waterman's multiple filtration [77]. We will have more to say about these alternate approaches in the next chapter, when we introduce our own randomized fingerprinting scheme.

1.4.2 Significance of the Thesis Revisited

Armed with some perspective on genome annotation, we now revisit the significance of this work. Comparative annotation is a powerful approach to finding features in genomic DNA, but the most sensitive algorithms for similarity search are too expensive to work on long sequences or large multiple alignments. Previous computational work includes a number of ways to trade off some of the sensitivity of similarity search for increased specificity, and hence increased efficiency, making large-scale comparative annotation practical.

In this work, we explore a new space of computational tradeoffs for similarity search, using the randomized technique of *random projection*. Like the above heuristics, random projection sacrifices some sensitivity for the sake of efficiency. However, the new tradeoff has some interesting properties: we can formally analyze and control how sensitivity degrades, in a manner similar to the inefficient heuristic of Lemma 1.1 and more complex fingerprinting schemes, while achieving a practical sensitivity/efficiency tradeoff closer to that of BLAST-like heuristics even when pursuing relatively low levels of similarity. Also, because random projection eschews the use of word matches, we remove a bias present in nearly all tools in wide use today[10], enabling us to find features that these tools miss. These new properties of our method extend, in somewhat altered form, to multiple alignment as well.

As we apply random projection to annotation, we will avoid the well-studied problem of comparing an essentially constant-sized query sequence to a large corpus, in the style of

[10] while, of course, introducing our own distinct bias

`blastn`. The cost of such comparisons scales linearly with corpus size, and the constant can be made extremely small by clever $O(1)$ query preprocessing (see, e.g., [9]) or hardware assistance [88, 74] with minimal loss of sensitivity. Instead, we concentrate on two important annotation problems whose quadratic or worse asymptotic cost makes an efficient yet sensitive search more challenging: finding *all* significant pairwise local alignments between very long sequences, and finding regulatory motifs by *multiple* local alignment. Besides being computationally more difficult, these problems are particularly germane at a time when biology is pursuing the comprehension of entire genomes, and noncoding regions in particular, through comparative annotation using sequence from an increasing number of organisms.

Before proceeding, it is worth considering whether existing, highly efficient annotation methods are already sensitive enough. In particular, one might object that comparative annotation today is *already* identifying more potential genome features than can reasonably be verified and studied in the lab. Our answer to this objection is two-fold. First, the ongoing effort to explain the 97% of the human genome that is not part of some gene suggests that increased sensitivity remains a biologically interesting goal. In particular, each new, more informed release of the RepeatMasker software seems to explain a slightly greater fraction of the genome as ancient interspersed repeats. Generalizing from this example, the fact that a sequence can be explained as an instance of a known feature means that, even if we decline to pursue *any* similarities between two unrecognized sequences, it is always worth producing algorithms that are more adept at matching long unknown sequences to a large database of known features.

Second, significant similarity is useful as evidence that a particular genomic sequence is worthy of further study. In annotating an entire genome, we are compiling reference materials for biologists who will ultimately decide which regions to study in depth. We cannot predict which criteria these investigators will use to decide between interesting regions, but we can state that, given the huge and growing volume of genomic sequence being studied, a region that contains no annotation seems unlikely to receive much further attention. Large-scale genomic sequence analysis therefore demands that annotation algorithms become *more* sensitive, trying their hardest to find whatever biologically meaningful features

can be found. To do otherwise potentially robs the investigator of the information necessary to guide future laboratory work.

Chapter 2

THE ALL-PAIRS LOCAL ALIGNMENT PROBLEM

We first apply random projection to the problem of finding interesting pairwise local alignments between substrings of two or more long DNA sequences. This problem abstracts a number of important annotation tasks based on pairwise sequence comparison. In this chapter, we develop a randomized pairwise alignment algorithm, LSH-ALL-PAIRS, that is both efficient and sensitive to biologically meaningful similarities that might otherwise be missed by standard annotation tools.

2.1 Problem Definition

The general *all-pairs local alignment* problem is formally stated as follows:

Problem 2.1 Let $C = \{s_1, s_2, ...\}$ be a collection of DNA sequences of total length N bases; let $\mathcal{F}$ be an alignment scoring function; and let θ be a fixed score threshold. Find every pair of substrings (s_1, s_2) of any sequences in C such that an optimal alignment $\mathcal{A}$ of s_1 with s_2 has score $\mathcal{F}(\mathcal{A}) \geq \theta$.

This formulation makes precise our intent from the previous chapter. The problem as stated permits arbitrary overlap and even containment between reported similarities; however, practical search tools avoid reporting trivial variations or subalignments of the same similarity by limiting the degree of permitted overlap. We do not include such restrictions in the problem definition because they vary between implementations and because they are usually enforced only after the core filtering algorithm.

All-pairs local alignment abstracts a variety of practical annotation problems. Below, we describe some specific variants of Problem 2.1 and illustrate the range of typical input sizes N. In all of these variants, the score threshold θ is determined by considerations of statistical significance, such as Karlin-Altschul theory [54].

- *comparing genome fragments across organisms:* The BTK example from the previous chapter illustrates a common restriction of the general all-pairs problem. We are given a collection of two or more sequences and wish to find all high-scoring local alignments between substrings of *distinct* sequences (in this case sequences from distinct organisms) in the collection.

  Large-scale sequence comparisons between organisms are useful for finding features that are *orthologous* (derived from the same sequence in the organisms' common ancestor), and to elucidate the history of genome rearrangements since two organisms diverged. In the case of BTK, the genes with greatest pairwise similarity occur in the same order in human and mouse, so the sequence at this locus appears to have undergone no major rearrangement since human and mouse diverged. Conserved gene order is interesting because it provides evidence that similar pairs of genes at a locus are indeed evolutionary orthologs, while disruptions of gene order are useful as evidence that an organism's genome has been rearranged over time. Delcher et al.'s MUMmer [30] is one tool that uses all-pairs local alignment as a way to visualize genome rearrangements.

  Typical problem sizes for cross-organism comparisons range from $N \approx 10^5$ to 10^6 bases for comparisons of long homologous regions (though contiguous homologous segments of more than ten megabases are known between mouse and human) to around 10^7 bases to visualize rearrangements between related bacterial or archaeal genomes. Future comparisons between substantial parts of two eukaryotic genomes may run to 10^8 bases or more.

- *finding novel repetitive elements:* Similar sequences that result from duplications within one organism are said to be *paralogous*. In particular, interspersed repeat families are paralogous groups of features arising from duplications of an ancestral transposon. Finding and cataloging novel families of repeats is important both to study their biology and to identify and remove them from comparisons where they are of no interest.

Unlike cross-organism comparison, repeat finding imposes no limit on which pairs of substrings from the input may yield interesting similarities, so the most general formulation of all-pairs local alignment applies. The frequency of repetitive elements determines the amount of sequence in which they must be found. 100 kb of human sequence would probably suffice to obtain a good sample of *Alu* elements, but for rarer repeat families such as the endogenous retroviruses, more typical problem sizes are $N \approx 10^6$ to 10^8 bases. Finding repeats across an entire genome is reasonable if the repeats are not too numerous, or to visualize large-scale duplications as are found in *Arabidopsis* [7]. For such problems, N might be 10^8 to 10^9 bases.

- *annotating a long sequence using a database:* This restriction of all-pairs local alignment is similar to the classic BLAST-style search: find all significant local alignments between a single query sequence and any of the sequences in a database. The problem becomes computationally more challenging if we allow a very long query sequence, perhaps as long as an entire genome.

  Two common examples of annotating long sequences from a database are repeat identification, as performed by RepeatMasker, and *EST matching*. In the latter problem, the goal is to find genes by their similarity to *expressed sequence tags (ESTs)* – known exonic DNA sequences derived by sequencing expressed mRNAs [1]. Repeat masking and EST matching use databases of very different sizes. RepeatMasker's library of consensus sequences for known human repeats is around 10^6 bases[1]; in contrast, the GbEST database[2] is closer to 10^{10} bases.

- *assembling overlapping sequences:* In this application, the input is a collection of short sequences, often individual reads of 500-700 bases from a DNA sequencing machine, all of which occur as substrings of one long sequence. The goal is to discover all overlaps between pairs of substrings. Given this set of overlaps, an *assembly algorithm* [40, 49, 101] attempts to reconstruct the original long sequence from its fragments.

---

[1] as of version 4-4-2000

[2] as of GenBank Release 123

Typical assembly problems include reassembly after shotgun sequencing, in which a long sequence is deliberately shredded to accommodate the size limitations of laboratory sequencing procedures, and EST assembly, which tries to reconstruct genes' exonic sequences from fragments obtained while building an EST library. While the sizes of these problems can be quite daunting – 10^9 to 10^{10} bases for a human-sized whole-genome shotgun assembly – finding overlaps is quite easy because the overlapping parts of the sequences are identical copies except for a small number of polymorphisms and experimentally introduced sequencing errors.

2.2 A Brief Survey of Filtering Strategies

All-pairs local alignment must be able to find similarities efficiently in millions, even billions, of bases of sequence. Unfortunately, searching such large sequences using the Smith-Waterman algorithm is cost-prohibitive: for inputs of size N, the cost is $\Theta(N^2)$ (with a nontrivial constant) regardless of how many significant similarities are actually present. This high cost motivates the use of filtering strategies to reduce the effective problem size to manageable levels when the actual number of significant similarities is small. We mentioned several such techniques in Section 1.4.1; here, we review them in more detail. All the methods described share one key property: their performance scales well with the problem size N for high similarity thresholds θ but rapidly worsens as the threshold decreases.

2.2.1 Measuring a Filter's Performance

Similarity search with filtering consists of three major tasks: **filtering** the initial search space by identifying *candidate pairs* of short (usually constant-length) substrings, each of which may indicate the presence of a significant similarity; **checking** each candidate substring pair to determine whether a similarity does indeed occur there; and finally, **enumerating** the similarities found. If the filter is effective *and* similarities occur only rarely, the number of candidate pairs will be orders of magnitude less than N^2, eliminating nearly all of the cost of applying an expensive checking algorithm like Smith-Waterman to the full-length input sequences.

It is critical to note that filtering does *not* improve a search algorithm's asymptotic worst-case complexity. For all the filters described in this work, an adversarial input (e.g., a string composed entirely of A's) can always force the number of candidate pairs, and therefore the running time, to $\Theta(N^2)$. Filtering algorithms seek only to reduce the quadratic cost of similarity search by a (large) constant factor, and only for a model of input sequence that is similar to the genomic sequences observed in practice. Filtering may be considered practically successful so long as it reduces the computational cost of all-pairs local alignment enough to make feasible problems of size 10^6 to 10^{10} bases – the size range of the applications described above.

The string matching literature measures the performance of filtering strategies under a simplified model that makes formal analysis tractable. A filter's primary performance metric is the expected number of *spurious* candidate pairs it produces, that is, the number of candidates that occur by chance in the absence of a significant similarity. The emphasis on spurious candidates is consistent with a belief that true similarities are rare, or at least that no algorithmic tricks can reduce running time when they are not rare. Performance analysis often neglects the cost of producing candidates in the first place. Ignoring this cost is asymptotically justified because it is usually $O(N\text{polylog}N)$, while the number of candidates is $\Theta(N^2)$. Even so, the method used to generate candidates can still have a large impact on the algorithm's constant factor at smaller problem sizes.

The expected number of spurious candidate pairs is estimated assuming a background sequence composed of i.i.d. random bases. Real DNA sequence has inter-base correlations that are more accurately captured by a Markov model, but the i.i.d. assumption makes analysis much more tractable and still gives order-of-magnitude correct cost estimates for many filtering strategies. The base composition of the background sequence is sometimes fixed at equal frequencies for the four bases. While a uniform distribution simplifies analysis, base frequencies in real genomic DNA may be highly nonuniform, varying both between organisms and within a single genome. If possible, performance analysis should account for nonuniform base composition.

2.2.2 Word Match Filtering

The *exact* or *word match* filtering heuristic, introduced as an example in Chapter 1, declares two substrings to be a candidate pair if they have length above some threshold k and match exactly. Such matches are sometimes referred to as *k-mer* matches, a k-mer being a string of k bases[3]. Word match filtering is the most popular filtering technique in use today.

Lemma 1.1 gives a lower bound on the word length k that is guaranteed to produce a candidate wherever a region of sufficiently high similarity occurs. Unfortunately, this bound is tight, and the word lengths it implies for even moderately low levels of similarity are considered too expensive for practical use. For example, to obtain all alignments with at least 80% identity, $k = 4$; the expected number of spurious candidates is roughly $4^{-k}N^2$, so about 0.4% of all tetramer pairs in the input are candidates. For $N = 10^6$ bases – a relatively small problem – this estimate implies *four billion* spurious candidates. 4×10^9 is less than the full quadratic cost $N^2 = 10^{12}$, but checking each candidate pair is substantially more expensive than filling in a single cell of a Smith-Waterman matrix. Such a large number of false candidates therefore implies a runtime cost not much better than that of the full dynamic programming algorithm.

Word match filtering can be made much more efficient if we are willing to sacrifice the formal guarantee of finding *every* desired similarity. Biosequence similarities rarely attain the worst case predicted by the lemma; indeed, the worst case occurs only if mutations are equally spaced along the length of the alignment [4]. The common case is more favorable because only the *longest* word match in a similarity need exceed the threshold k. Figure 2.1 illustrates the expected length of this longest word for various similarity lengths at 80% base identity, assuming that the 20% of mutated bases in each similarity are chosen uniformly at random without replacement. A 100-column alignment with 80% identity has an average longest word match of about fifteen bases, implying less than 1000 spurious similarities when $N = 10^6$. Moreover, long conserved features frequently have short regions that are better conserved than average and may therefore contain a longer-than-expected word match.

[3]Standard terminology for short k-mers uses Greek prefixes, i.e. *monomer, dimer, trimer, tetramer*, etc.

[4]Even so, at least one published filtration algorithm, that of Baeza-Yates and Perleberg, uses worst-case word lengths [10].

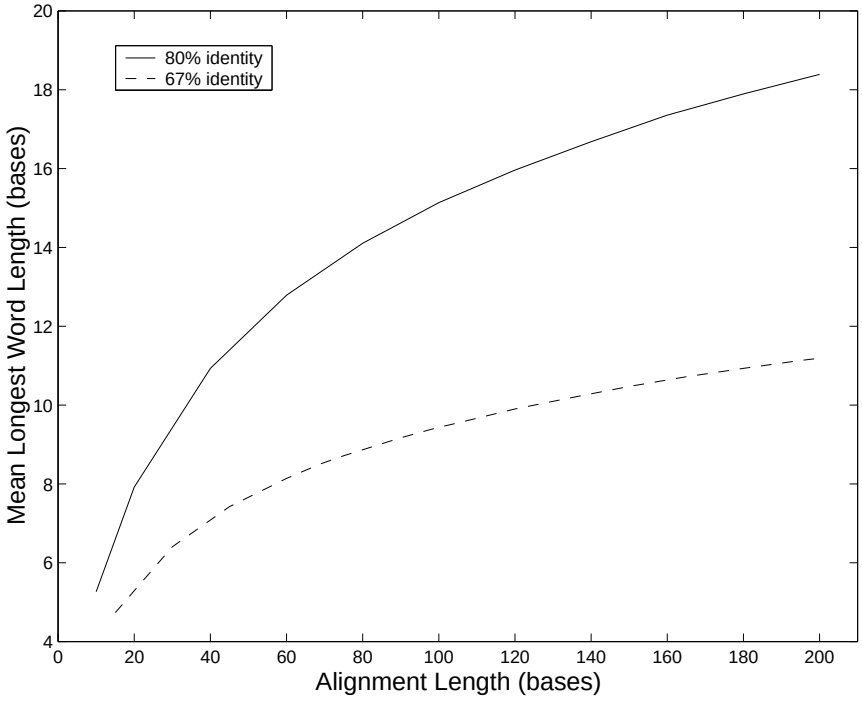

Figure 2.1: average length of longest word match in similarities with uniformly distributed mutations. Averages were estimated empirically from 10^5 randomly generated similarities.

Perhaps the most widely used similarity search tool based on word match filtering is NCBI BLAST [5], whose `blastn` program uses 11-mer candidate pairs. Other local alignment search tools using variations on word matching include Gish's WU-BLAST [4] and Pearson and Lipman's FASTA [75] – both, like NCBI BLAST, designed to compare a short query against a long corpus – as well as all-pairs tools such as Green's CrossMatch [40] (the search engine behind RepeatMasker), Batzoglou et al.'s GLASS [12], and the `blastz` algorithm of Schwartz et al.'s PipMaker [85]. In all of these tools, candidate pairs are found (without explicit enumeration) by a linear- or $N \log N$-time method such as hashing or constructing a sorted suffix array, while checking is implemented by some variant of Smith-Waterman.

Word match filtering with much greater than worst-case word lengths is both sensitive and efficient at levels of similarity above about 80%. It is the method of choice for detecting overlaps in assembly problems, where overlapping fragments, even with sequencing errors, commonly have identities above 95%. However, the tradeoff of word length against sensitivity rapidly becomes less favorable as the number of mutations approaches one-third of the alignment length. As Figure 2.1 shows, the longest word match in a 100-base alignment with 67% identity averages less than ten bases, compared to fifteen bases at 80% identity. A highly sensitive search, like that used for ancient LINE elements in RepeatMasker, might require a word length of eight bases or less. Moving from fourteen- to eight-base words increases the rate of spurious candidates more than 4000×.

2.2.3 Exclusion Methods

Although word match filtering is the most popular technique for accelerating local alignment, its sensitivity to similarities is closely tied to how their mutations are distributed. Lemma 1.1 provides the method's only formal guarantee of sensitivity, and that only for impractically short word lengths. Several more elaborate filtering strategies, collectively known as *exclusion methods*, provably find all similarities above a given threshold while maintaining a lower rate of spurious candidates than that implied by the lemma.

Exclusion methods in the literature are formulated to solve the following problem:

Problem 2.2 Let C be a corpus of sequences, and let P be a *pattern* sequence. Find all substrings of C that match P to within at most d mutations (alternatively, d substitutions).

As in BLAST, the pattern P is assumed to be a short sequence, while the corpus can be of arbitrary length. Every reported match must utilize the entire pattern sequence. In contrast, the all-pairs problem, which we reformulate below to use the exclusion algorithms' restricted notion of similarity, specifies *no* fixed pattern:

Problem 2.3 Let C be a corpus of sequences. Find all pairs of substrings in C such that both members have length at least ℓ and match to within at most d mutations (alternatively, d substitutions).

Any solution to Problem 2.2 can naively be adapted to the all-pairs problem by taking every ℓ-mer of C in turn as the pattern string and solving $|C|$ pattern matching problems. Unfortunately, exclusion methods typically perform a linear-time filtering pass over the corpus for each pattern, so the naive all-pairs solution, which performs $|C|$ such passes, spends quadratic time in producing candidates even before checking them[5].

Some of the most efficient exclusion methods depend inherently on having a fixed pattern, making them difficult to extend to the all-pairs problem other than by the naive solution. For example, Myers' method [67] attains higher performance than most exclusion methods by first finding approximate matches to substrings of P that are logarithmic in the corpus length, then progressively extending these matches to encompass the full pattern. Each extension phase doubles the match length and discards any matches that cannot be extended without incurring more than d mutations. One might expect that a match would have to be extended both to the left and to the right to avoid missing any similarities; however, Myers exploits the fact that the position of each match in the pattern is known to extend a match *either* left or right but not both, thereby eliminating redundant extensions and

[5] We can reduce the number of passes without sacrificing correctness by the following prefiltering scheme: for some $r > 1$, divide the corpus into $r\,|C|\,/\ell$ *non-overlapping* regions of length ℓ/r; solve Problem 2.2 using the original threshold d and each region as the pattern; and finally, test only those ℓ-mers that overlap a region with at least one match. This scheme reduces the initial number of passes by a factor of ℓ/r, but whether it is practically useful depends on how many spurious ℓ/r-mer matches are produced after the initial passes.

achieving higher efficiency. If, as in the all-pairs problem, the boundaries of the pattern are unknown, this strategy fails. Automaton-based exclusion methods like that of Baeza-Yates and Navarro [9] do not extend well to the all-pairs problem either, simply because the size of the automaton used to filter the corpus quickly becomes prohibitive. Other methods, such as the Chang-Lawler algorithm, do have nontrivial all-pairs extensions (see Appendix A for details), but the performance of such extensions in practice is at best uncertain because the extended algorithm may generate many more spurious candidates than the naive all-pairs extension.

One exclusion algorithm that *does* extend well to the all-pairs problem is Pevzner and Waterman's ungapped double filtration scheme [77]. This method improves on the basic word match filter of Lemma 1.1 by introducing a second kind of match, the *gapped k-tuple match*. A gapped k-tuple is simply a set of k sequence positions with a fixed spacing (called the *gapsize*) between them. For example, the sequences "$ACAGGTA$" and "$AGACGAG$" exhibit a gapped 3-tuple match with gapsize 2, namely the set of positions $\{1, 3, 5\}$. It can be shown (see Appendix B) that an *ungapped* alignment with sufficiently few substitutions contains both a long word match and a long gapped tuple match. The tuples in a sequence, like its component words, can easily be tabulated by hashing, so it is straightforward to restrict the set of candidate pairs to substrings in which a word match and a tuple match occur together. Using two simultaneous filtering criteria reduces the number of candidates and therefore improves overall search efficiency.

Unfortunately, the performance of most exclusion methods, including double filtration, rapidly decays as the user's similarity threshold decreases. The rate of decay is substantially *worse* than practical word match filtering. Consider, for example, the aforementioned problem of finding a match to a pattern of length 100 with at least 67% identity. By even a conservative estimate, Chang-Lawler is expected to pass at least 10% of all corpus regions through to checking, and the actual fraction is likely much higher. Double filtration's performance decays equally badly: the word length and tuple length bounds at 67% identity are both two bases, and the expected number of spurious candidates approaches the $N/16$ expected from word matching alone. Even worse, the initial filtering phase of double filtration must process about $N/16$ word matches just to find the candidate set. The all-pairs

equivalents of these algorithms perform little better at 67% identity than simply checking every pair of ℓ-mers in the input.

Why does exclusion algorithms' performance decay so rapidly at low levels of similarity? The fundamental problem is the same as that seen in Lemma 1.1: the filtering criteria on which the algorithms depend are strong at high levels of similarity but become unusably weak by the time identity falls to 67%. This unfortunate property seems to hold for *every* known deterministic filtering criterion; practical word match filtering gets around it only by providing no formal guarantee of sensitivity. An analogous issue is the (somewhat amorphous) "curse of dimensionality" in computational geometry: as the problem dimension increases, a point's neighborhood becomes exponentially larger, and algorithms for locating points (e.g. nearest-neighbor) must sift through exponentially more space to find them. Empirically, deterministic point-location algorithms tend to scale badly with dimension.

In sequence alignment, the problem "dimension" is the similarity threshold. Algorithms for Problem 2.2 search for sequences within a fixed Hamming distance of a query sequence. As the distance bound d increases, the d-neighborhood of a sequence in Hamming space also increases exponentially in size. Just as in geometry, deterministic similarity search algorithms empirically appear to scale badly as the dimension increases.

2.3 Filtering by Random Projection

Thus far, we have seen two kinds of filtering scheme: deterministic and heuristic. The deterministic exclusion algorithms of Section 2.2.3 are designed to have 100% guaranteed sensitivity to similarities above a user-specified threshold, but they do not scale well to large all-pairs problems at low but practically interesting identity thresholds of 65-70%. Practical word matching is much more efficient in practice, but its sensitivity depends in an uncontrolled way on the distribution of mutations in the similarities of interest, and its efficiency or sensitivity also decays substantially for similarities around 67% identity.

We now introduce a third option for similarity search: *randomized filtering*. Like practical word matching, randomized filtering makes no guarantee of finding *all* similarities above a given threshold. However, it does guarantee its sensitivity *in expectation*, regardless of

how mutations are arranged in a similarity. By relaxing the deterministic guarantee of most exclusion algorithms in a controlled way, randomized filtering reduces its rate of performance decay with decreasing similarity, outperforming methods like double filtration at low similarity rates. Moreover, by using a filtering criterion that does not depend on word matches, randomized filtering becomes sensitive to significant similarities that are difficult to detect at the word lengths used by BLAST and other existing tools.

The basic technique we will describe, like double filtration, is restricted to finding ungapped alignments. The precise problem to be solved is the following:

Problem 2.4 Let C be a corpus of sequences. Find all pairs of substrings in C of common length ℓ that match (without gaps) to within at most d substitutions.

Below, we first describe our core filtering technique, *random projection of strings*, then show how to build an algorithm, LSH-ALL-PAIRS, for all-pairs alignment using this filter. Using LSH-ALL-PAIRS efficiently requires setting certain parameters correctly, so we next analyze how best to parameterize the algorithm. Finally, we discuss additional details required to produce an efficient implementation and to heuristically extend the core, ungapped algorithm to produce gapped alignments.

2.3.1 The Filter

Consider two strings s_1 and s_2 of common length ℓ over an alphabet Σ. Fix $d < \ell$; we say that s_1 and s_2 are *d-similar* if they differ by at most d substitutions.

To detect d-similarity between s_1 and s_2, we construct the following randomized filter. Choose k indices $i_1 \ldots i_k$ uniformly at random from the set $\{1 \ldots \ell\}$; for ease of analysis, assume that the indices are sampled with replacement, so that an index can be chosen multiple times. Define the function $f : \Sigma^\ell \to \Sigma^k$ by

$$f(s) = \langle s[i_1], s[i_2], \ldots, s[i_k] \rangle .$$

The function f is called a *projection function*. It concatenates characters from at most k distinct positions of s to form a k-mer, the *projection of s onto* $\{i_1 \ldots i_k\}$. Formally, f is a projection from an ℓ-dimensional generalized Hamming space into one of its k-dimensional

subspaces. Our filter accepts the pair of ℓ-mers (s_1, s_2) as a candidate pair if and only if $f(s_1) = f(s_2)$.

Readers familiar with recent literature in computational geometry may recognize random string projection as being a *locality-sensitive hashing (LSH)* scheme in the sense defined by Indyk and Motwani [51]. The functions f correspond to Indyk's locality-sensitive hash functions, so called because the probability that two strings project to the same value under f varies directly with their degree of similarity. More precisely, if s_1 and s_2 are d-similar,

$$\Pr[f(s_1) = f(s_2)] \geq \left(1 - \frac{d}{\ell}\right)^k, \tag{2.1}$$

where the probability is computed over all random choices of $i_1 \ldots i_k$ for f. Our randomized filter is therefore likely to accept pairs of strings that differ by only a few substitutions while rejecting pairs that differ by many substitutions. Pairs that match exactly are always accepted.

Like any filtering scheme, random projection can commit *false positive errors*, producing spurious candidate pairs. A false positive occurs when strings s_1 and s_2 are *not* d-similar but $f(s_1) = f(s_2)$, i.e. if f samples only positions at which the two strings agree. The false positives produced by random projection are caught and discarded when candidate pairs are checked. However, unlike the deterministic exclusion algorithms of Section 2.2.3, which are guaranteed to find every requested similarity in the input, our filter can also commit *false negative errors* that cause d-similarities to be missed. A false negative occurs if s_1 and s_2 are d-similar but $f(s_1) \neq f(s_2)$, i.e. if f samples any position at which the strings disagree.

Although both random projection and practical word match filtering can commit false negative errors, the cause of these errors and their remedy differ substantially between methods. Word match filtering will *never* discover an ungapped d-similarity that fails to contain a sufficiently long word match. The only remedy is to decrease the word length, perhaps as low as the adversarial bound of Lemma 1.1. In contrast, whether a projection function f finds a given d-similarity depends on the set of indices sampled by f. For randomly chosen f, the sensitivity of our filter is *independent* of the arrangement of substitutions in the input's d-similarities. Moreover, we can make the chance of a false negative error arbitrarily

small, and therefore achieve sensitivity arbitrarily close to 100%, by repeating the filtering scheme with different, independent random projections. Each new filter provides another chance at finding a previously missed similarity.

2.3.2 Prior Work in Projection

Random projection's provenance includes prior work in sparse indexing of databases and high-dimensional computational geometry. Distance estimates based on sparse sampling of numerical feature vectors have long been used in machine vision to match a perceived object against a database of known objects [32, Chapter 6]. Closer to our own application, Rigoutsos and Califano experimented with random projection in their FLASH protein similarity search tool [78] and noted empirically that it outperformed a variety of other sparse indexing schemes. Projections in Hamming spaces are a special case of distance-preserving embeddings of points from general metric spaces, which have been studied by e.g. Johnson and Lindenstrauss [53], Bourgain [19], and Linial, London, and Rabinovich [61].

In computational geometry, Indyk and Motwani [51] used random projection, in the form of locality-sensitive hashing, as part of a randomized algorithm for the high-dimensional nearest neighbor problem. This problem is a geometric analog of Problem 2.2: given a query point in space (in their case Euclidean rather than Hamming), find the point in a database that is closest to the query. It is also a classic victim of the curse of dimensionality. By introducing randomization, and in particular by reducing the original problem to a series of tractable low-dimensional analogs, Indyk and Motwani produced an algorithm that scales better with dimension than deterministic methods and is efficiently implementable in practice for sparse indexing of databases [38]. Just as Indyk and Motwani used random projection to ameliorate the curse of dimensionality, we use it to improve sensitivity to significant but poorly conserved similarities.

2.3.3 The LSH-ALL-PAIRS Algorithm

Figure 2.2 gives the algorithm LSH-ALL-PAIRS, a complete solution to Problem 2.4 based on filtering by random projection. Given a collection of sequences C, the algorithm attempts

to find all ungapped d-similarities between any pair of ℓ-mers in C.

LSH-ALL-PAIRS iterates the following four steps:

1. Choose a random locality-sensitive hash function f by picking k indices $i_1, \ldots, i_k$ from $\{1, \ldots, \ell\}$.

2. For every ℓ-mer s of every sequence in C, store a tuple $\langle f(s), \pi(s) \rangle$, where $\pi(s)$ is the position of s in C.

3. Partition the tuples into classes $\{C_1, C_2, \ldots\}$, such that all tuples in a class have the same projection $f(s)$.

4. For each class C_q, compare all pairs of ℓ-mers whose tuples are in C_q. Store those pairs that actually match to within at most d substitutions.

Steps 1–4 are iterated m times, after which LSH-ALL-PAIRS outputs the union of the sets of d-similarities found in each iteration. The constants m and k are chosen based on efficiency and sensitivity considerations described in the next section.

Steps 1–3 of the LSH-ALL-PAIRS algorithm constitute its filtering phase, while step 4 is the checking phase. The candidate pairs of ℓ-mers are those with the same projection value; these pairs are generated explicitly at the beginning of step 4. LSH-ALL-PAIRS is sound because the checking phase discards any candidate pair that is not a d-similarity, but it is not complete: for $d > 0$, the algorithm will fail to find any d-similarity that happens to be a false negative for every one of the m projections chosen.

The running time for each iteration of LSH-ALL-PAIRS is dominated by two factors: the cost of creating and partitioning tuples for each ℓ-mer in C, and the cost of checking the candidate pairs in each class C_q. Creating the tuples clearly takes time $\Theta(kN)$ per iteration, where N is the size of the input C, but they can also be partitioned in time $\Theta(kN)$ by a variety of methods. For example, we can partition tuples by hashing keyed on their projection values or, because the length k and alphabet $\{A, C, G, T\}$ are both finite,

Figure 2.2: The LSH-ALL-PAIRS algorithm. Trivial subroutines referred to in the algorithm are: GEN-RANDOM-PROJ(ℓ, k), which generates a random projection by picking k values uniformly at random with replacement between 1 and ℓ; PARTITION(Φ), which partitions the elements of Φ into classes with the same projection value; and COUNT-SUBSTITUTIONS(s_i, s_j), which counts the number of substitutions between the ℓ-mers s_i and s_j. The parameters m and k, respectively the number of random projections to try and the number of sequence positions in each projection, must be chosen to achieve a particular sensitivity bound, as described in Section 2.3.4.

LSH-ALL-PAIRS(C, ℓ, d, m, k)

C: a collection of genomic sequence

ℓ: length of similarities to find

d: max. substitutions allowed in a similarity

m: number of random projections to perform

k: number of positions in each projection

A: set of all ungapped d-similarities found

$A \leftarrow \emptyset$

repeat the following m times

$f \leftarrow$ GEN-RANDOM-PROJ(ℓ, k)

/* compute projection of each ℓ-mer in C */

$\Phi \leftarrow \emptyset$

foreach sequence $c \in C$

for $1 \leq j \leq |c| - \ell + 1$ **do**

$s \leftarrow c\,[j \ldots j + \ell - 1]$

$\Phi \leftarrow \Phi \cup \{\langle f(s), \pi(s)\rangle\}$ /* $\pi(s)$ = position of s in C */

end

end

/* partition Φ into classes C_q by projection value */

$\{C_1, C_2, \ldots\} \leftarrow$ PARTITION(Φ)

foreach class C_q

foreach pair $\langle f(s_i), \pi(s_i)\rangle, \langle f(s_j), \pi(s_j)\rangle \in C_q$

if COUNT-SUBSTITUTIONS$(s_i, s_j) \leq d$

$A \leftarrow A \cup \{(\pi(s_i), \pi(s_j))\}$

endif

end

end

end

return A

by radix sorting by projection value[6]. Because k is typically a small constant (less than 20), we account the total cost of the filtering phase as $\Theta(mN)$.

The algorithm's checking time is, as usual, determined by the total number of candidate pairs, which is $\sum_q \Theta\left(|C_q|^2\right)$. If the sets C_q are all much smaller than C, the checking cost will be substantially less than the naive $|C|^2$. In practice, we can usually choose parameters that make the C_q's small – at most some tens of ℓ-mers each – so that the number of candidates checked is *orders of magnitude less* than N^2. For a more detailed, quantitative analysis of the number of spurious candidates, we must consider the optimal choices of the parameters m and k as described in the next section.

2.3.4 Performance Analysis

LSH-ALL-PAIRS works efficiently because it trades a linear amount of filtering work for a large reduction in the quadratic checking cost. Whether this tradeoff is advantageous depends on how tolerant the user is of false negatives. We assume that the user specifies a lower bound on expected sensitivity, in the form of an allowable false negative rate ρ_{fn}, and that we may freely choose the parameters m and k to maximize efficiency subject to this sensitivity bound. To find the most efficient parameterization, we need to predict the algorithm's false positive rate ρ_{fp} for various parameter values.

Qualitatively, the tradeoff that determines the values of m and k is as follows. We can increase the sensitivity of the algorithm in one of two ways: try more random projections by increasing m, or make each projection more effective at finding d-similarities. The latter effect can be achieved by decreasing the number of positions k per projection, since the chance of sampling a mismatched position in a d-similarity scales exponentially with k. However, both ways of increasing sensitivity also increase the total number of false positives. Each projection produces some number of spurious candidates, so increasing m linearly increases the total checking cost. Decreasing k exponentially increases the chance that two ℓ-mers which are *not* d-similar will spuriously project to the same value, simply

[6]Linear-time partitioning is possible because projection values are simply short strings. In effect, we reduce the inexact matching problem to a series of exact matching problems, each of which (as noted in Chapter 1) can be solved in linear time because exact matching is an equivalence relation.

because the projection function fails to sample a position at which they differ. In the limit, when $k = 0$, every pair of ℓ-mers in the input becomes a candidate. The following analysis shows how to balance these considerations of sensitivity and efficiency.

The False Negative Rate

We begin by giving a formula for ρ_{fn}. Let s_1 and s_2 be a d-similar pair of ℓ-mers. As shown by Inequality (2.1), a single randomly chosen projection sampling k positions will project s_1 and s_2 together with probability at least $(1 - d/\ell)^k$. Hence, the probability q_{fn} that s_1 and s_2 *never* project together under any of m independent projections is bounded by

$$q_{fn} \leq \left[1 - \left(1 - \frac{d}{\ell}\right)^k\right]^m \tag{2.2}$$

q_{fn} is computed for one d-similar pair, but by linearity of expectation, $q_{fn} = \rho_{fn}$, the expected fraction of *all* similar pairs missed by the algorithm after m iterations. Note that Inequality (2.2) holds with equality only for pairs of ℓ-mers that differ by exactly d substitutions; pairs differing by fewer substitutions are more likely to be detected.

Suppose we fix a particular value of k and wish to achieve a false negative rate of at most θ. Isolating m in the upper bound of Equation (2.2), we find that $\rho_{fn} \leq \theta$ iff

$$m \geq \frac{\log \theta}{\log\left(1 - \left(1 - \frac{d}{\ell}\right)^k\right)} \tag{2.3}$$

This bound determines, for any k, the smallest m that can be used without compromising (in expectation) a fixed sensitivity $\rho_{fn} = \theta$. The bound is consistent with previous intuition: as the projection size k increases, the denominator becomes closer to zero, so m increases. We must therefore trade more projections for more stringent filtering in each projection.

Finally, note that ρ_{fn} is independent of either the background distribution or the arrangement of substitutions – in particular, the longest word length – in the input's similarities.

The False Positive Rate

To estimate the false positive rate ρ_{fp}, we make the standard assumption of an i.i.d. background sequence model. Although this assumption is no better justified for our algorithm

than for any other, random projection is less sensitive than some methods to deviations from the i.i.d. model because the positions in each projection are often widely separated, limiting the effects of adjacent-base correlation. We report our results for an arbitrary base distribution, assuming that two bases in the background match by chance with probability ϕ. For equal base frequencies, ϕ achieves its minimum value of 0.25; except in unusual, highly biased genomes, it rarely exceeds 0.3.

Under the background distribution, the probability that two independent random ℓ-mers differ by exactly t substitutions is the binomial probability

$$\beta_{1-\phi,\ell}[t] = \begin{pmatrix} \ell \\ t \end{pmatrix} (1-\phi)^t \, \phi^{\ell-t}$$

This is also the chance that two *nonoverlapping* ℓ-mers from the background differ by t substitutions; we ignore rare but hard-to-analyze overlapping ℓ-mer pairs. The chance that the two ℓ-mers project to the same value in a single projection is $(1 - t/\ell)^k$. Applying the prior binomial distribution on t induced by our sequence model and summing over all $t > d$, we have that

$$q_{fp} = \sum_{t=d+1}^{\ell} \beta_{1-\phi,\ell}[t] \left(1 - \frac{t}{\ell}\right)^k \tag{2.4}$$

The probability q_{fp} is also the expected rate of false positives for one projection; hence, the algorithm's overall false positive rate ρ_{fp} is $m \cdot q_{fp}$.

Choosing k and m

Now that we can predict the expected values of ρ_{fp} and ρ_{fn}, we can specify how to optimally parameterize the LSH-ALL-PAIRS algorithm. Assume the user specifies ℓ, d, and a target false negative rate ρ_{fn}. For each value of k up to some large number (say, $k = 16$, large enough for a projection $f(s)$ to fill a 32-bit machine word), we first compute the false positive rate ρ_{fp} for the smallest feasible value of m, as determined by Inequality (2.3). These values form a curve as shown in Figure 2.3.

To pick the optimal point on the curve, we must choose k to minimize the total running time, which includes $\Theta(mN)$ time in filtering plus $\Theta(\rho_{fp}N^2)$ time in checking. The relative cost of filtering and checking depends on implementation constants that must be measured

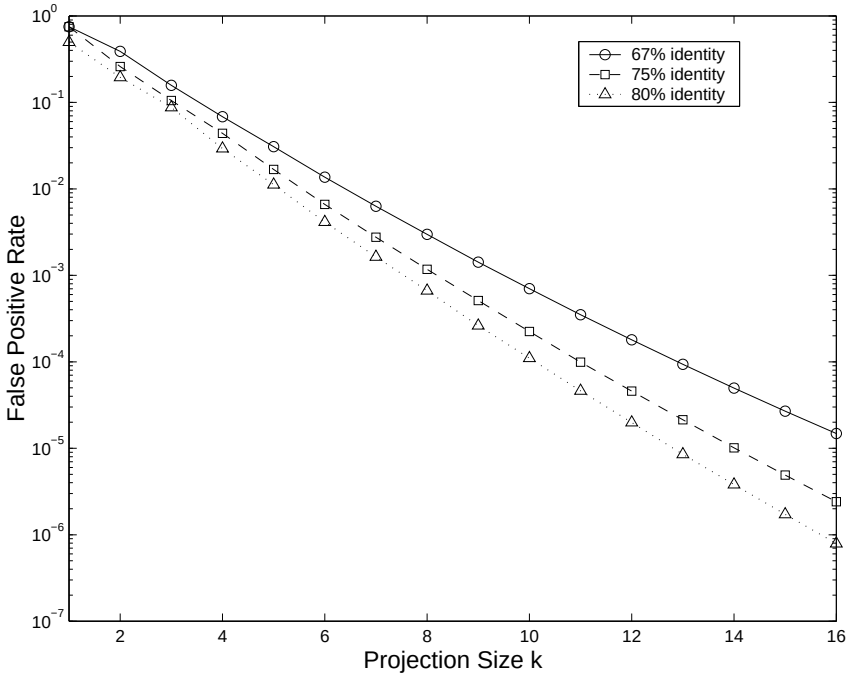

Figure 2.3: false positive rates as a function of projection size k for $\ell = 75$, $\rho_{fn} = 0.05$, and various levels of similarity, assuming background sequence with equal base frequencies. From top to bottom, the lines represent $d = 25$ (67% identity), $d = 19$ (75% identity), and $d = 15$ (80% identity). For each value of k, we used the smallest possible m as determined by Inequality (2.3) to compute the false positive rate.

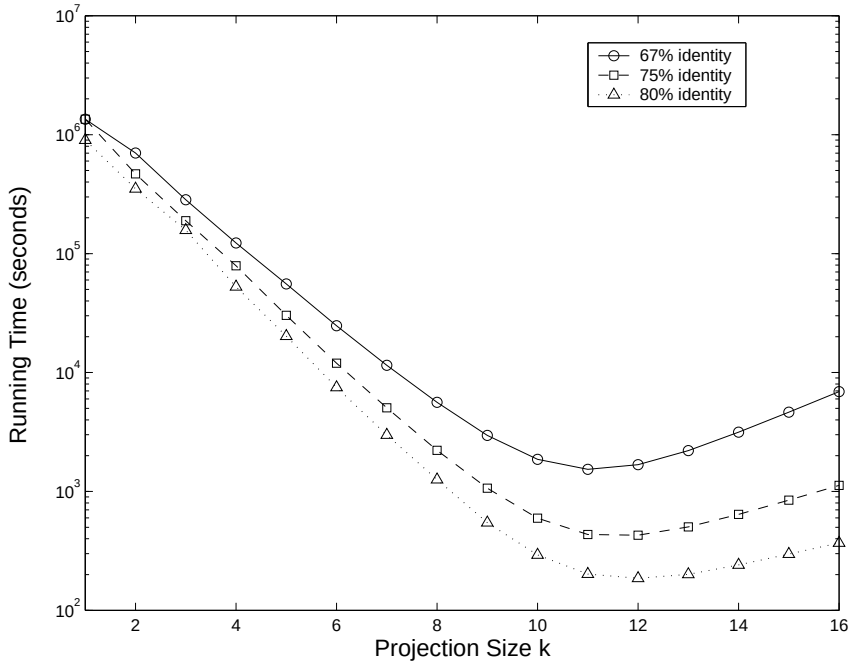

Figure 2.4: total computational cost as a function of projection size k for the parameters of Figure 2.3. We assume a comparison between two one-megabase sequences, which requires roughly 3.5 seconds per iteration on a 550 MHz Intel Pentium III workstation, and 1.8 μs per checked candidate pair, which is roughly accurate for the same machine. Minimal costs are achieved for $k = 11$, $m = 258$ at 67% identity; $k = 12$, $m = 99$ at 75% identity, and $k = 12$, $m = 43$ at 80% identity.

empirically. For simplicity, we generally measure the cost of projection once, then assume it is independent of the projection size k; a more careful cost estimate would model the cost increase with increasing k. Weighting both costs by their respective constants yields a total cost curve with a clearly defined minimum, as shown in Figure 2.4. We choose the values of k and m that achieve this minimum.

In practice, the optimal value of k for similarity thresholds of 67-75% and a false negative rate of 5% ranges from 10 to 16. The corresponding typical values of m are 150 to 600 projections.

2.3.5 Implementation of LSH-ALL-PAIRS

Reducing the LSH-ALL-PAIRS algorithm to practice requires some effort to avoid an inefficient implementation. Below, we provide both hints for maximizing the algorithm's efficiency and sensitivity and workarounds for cases where real genomic sequence does not fit the formal model. Because LSH-ALL-PAIRS is expected to work with very large sequences, limiting memory usage is an especially important implementation goal. We also describe how we postfilter the core algorithm's output to produce ungapped or gapped similarities that score highly under more general biosequence scoring functions.

Filtering

The first implementation decision in LSH-ALL-PAIRS is how to represent projection values. Since projections in practice generally sample sixteen positions or less, and the base alphabet has only four characters, a straightforward implementation might represent projection values explicitly using two bits per base, allowing a projection value to fit into a 32-bit machine word. However, this approach has two disadvantages: it imposes a hard upper limit on the projection size, and it does not handle extensions to the base alphabet, in particular the common use of "N" and "X" to indicate unknown or masked bases.

A more flexible representation of projection values uses a hash function h that maps k-mers to integers. LSH-ALL-PAIRS only needs to test projection values for equality, so any 1:1 mapping of k-mers to integers is acceptable. Even if h is not perfectly 1:1, collisions can only cause distinct classes C_q and $C_{q'}$ to be merged, increasing the rate of spurious candidates but preserving the algorithm's soundness. The advantage of hashing the raw projection values is that the implementation easily accommodates both large projection sizes and additions to the alphabet. In the common case where $k \leq 16$ and most ℓ-mers contain only the usual four bases, a good hash function into 32-bit integers should produce essentially no collisions and hence minimal loss of performance.

A second issue is how to implement the PARTITION subroutine of Figure 2.2, that is, how to efficiently partition the ℓ-mers of the input according to their projection value. In theory, this operation requires only $O(N)$ time, but partitioning methods that achieve this

bound have poor space efficiency and should be avoided unless they are necessary for other reasons. For example, Gionis et al., in their implementation of locality-sensitive hashing for database search [38], used a sparse indexing scheme whereby projection values are mapped to "buckets" on disk that store all objects in a given class C_q. Once the database is partitioned in this way, it supports efficient online search and update as new objects appear; however, the mapping has a high space overhead for sequences – as much as 15 to 20 bytes per base.

LSH-ALL-PAIRS does not need the online property of Gionis et al.'s bucket representation because all of its input is available offline at the start of the algorithm. The space overhead of sparse indexing is therefore unnecessary. A more space-efficient partitioning scheme builds a dense array of all input tuples $\langle f(s), \pi(s) \rangle$, then sorts them by their projection values $f(s)$. After sorting, the classes C_q are contiguous runs of tuples with the same projection value. Constructing and storing classes in this way requires only 8-12 bytes per base, depending on how compactly the sequence positions $\pi(s)$ are stored. Although linear-time sorting is possible given the limited length and alphabet of projection values, such a sort requires working storage equal to the input size, which is impractical in the (relatively common) case that the tuples take up more than half of available memory. We therefore choose instead to use an in-place quicksort, which takes time $\Omega(N \log N)$ but requires only $O(1)$ additional storage.

Finally, the filtering stage of LSH-ALL-PAIRS must take measures to deal with unusually large classes. Although classes larger than a few ℓ-mers are unlikely to occur purely through random variation in an ideal background sequence model with a sufficiently large projection size, several factors cause such classes to arise in practice. First, because the positions of a projection are sampled with replacement, the number of *unique* positions in a projection may occasionally be much less than the target value k. For such projections, the average class size is unusually large. Large classes can also arise when the background sequence departs strongly from the simple i.i.d. model, usually in one of two ways: first, when the sequence contains unmasked regions of low complexity, such as short tandem repeats, that generate many spurious candidates; and second, when the background contains long runs of X bases where repeats have been removed. The latter problem is the easier to solve, provided that X is always considered a mismatch to any other base (including another X).

Under this assumption, ℓ-mers containing more than d X's cannot be d-similar to anything and so may be discarded during tuple creation. Moreover, if a projection function samples an X from an ℓ-mer, the resulting projection value should be treated as unique, since no other ℓ-mer's projection value can match it in the position containing the X. A tuple whose projection value is equivalent to no other cannot participate in a candidate pair, so all such tuples may be discarded.

For large classes that do not arise because of X bases and are not obviously composed of low-complexity sequence, the most correct solution would be simply to process every candidate pair in the class. Unfortunately, the cost of processing large classes grows quadratically with class size, so this solution proves extremely expensive for classes with hundreds of tuples. Rather than simply discarding such classes, we reduce their size by sampling a random subset of their tuples and checking for d-similarities only between those ℓ-mers in the sample. If the class contains many d-similar pairs, the output will contain at least a representative sample of these similarities.

Checking

The checking phase of LSH-ALL-PAIRS generates a potentially quadratic number of similarities, many of which may be repeated because they are found by more than one iteration of the algorithm. Removing redundant similarities from the algorithm's output is always desirable; for the occasional problem, such as whole-genome repeat finding, that produces $\Theta(N^2)$ interesting similarities, removing redundancies is mandatory for the algorithm to run within a reasonable amount of space.

Eliminating multiple instances of the same d-similarity is a simple matter of remembering which similarities have been previously recorded. To minimize both space usage and the work done to eliminate spurious matches, we do not detect individual duplications during checking but rather remove them all at once after every few iterations of the algorithm.

An even better strategy than removing duplicates and overlaps is to avoid creating them in the first place. While we cannot avoid all such redundancies, a simple modification to checking can eliminate many of them. Specifically, we require that each candidate pair to

be checked must be *canonical* in the following sense: the two ℓ-mers being compared must start with a pair of matching bases, *and* the pair of bases (if any) immediately preceding the ℓ-mers must differ. Given any candidate pair, we can find a nearby canonical pair on the same diagonal using the following rule: if the ℓ-mer pair begins with a match, move its starting positions in each sequence backwards until a mismatch is encountered; otherwise, move the starts forwards until a match is encountered[7]. The following lemma shows that this procedure is sound:

Lemma 2.1 Let s_1 and s_2 be a pair of ℓ-mers. Then the nearest canonical pair contains at least as many matching base pairs as the pair (s_1, s_2).

Proof: We have two cases: either the pair (s_1, s_2) starts with a match, or it starts with a mismatch. If the pair starts with a match, let x and y be the bases immediately preceding s_1 and s_2 respectively. If $x \neq y$, the pair is canonical, and we are done. Otherwise, $x = y$, in which case the starting point moves backwards to encompass one matching base pair and drops the last base pair of (s_1, s_2). The total number of matching base pairs therefore remains the same or increases by one.

If the pair starts with a mismatch, moving its starting point forwards drops one mismatched base pair and adds the base pair following the ℓ-mers. This new base pair may or may not be a mismatch, so the total number of mismatched base pairs either remains the same or decreases by one.

Proving the initial claim is a simple inductive argument from the above two cases. ∎

Corollary 2.2 If (s_1, s_2) is a d-similar pair, the nearest canonical pair will also be d-similar.

Converting each d-similar pair to a canonical pair can collapse a number of nearby d-similar pairs into a single canonical pair. Overlapping similarities therefore become exact duplicates that are easily removed from the output. Even better, the nearest canonical neighbor of a spurious candidate pair may be a d-similarity, since shifting a pair's starting point may increase the number of matching base pairs it contains. Typically, a canonical pair

[7]We must still accept d-similar but noncanonical pairs in which one ℓ-mer occurs at the end of a sequence.

is found after inspecting only a few extra bases, so making every candidate pair canonical before checking it costs little while increasing the sensitivity of LSH-ALL-PAIRS.

Postfiltering

LSH-ALL-PAIRS addresses an abstract similarity search problem, but its results are not in the expected form for practical annotation tools. In particular, the algorithm produces only fixed-length alignments without indel mutations, and it is restricted to a single, trivial substitution score function. To work around these limitations, we postprocess the output of LSH-ALL-PAIRS in three steps – merging and extension, ungapped postfiltering, and optionally gapped postfiltering – to produce local alignments comparable to those output by BLAST or other similarity search tools. The first step removes the fixed-length limitation, while the second and third steps resemble the filtering approach taken by BLAST and related tools.

The merging and extension phase of postfiltering turns a collection of fixed-length, possibly overlapping d-similarities into a smaller set of variable-length, non-overlapping ungapped alignments. Merging simply coalesces overlapping d-similarities on the same diagonal into one alignment, while extension attempts to find additional, previously undiscovered d-similarities that occur near the known ones. The extension phase searches for additional d-similarities 500 bases upstream and downstream on the diagonal of each coalesced alignment. After merging and extension, the remaining distinct alignments are nonoverlapping and may be of any length. These alignments become *secondary candidates* that are subjected to roughly the same filtering treatment that BLAST applies to word matches.

Postfiltering, unlike merging and extension, has the goal of discarding similarities that are not significant by the standards of Karlin-Altschul theory [54], as well as making the output conform to the reporting standards of existing similarity search tools. Each secondary candidate is subjected to two filters: ungapped postfiltering and, if it survives, gapped postfiltering. Ungapped and gapped postfiltering both use dynamic programming to score candidate alignments with an arbitrary affine scoring function. The ungapped postfilter uses a linear-time dynamic programming algorithm, while the gapped postfilter uses *banded*

Smith-Waterman, a variant that does not consider alignments that deviate more than a fixed number of diagonals from their starting point. The ungapped postfilter uses only a substitution score matrix, while the gapped postfilter allows both substitution scoring and affine gap penalties. The results of each phase are filtered to discard alignments whose score is not significant. As with BLAST, there is no guarantee that every high-scoring local alignment in the input will contain a long ungapped d-similarity, so the formal sensitivity guarantees of LSH-ALL-PAIRS do not apply to postfiltered similarities.

One traditional question in postfiltering based on local alignment is how to bound the amount of sequence searched by the dynamic programming algorithm. In principle, an optimal alignment could have arbitrary length, which is problematic for algorithms like Smith-Waterman that work by filling in a fixed-size matrix or band. For ungapped filtering, we require that each computed alignment pass through an *anchor point*, which we arbitrarily take to be the center of the candidate's longest word match. This restriction does not seem odious – it essentially requires that the final ungapped alignment overlap the candidate. The optimal ungapped alignment passing through a fixed anchor point is simply the concatenation of the best alignments computed upstream and downstream from that point. To limit the amount of sequence searched, we extend the alignment in each direction only until its score falls below a fixed threshold value.

The anchor point approach to limiting the search region is also feasible for gapped postfiltering, and indeed is used for that purpose in NCBI BLAST 2.0 [5]. However, forcing a gapped alignment to pass through a fixed base pair has greater potential to arbitrarily restrict the search because the alignment path can vary in two dimensions. Restricting the alignment is dangerous because the optimal restricted alignment might score below the reporting score threshold, while the best unrestricted alignment would score above it. We therefore use the following more expensive procedure to determine a reasonable alignment length. We first find the best alignment in a band of width 101 centered on, and twice as long as, the ungapped candidate. We then compute further optimal alignments, doubling the length of the band each time, until no further improvement is observed in the alignment score. This procedure has the freedom to pick the best alignment length, but it is somewhat *too* free: it may extend the band so far as to find an alignment completely disjoint from

the candidate, perhaps corresponding to an adjacent sequence feature. To prevent a high-scoring alignment from shadowing nearby similarities, we require that each alignment found pass through the antidiagonal (but not the point) at the center of the ungapped alignment.

Finally, we note that, while our ungapped and gapped postfiltering is similar to BLAST's, its purpose is as much to produce similarities in the desired form as to filter for significance. Each d-similarity produced by the core LSH-ALL-PAIRS algorithm has a high prior probability of being interesting because we typically choose ℓ and d so that a d-similarity is highly unlikely to occur by chance alone. Hence, a large fraction of d-similarities lead to significant alignments even at high filtering stringency. In contrast, the candidates produced by word match filtering typically have a *low* prior probability of being interesting because the rate of spurious word matches is high. Word match algorithms *must* therefore implement highly stringent postfiltering to avoid reporting too many spurious similarities.

2.4 Experimental Results

We performed a variety of analyses to investigate different aspects of LSH-ALL-PAIRS' performance in annotating genomic sequence. We first tested how accurately we can predict the algorithm's cost and compared that cost to existing exclusion methods. We then compared the algorithm's practical sensitivity to that of word match filtering, to see if we can indeed detect meaningful similarities missed by the standard approach to similarity search. Finally, we performed large-scale analyses to investigate how well LSH-ALL-PAIRS scales to very long sequences in practice. Unless otherwise specified, all experiments were performed on a 550 MHz Intel Pentium III workstation.

2.4.1 Accuracy of Cost Estimate and Performance vs. Exclusion Methods

Proper parameterization of LSH-ALL-PAIRS requires accurately estimating the algorithm's cost, in particular the false positive rate ρ_{fp} due to random matches in background sequence. To validate the estimated false positive rate of Section 2.3.4, we compared the estimate to the false positive rate measured for actual DNA sequences drawn from the *Drosophila* genome, and to the estimated false positive rate of the double filtration algorithm for the

same comparison.

Two regions of approximately one megabase each were extracted from the Celera assembly of *Drosophila* [2], one from the left arm of chromosome 2 and the other from the right arm of chromosome 3. All repetitive elements identified by RepeatMasker were removed, resulting in two sequences of length one million ±10 bases. Comparison of these two sequences shows few significant local similarities at the 65% identity level ($\ell = 72$, $d = 25$); those that exist cover at most a few hundred bases and are consistent with either spurious matches or unmasked low-complexity sequences. The checking cost associated with these sequences is therefore due almost entirely to spurious candidates. An i.i.d. model parameterized with the sequences' base frequencies has a match probability $\phi = 0.254$.

We chose optimal parameterizations of LSH-ALL-PAIRS to analyze these sequences for levels of ungapped identity ranging from 95% to 65%, using $\rho_{fn} \leq 0.05$ and running time constants obtained on the test workstation as described in Section 2.3.4. The similarity length ℓ for each level of identity was chosen large enough to produce (in expectation) at most 1.5 spurious similarities.

Figure 2.5 shows, for each level of similarity tested, the predicted number of false positives and the total numbers of candidates checked in each of three runs of the algorithm performed with different random seeds. The false positive rate is reported as a fraction of 10^{12}, the cost of naively comparing all pairs of substrings in the input. The predicted and observed numbers of false positives scale roughly exponentially with the similarity threshold, as indicated by the dashed exponential trend line. Because the predicted false positive rates are only expectations, the measured numbers of candidates show some deviation from them; however, the predictions become progressively more accurate as the level of identity decreases. At 65% identity, the predicted cost agrees with experiment to within a factor of 1.25. The use of a simple i.i.d. background sequence model evidently does not strongly affect the accuracy of the prediction, perhaps because projections usually sample nonadjacent sequence positions and so, unlike word matching, are not strongly affected by correlations between adjacent bases.

The observed increase in prediction accuracy with decreasing similarity is a side effect of the fact that lower similarity levels require a greater number of filtering iterations m to

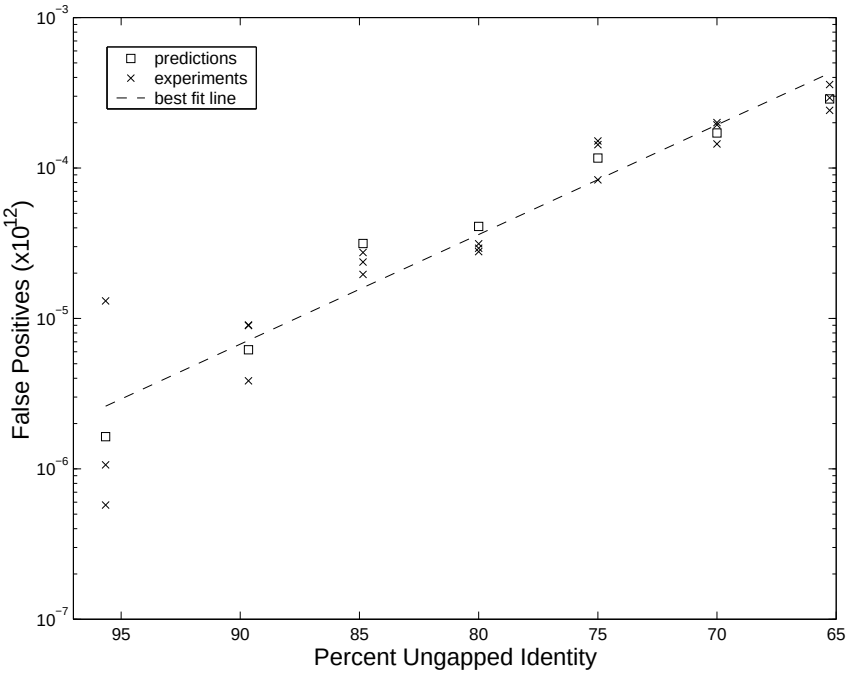

Figure 2.5: estimated and measured false positive rates versus similarity threshold for the best parameterization of the LSH-ALL-PAIRS algorithm on a pair of one-megabase extracts of the *Drosophila* genome. For these measurements, $\rho_{fn} \leq 0.05$, and the length ℓ for each similarity threshold was chosen so that fewer than 1.5 similarities are expected to occur by chance alone. Squares are predicted measurements, while x's are experimental trials with different random seeds. Dashed line shows best exponential fit to predicted points.

achieve a given false negative bound. The number of iterations grows from $m = 5$ at 95% identity up to $m = 499$ at 65% identity. Each iteration produces a number of candidates that depends on the projection chosen. Over a large number of iterations with randomly chosen projections, the total number of candidates, which is a sum of independent random variables, naturally converges to its expectation.

We also compared LSH-ALL-PAIRS' estimated false positive rate to the comparable rate for double filtration. An estimate of the latter rate, given in [77], is

$$\begin{aligned} \rho_{fp} &\leq \phi^{2r-\delta}\left[(\ell - r + d)(1-\phi) + d + 1\right] \\ \delta &= \left\lceil \frac{r-d}{d+1} \right\rceil \\ r &= \lfloor \ell/(d+1) \rfloor \end{aligned}$$

Using the values of ℓ and d from the figure, the false positive rate of double filtration starts to exceed that of LSH-ALL-PAIRS between 85% and 90% identity, with the gap between the rates growing as the percent identity decreases. At 65% identity ($l = 72$, $d = 25$), LSH-ALL-PAIRS obtains a significant advantage over double filtration, whose estimated false positive rate is roughly 0.4 – almost three orders of magnitude greater than the estimate in the figure. Moreover, the filtering stage of LSH-ALL-PAIRS, unlike that of double filtration, does not require enumerating all word and tuple matches of the worst-case length specified by Lemma 1.1. Not having to hash all 6×10^{10} expected dimer matches at 65% identity yields a substantial (and quadratic in N) practical performance advantage for LSH-ALL-PAIRS on these inputs. Double filtration is known to be more efficient for finding ungapped similarities than many other exclusion algorithms [42, Section 12.3.4], so LSH-ALL-PAIRS actually outperforms a large class of previous methods, not just this one.

2.4.2 Sensitivity vs. Word Matching

We have shown that LSH-ALL-PAIRS' filtering efficiency is predictable and that it exceeds that of deterministic exclusion algorithms at low similarity thresholds. To complement these results, we compared our algorithm's performance to that of word match filtering. A key test of LSH-ALL-PAIRS – indeed the test that addresses the overall goal set at the beginning

of this work – is whether our algorithm finds significant similarities that would otherwise go unreported by the word match-based annotation tools commonly used in practice.

The Competition

We compared LSH-ALL-PAIRS to Schwartz et al.'s PipMaker program [85], an all-pairs local alignment tool using W. Miller's `blastz` variant of the word match-based BLAST algorithm. Unlike other annotation tools such as NCBI BLAST, PipMaker is designed specifically to find local similarities in comparisons of two or more long genomic sequences. For the following tests, we used the Advanced PipMaker web server[8] with its default settings. We asked the server to return its results as gapped alignments for each similarity found, rather than as more compact but less informative percent identity plots (PIPs). We then compared the set of alignments returned by PipMaker to the output of our own gapped postfiltering code, fed with the results of the core LSH-ALL-PAIRS algorithm.

Although PipMaker works on any pair of genomic sequences, it has been especially well tuned for comparisons between the human and mouse genomes. In particular, Schwartz et al. developed a substitution score matrix and gap penalties for human-mouse comparisons to optimize PipMaker's ability to differentiate biologically meaningful similarities from spurious matches in the background sequence. To take advantage of this prior effort and to avoid unintentionally biasing the comparison against PipMaker, we chose in our tests to compare human and mouse genomic sequence with known orthologous features. We used PipMaker's own scoring function to postfilter the similarities produced by LSH-ALL-PAIRS and reported only those similarities that scored at least as high as the lowest-scoring similarity reported by PipMaker. The latter restriction was necessary because we do not presently have an implementation of gapped Karlin-Altschul statistics with which to set independent score thresholds for gapped alignments.

---

[8] *http://bio.cse.psu.edu/PipMaker*

The Problems

We investigated the sensitivity of LSH-ALL-PAIRS vs. PipMaker on three pairs of human and mouse sequences: the BTK locus [72] introduced in Chapter 1, the *mnd2* locus [52], and the T-cell receptor (TCR) alpha/delta locus [104]. The first two loci represent typical pairs of sequences that might be studied closely by biologists. They share the following properties:

- Both loci are disease-related. The BTK gene is implicated in human agammaglobulinemia, while the *mnd2* locus was originally identified on the basis of genetic linkage to motor neuron degeneration disease in mouse, a syndrome similar to Parkinson's disease in human.

- Both loci have been intensively analyzed and annotated in at least one organism, primarily because investigators wished to catalog all genes at the loci as *positional candidates* for the diseases being studied. The known annotations allowed us to identify which feature gives rise to each similarity found.

- Both loci contain extensive regions of human-mouse orthology, including several genes whose order is conserved between the two species.

The T-cell receptor (TCR) alpha-delta locus is one of several loci that determine the shapes of antigen-specific T-cell receptors in the vertebrate immune system. It contains two families of paralogous features – *V-segments* and *J-segments* – each of which codes for part of the T-cell receptor protein [104]. Sequence divergence among V- and J-segments is the key source of diversity used to produce diverse lineages of antigen-specific T-cells: each lineage's progenitor cell edits its genome to create a new TCR gene containing a randomly chosen V- and J-segment, along with a fixed constant or C-segment.

The TCR alpha-delta locus has a much different structure than the other loci studied here. Each member of the V- and J-segment families matches all other family members to some degree, resulting in a large number of similarities, not all of which arise from simple orthology. For this reason, the locus is useful for gathering quantitative statistics about how often each algorithm finds similarities not encountered by the other. Our other motivation

Table 2.1: sequence properties and parameters used in comparing LSH-ALL-PAIRS to PipMaker. "Unmasked" length counts bases remaining after preprocessing with RepeatMasker. The human and mouse TCR alpha-delta sequences were assembled from multiple GenBank entries by J. Roach; for mouse, the accessions used were: AC003057, AC003993–7, AC004096, AC004101–2, AC004399, AC004404–7, AC005240–1, AC005402–3, AC005835, AC005855, AC005938, AC005964, AC006119, AF259071–4, M64239.

Sequence	Accessions	Length (KB)	Unmasked (KB)	ℓ	d	k	m
BTK (human)	U78027	99.0	53.5	57	19	9	114
BTK (mouse)	U58105	88.9	59.7				
mnd2 (human)	AC003065	90.8	45.9	57	19	10	172
mnd2 (mouse)	AC003061	162.7	96.7				
TCR (human)	AE000658–62	1071.7	610.8	69	23	10	172
TCR (mouse)	(see caption)	1666.3	811.6				

for studying TCR alpha-delta was to improve its annotation, as part of a collaboration with J. Roach. The feature annotations used in this work were kindly provided by J. Roach for mouse and C. Boysen for human.

The Parameters

Table 2.1 gives the parameters used for each sequence comparison performed, along with information about the sequences being compared. Each sequence was first preprocessed with RepeatMasker[9] to remove known interspersed repeats, including many human *Alu* and mouse B1/B2 elements that would otherwise have dominated the set of similarities found. We then chose parameters for each comparison to find on average 95% of all similarities with at least 67% ungapped base identity. The similarity length ℓ and number of substitutions d were inferred from the 67% identity threshold, with ℓ chosen long enough that the expected number of spurious ungapped d-similarities of length ℓ in the background was less than one. The expectation was computed from an i.i.d. background sequence model, so the actual number of spurious d-similarities produced was probably somewhat greater than one.

[9] version 04-04-2000

Given ℓ, d, and the 95% sensitivity threshold, we chose the remaining parameters of LSH-ALL-PAIRS as described in Section 2.3.4. The lengths used to estimate the number of spurious similarities were the total numbers of unmasked bases in each sequence, which were easy to compute but slightly overestimated the true problem size. A more accurate estimate would account for the fact that the unmasked bases are not contiguous but rather appear as a number of smaller regions separated by masked repeats.

In postprocessing the d-similarities found by LSH-ALL-PAIRS, we used only a weak ungapped postfilter that passed every similarity with p-value less than 0.5 under ungapped Karlin-Altschul statistics using PipMaker's human-mouse substitution matrix. This weak prelude to gapped postfiltering is the default even for much longer sequences than those analyzed here because the core algorithm is already a highly efficient filter compared to the word match filter used by PipMaker and other BLAST-like algorithms. Again, the gapped postfilter used the same (more stringent) score thresholds as PipMaker.

The Comparisons: BTK and mnd2 Loci

Figures 2.6 and 2.7 illustrate the comparisons of the human and mouse BTK and *mnd2* loci, including any differences between the alignments reported by LSH-ALL-PAIRS and PipMaker. The significant alignments found by either program constitute the dot plots in the figures, while circles mark the locations of significant similarities found by LSH-ALL-PAIRS but not by PipMaker[10]. In these two examples, no similarities reported by PipMaker were missed by LSH-ALL-PAIRS.

LSH-ALL-PAIRS found eleven significant gapped similarities at the BTK locus, all in noncoding DNA, that were not reported by PipMaker. Three of these similarities fell in intergenic regions: one in the unexplained region of similarity at the $5'$ end of the sequence, and two in a probable enhancer region roughly 400 and 600 bases upstream of the FTP3 gene. One similarity fell in intron 1 of the FCI-12 gene, while the remaining similarities were distributed across several introns of the BTK gene. Previous experimental evidence [72]

[10]We say that a similarity represented by alignment $\mathcal{A}$ is not found by program P if the smallest rectangle enclosing $\mathcal{A}$ does not overlap the rectangle enclosing any alignment found by P. In other words, P must not report any evidence of association between the two sequence intervals covered by $\mathcal{A}$.

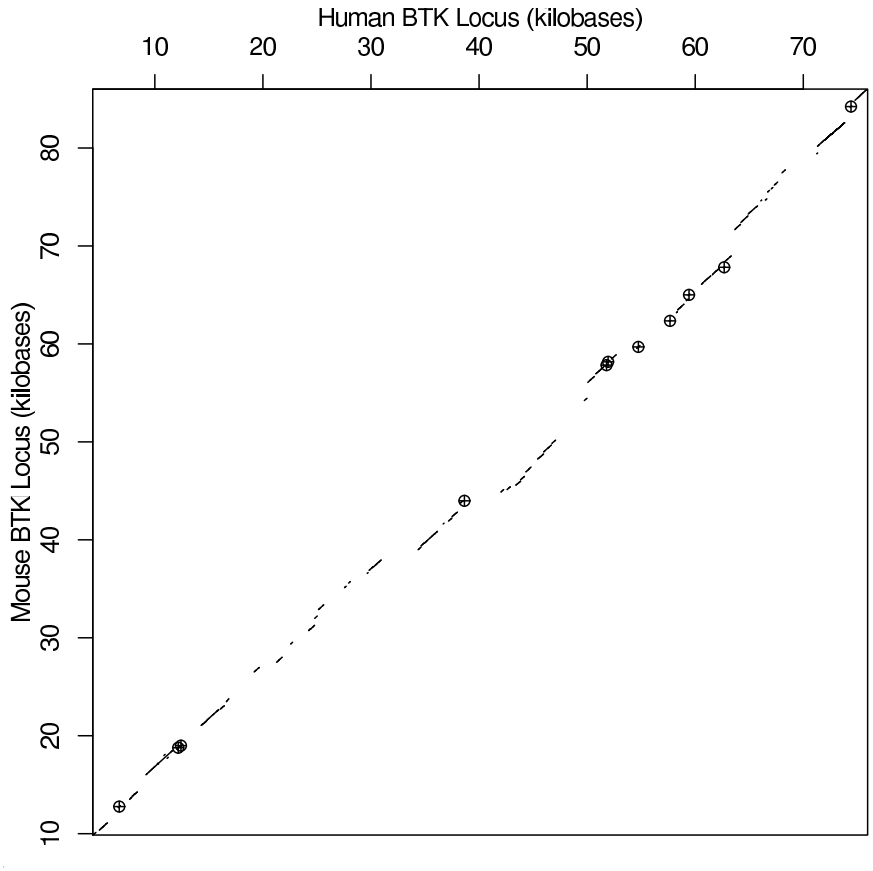

Figure 2.6: performance of LSH-ALL-PAIRS vs. PipMaker in comparing the human and mouse BTK loci. Circles indicate significant similarities found by LSH-ALL-PAIRS but not by PipMaker.

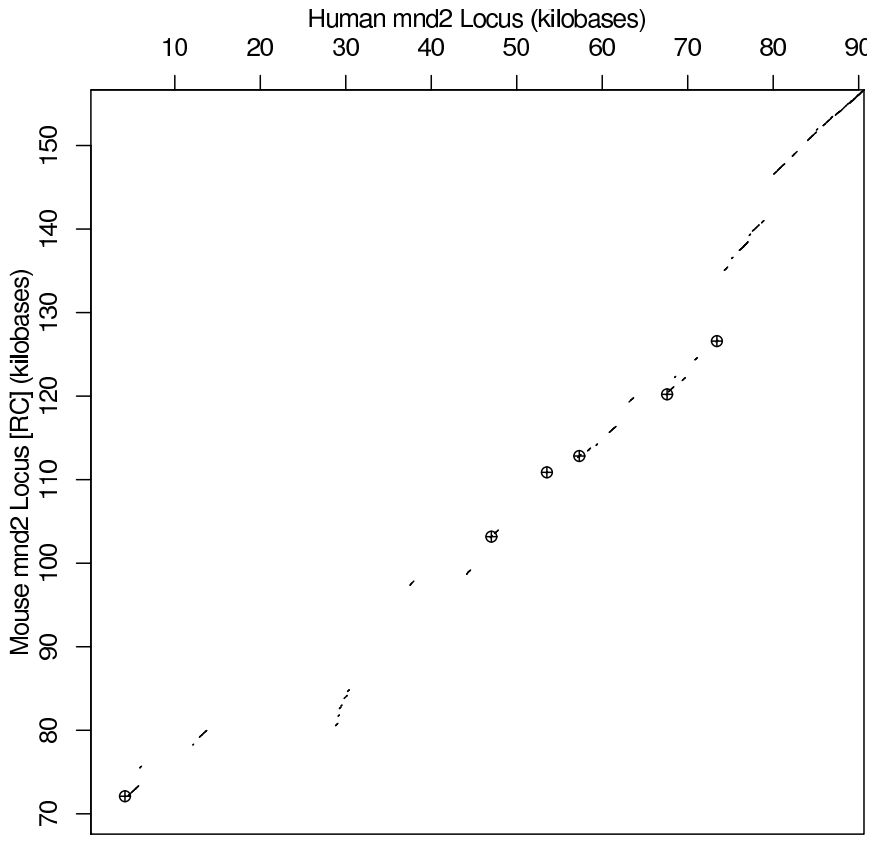

Figure 2.7: performance of LSH-ALL-PAIRS vs. PipMaker in comparing the human and mouse *mnd2* loci. Circles indicate significant similarities found by LSH-ALL-PAIRS but not by PipMaker.

suggests that BTK regulation depends on its intronic sequences, so the observed similarities could be evidence of regulatory regions.

At the *mnd2* locus, LSH-ALL-PAIRS found six noncoding similarities not found by PipMaker, all related to the gene D6Mm5e[11]. One similarity was in the gene's promoter region, about 200 bases upstream of its start, while the remaining matches were intronic. Intron 3 of the gene exhibited most of the observed similarities. D6Mm5e was first described by Jang et al. [52] in their annotation of the mouse *mnd2* locus; so far as we know, the function of this gene and its human ortholog remain unknown.

The alignments found only by LSH-ALL-PAIRS had lengths ranging from 83 to 401 bases. The substitution rates in their ungapped segments averaged 20 to 40%, with most falling in the range of 30-36%. The median length of the longest word match in these alignments was eleven bases, with 5/17 similarities having longest word matches of nine bases or less. The absence of long word matches was not strongly correlated to the alignment score, with several of the highest-scoring alignments having 8- or 9-mer longest words.

The alignments found by LSH-ALL-PAIRS support our hypothesis that biosequences contain significant noncoding similarities that are not detected by existing word match-based tools. Karlin-Altschul theory argues for the statistical significance of these alignments, while biological evidence for their meaning includes the fact that all the new alignments fall along the main diagonals evident in the dot plots, and in particular that they align orthologous promoter and intron sequences. Still unknown is whether the alignments represent *functional* features; while their positions in and near genes are consistent with possible regulatory function, they must still be shown to affect those genes' expression in an experimental setting. The evidence from LSH-ALL-PAIRS, along with prior evidence of regulatory sequences in the introns of the BTK gene, argues for performing the necessary experiments.

One surprise in the analyses of BTK and *mnd2* was that three of the novel alignments found by LSH-ALL-PAIRS had word matches of fifteen bases or more. We had expected that any significant alignment with such a long match would always be reported by algorithms based on word match filtering. While we can only speculate as to why these long word

[11] Genes were not annotated in the BAC sequences used for comparison, but annotations may be found in GenBank sequence AF084363.

matches did not result in a reported alignment, it seems likely that they were discarded during ungapped postfiltering. Two of the three alignments, along with two others having word matches of thirteen bases, scored low compared to most of the other novel similarities; the third contained a word match of length sixteen, but the reported alignment switches to a different diagonal almost immediately on either side of the match[12]. These observations suggest that the best ungapped alignments containing the long word matches may have scored too low or been too short to pass PipMaker's postfilter. In this respect, LSH-ALL-PAIRS has the advantage that the core algorithm produces long similarities with few false positives, so we can afford to be relatively permissive in postfiltering without passing too many spurious similarities.

The Comparisons: TCR Alpha-Delta Locus

Figure 2.8 shows a dot plot comparing the human and mouse TCR alpha-delta loci. The grid-like central portion of the plot consists of matches between V-segments, with each vertical column matching one human V-segment both to its mouse ortholog and to numerous paralogous segments. Other repeated features appearing on the dot plot include J-segments, which form the cluster at top right, and several olfactory receptor family genes at upper and lower left. The long conserved diagonal appearing between the V- and J-segment clusters is largely of unknown function, though it does contain a known enhancer sequence [47].

LSH-ALL-PAIRS found alignments involving almost all annotated features of TCR alpha-delta, including 57/57 annotated V-segments in human and 109/110 segments in mouse. In addition, some of the alignments reported by LSH-ALL-PAIRS matched known V-segments in one organism to unannotated sequence in the other; as a result of our analysis, the annotation files were updated to reflect the presence of additional segments. The new V-segments are consistent with the inferred history of sequence duplication and rearrangement at the locus but appear to be unexpressed pseudogenes.

The TCR alpha-delta locus, unlike the previous examples, contains significant similarities that were found by PipMaker but were missed by LSH-ALL-PAIRS. Figure 2.9 shows

[12] LSH-ALL-PAIRS found this alignment not because of the word match but because of a much longer adjacent segment of 51 bases with 67% ungapped identity.

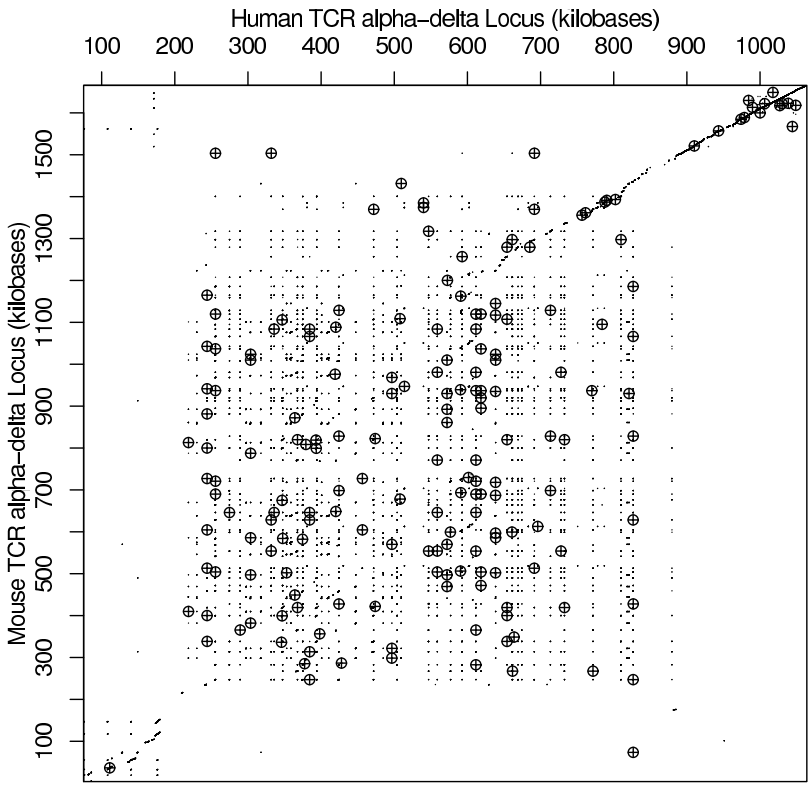

Figure 2.8: dot plot of the human and mouse TCR alpha-delta loci. Circles indicate significant similarities found only by LSH-ALL-PAIRS.

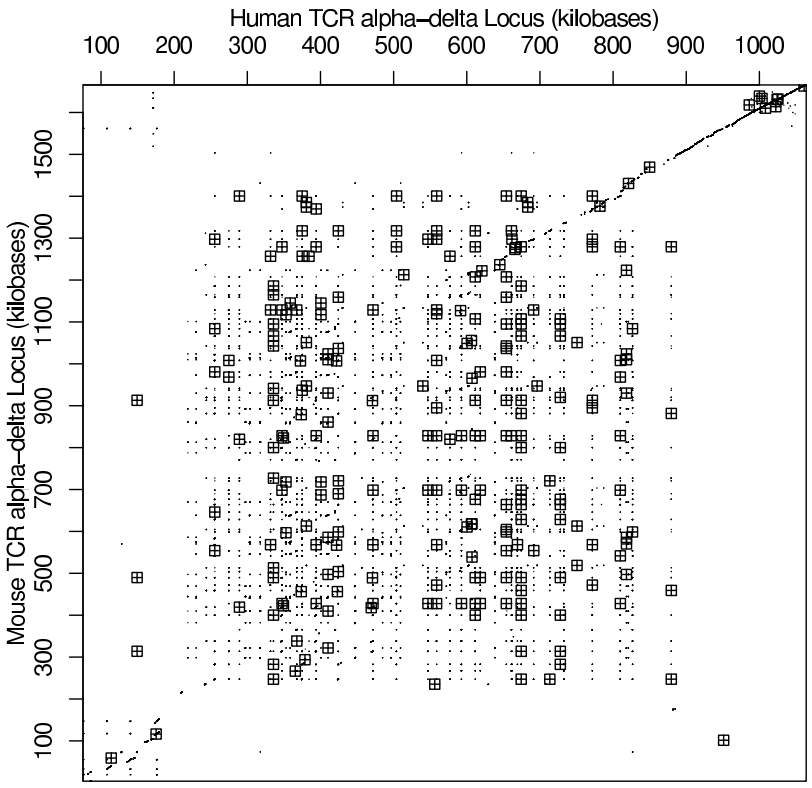

Figure 2.9: dot plot of the human and mouse TCR alpha-delta loci. Squares indicate significant similarities found only by PipMaker.

the locations of these alignments. Although LSH-ALL-PAIRS makes strong guarantees about its ability to find ungapped similarities, the figure illustrates that it is prone to missing significant gapped similarities that lack a sufficiently long (in this case 69 bases) ungapped segment. In contrast, PipMaker is sensitive to similarities with long word matches but does not demand that similarities contain even longer ungapped regions. Notwithstanding LSH-ALL-PAIRS' performance on BTK and *mnd2*, the word-based and projection-based approaches to annotation are in general *complementary*, each finding similarities not found by the other.

To quantify the importance of the similarities unique to each annotation algorithm, we assessed their impact on the number of observed associations between annotated V-segments in human and mouse. Because the V-segments form a paralogous family, each segment in

human should in principle show similarity to each segment in mouse. In practice, some instances have diverged too much to detect their similarity by either algorithm, but a nontrivial fraction of the 57×110 possible matches are still detectable. The number of V-segment associations found is a more biologically meaningful measure of sensitivity than the raw number of alignments, since the latter varies depending on how aggressively each program tries to extend its gapped alignments to encompass multiple regions of locally high similarity.

For counting purposes, we considered a V-segment to be the contiguous sequence interval covered by its three parts – two exons and a $3'$ recombination signal sequence (RSS) – plus the short intronic and intergenic regions between these parts. Empirically, most alignments between V-segments covered only the second exon and the RSS, excluding the more variable first exon. We considered two V-segments to be associated if some similarity aligned them over a region of least twenty bases, and if the number of matched base pairs over this region was at least 50% of its length (matches + mismatches + gaps). The length and identity thresholds were added to avoid counting associations between V-segments that were aligned for only a few bases or fell in a low-similarity region between two high-scoring parts of a long alignment.

LSH-ALL-PAIRS and PipMaker respectively found 1658 and 1725 associations between annotated V-segments. Of these associations, 113 (6.8%) were unique to LSH-ALL-PAIRS, while 180 (10.4%) were unique to PipMaker. Hence, each annotation algorithm produced biologically relevant associations not found by the other, with PipMaker being slightly more informative by itself.

To further quantify the sensitivity of LSH-ALL-PAIRS, we assessed whether it behaved "equivalently" to word matching of a given size, in the sense that using short enough words would find every similarity found by projection. For these experiments, we simulated word matching with word length w by running our algorithm with the degenerate parameters $\ell = w$, $d = 0$, $k = w$, and $m = 1$. We used the same postfiltering procedure on the word matches thus discovered as on the ungapped similarities produced by LSH-ALL-PAIRS. Table 2.2 shows the results for word lengths w ranging from eight to twelve bases. While word matching, like PipMaker, found V-segment associations not accessible to LSH-ALL-

Table 2.2: V-segment associations found by LSH-ALL-PAIRS vs. word matching, using equivalent postfiltering. "Total Assocs" = total associations found; "Only Word" = associations found by word matching but not by LSH-ALL-PAIRS; "Only LSH-ALL-PAIRS" = associations found by LSH-ALL-PAIRS but not by word matching.

Word Length	Total Assocs	Only Word	Only LSH-ALL-PAIRS
12	1192	120	586
11	1483	180	355
10	1685	239	212
9	1860	310	108
8	2056	442	44

PAIRS, our algorithm continued to produce associations not found by word matching even for word lengths as low as eight bases. For $w > 10$, LSH-ALL-PAIRS found more associations that word matching missed than vice versa.

In summary, the evidence gleaned from the TCR alpha-delta locus and the other examples indicates that LSH-ALL-PAIRS does in practice find biologically meaningful similarities that would be missed by word match filtering, even for quite short word lengths. Our algorithm does not share word match filtering's bias toward similarities with long words, making it a useful complement to standard methods for discovering features in DNA sequence. The evidence does not, however, support wholesale replacement of word match filtering by LSH-ALL-PAIRS because our method demands the presence of long ungapped matches that may not always appear in significant gapped similarities.

2.4.3 Performance on Large Sequences

We have argued that LSH-ALL-PAIRS enhances the sensitivity of DNA sequence annotation, and that it is substantially more efficient at low similarity thresholds than deterministic exclusion algorithms. However, the algorithm is still expensive compared to word match filtering; for example, the BTK and *mnd2* comparisons each required roughly one minute

(for forward and reverse-complement comparison together), while the TCR alpha/delta comparison required about thirty minutes. We must therefore demonstrate the practicality of scaling LSH-ALL-PAIRS to all-pairs local alignment problems as large as those considered at the beginning of this chapter.

Human chromosome 22, the shortest of the human chromosomes, is 34.6 megabases in length. After removing known interspersed repeats with RepeatMasker, roughly 17.4 megabases of sequence remain. To test LSH-ALL-PAIRS' performance on this moderately large sequence, we performed repeat finding at a 67% identity threshold on chromosome 22 using the Sanger Center's May 2000 assembly of the chromosome, which is an improved version of the sequence described by Dunham et al. in [33]. We compared this sequence to itself using LSH-ALL-PAIRS with $\ell = 81$, $d = 27$ to minimize spurious matches and $k = 14$, $m = 874$ to maximize efficiency. For ungapped postfiltering, we used the DNA-PAM-30 similarity matrix (+1 for a match, −1 for a mismatch). Because we have not implemented gapped Karlin-Altschul statistics, we did not perform gapped postfiltering, instead reporting any significant ($p < 0.05$) ungapped alignments. We did not compute matches between the sequence and its reverse complement.

Our analysis produced 41387 ungapped similarities, of which 36422 proved significant at $p < 0.05$ by the standard of Karlin-Altschul theory. The significant similarities, illustrated by dot plot in Figure 2.10, had a median length of 213 bases and a median identity of 79.2%. 8638 similarities, or approximately 24% of the total, had identities of at most 70%. In the figure, we identified clusters of similarities corresponding to a number of repetitive gene families on chromosome 22, the most striking of which is the immunoglobulin λ (Igλ) locus located between 5.9 and 6.9 Mb. Igλ contains families of paralogous gene segments similar to the TCR loci. Comparing our dot plot to that provided in [33], we also identified clusters corresponding to the glutathione S-transferase, β-crystallin, Ret-finger-protein-like, apolipoprotein, and APOBEC gene families. The vertical and horizontal bands in the figure correspond to the LCR-22 family of chromosome-specific repeats. We did not report the trivial match on the main diagonal, so the central line apparent in the figure actually consists mainly of approximate tandem repeats, some hundreds of bases long.

LSH-ALL-PAIRS required 11.9 hours to process chromosome 22. This length of time is

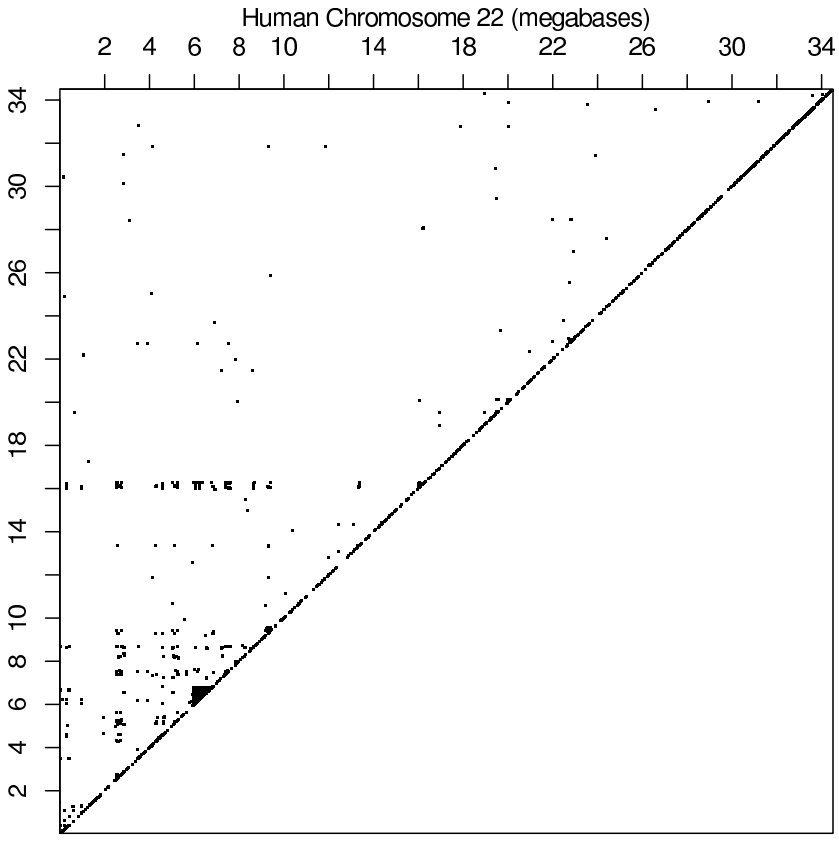

Figure 2.10: significant similarities ($p < 0.05$) detected in the self-alignment of human chromosome 22. The median identity of all similarities shown is 79.2%. Compare Figure 3 of [33].

reasonable for a large-scale analysis but is still slow compared to word match filtering: ungapped postfiltering of one candidate match takes a few microseconds on the same machine, so a similarity search based on 8-mer words could filter the expected number of matches in chromosome 22 in only a few hours. We therefore investigated whether LSH-ALL-PAIRS could be made more competitive in efficiency with word-based algorithms. In particular, because LSH-ALL-PAIRS' running time scales linearly with the number of iterations, we questioned whether the full number of iterations demanded by the analysis of Section 2.3.4 is really necessary in practice.

There is good reason to believe that the parameterization produced by formal analysis of LSH-ALL-PAIRS is conservative. First, the analysis assumes that every d-similarity has exactly d mismatches; better-conserved similarities are more likely to be found with each projection because they share more positions in common. Second, as discussed in Section 2.3.5, conversion to canonical ℓ-mer pairs increases the number of different candidate pairs that lead to the same d-similarity. Finding any one of these pairs is sufficient. Third, the longer a similarity is, the more opportunities exist for LSH-ALL-PAIRS to discover it. For example, using the parameters given above, an 82-mer with 27 mismatches contains two 81-mers, each of which may independently be discovered by any projection. The similarities found by LSH-ALL-PAIRS averaged over 200 bases in length with 67% or greater identity, so many (though not necessarily all) 81-mer substrings of them are likely to be at least 67% identical. All of these factors led us to the hypothesis that meaningful similarities are likely to be found in fewer iterations than predicted analytically.

To test our hypothesis, we compared the number of similarities in chromosome 22 found in the algorithm's full $m = 874$ iterations with the numbers found after 50 and 100 iterations – about 6% and 12% of the total running time. Because we have focused on LSH-ALL-PAIRS' sensitivity at low levels of similarity, we restricted attention to the most difficult similarities, those with at most 70% identity. Table 2.3 shows for each number of iterations the numbers of similarities found at two points: after running the core algorithm with canonical pair conversion, and after postfiltering. Both canonical pair conversion and postfiltering did indeed substantially increase the rate at which similarities were recovered, compared to the pessimistic predictions of formal analysis. Although the theory predicts that only about

Table 2.3: similarities with identity at most 70% found by LSH-ALL-PAIRS in human chromosome 22 after 50 and 100 iterations vs. the full 847 iterations required by formal analysis. Types of similarities shown are: raw d-similarities, d-similarities after conversion to canonical pairs, and significant ungapped alignments after merging, extension, and postfiltering. Percentages are computed with respect to the value at 847 iterations. The percentages for raw d-similarities are estimated from Inequality (2.2) because LSH-ALL-PAIRS does not compute such similarities directly.

Sim Type	847 Iters	50 Iters	100 Iters
d-similarities	–	(16.6%)	(30.6%)
Canonical Pairs	127449	67582 (53.0%)	97825 (76.8%)
Significant Sims ($p < 0.05$)	36422	35750 (98.1%)	36244 (99.5%)

30% of raw d-similarities found in 847 iterations would be present after 100 iterations, the number of canonical pairs found in this time was more than 75% of the total, and the number of significant similarities was over 99%. The accelerated recovery of similarities observed for chromosome 22 was also seen in the BTK, *mnd2*, and TCR alpha/delta comparisons.

Taking advantage of LSH-ALL-PAIRS' faster-than-expected recovery of similarities has the potential to reduce its running time in practice by nearly an order of magnitude. To exploit this phenomenon, however, we need a less conservative way to choose the number of iterations m than the theoretical performance curves of Section 2.3.4. Fixing a lower number of iterations *a priori* is difficult because the effects of canonical pair conversion and postfiltering are both challenging to model and input-dependent. Instead, we propose the following empirical procedure to determine when the core algorithm has "converged" to a nearly complete answer. After each few iterations of LSH-ALL-PAIRS, postprocess any new d-similar pairs that have been found since the last postprocessing phase and do not fall in sequence regions already covered by alignments. Count the total number of new significant similarities found each time; when it falls below a given threshold, stop and declare that the search has converged. The theoretical estimate of m would still serve as an upper bound on running time. This enhancement requires some future changes to our present separate implementations of LSH-ALL-PAIRS and postfiltering but seems to pose no obvious

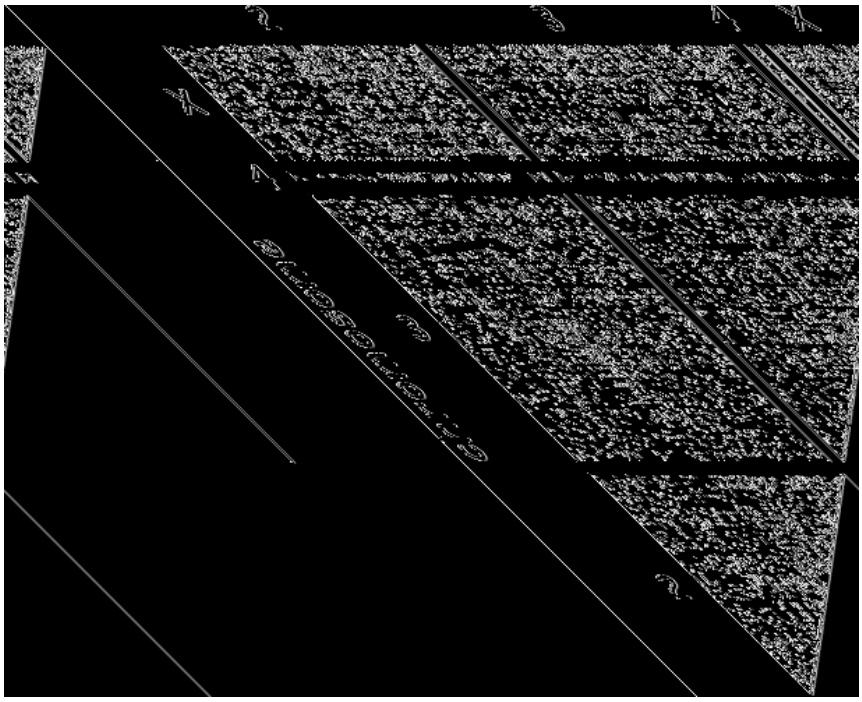

Figure 2.11: whole-genome repeat finding in *Drosophila* (116 Mb) with a 67% ungapped identity threshold.

engineering challenges.

As a final demonstration of the feasibility of running LSH-ALL-PAIRS on a large sequence, we performed whole-genome repeat finding on the *Drosophila* genome. The Celera assembly of this genome [2] contains 116 megabases of euchromatic sequence. After removing a relatively small number of repeats known to RepeatMasker, we compared each of the genome's six chromosome arm assemblies to itself and to the other arms at a 67% identity threshold, using $\ell = 81$, $d = 27$, and the usual choices of other parameters. (The number of iterations in each comparison was still chosen using the conservative theoretical prediction.) The comparisons, in both forward and reverse-complement directions, were run in parallel on seven Sun UltraSparc-80 processors, requiring roughly 1.5 weeks overall. Figure 2.11 shows the resulting similarities, including the usual tandem repeats near the diagonal and bands at the centromeres of chromosomes 2 and 3 where the repeat density sharply increases.

The whole-genome analysis of *Drosophila* is noteworthy not only because it shows that

LSH-ALL-PAIRS can process a genome of 10^8 bases but also because it exposes at least three future opportunities to improve the way the algorithm is implemented and used. The following enhancements will help to expand LSH-ALL-PAIRS' capability to 10^9 bases and beyond:

1. The version of LSH-ALL-PAIRS used for the experiment always allocated twelve bytes per input base, storing both sequence number and sequence position as separate integer values. For such large comparisons, a 33% memory savings (and therefore a performance improvement) can be had by concatenating the sequences together and treating them as one long sequence, reducing the storage to eight bytes per base. We observed a nontrivial running time improvement after applying this optimization to the self-alignment of chromosome 22, where there was no need to store sequence numbers.

   Depending on the cost of hashing ℓ-mers to their projections, it may even be worthwhile to recompute projection values for each ℓ-mer on the fly during partitioning, reducing the space cost to only four bytes per base.

2. To avoid monopolizing the memory of the busy servers used for the analysis, we parallelized it across processors by comparing each pair of chromosome arms separately. This approach resulted in small analyses compared to the total genome size but entailed much redundant work because, for example, the ℓ-mers of arm 2L had to be partitioned once per iteration for comparison with each of arms 2R, 3L, 3R, 4, and X. Concatenating the entire genome into a single sequence in one analysis would have reduced the number of times arm 2L was partitioned by a factor of five.

   This observation raises the question of how best to parallelize LSH-ALL-PAIRS. Because each iteration is independent, it is possible to attain linear speedup in partitioning and checking by splitting the iterations across p processors. Of course, this technique requires $\Theta(pN)$ memory for a problem of size N, p times as much as for a serial analysis. In contrast, parallelizing the comparison by splitting the sequence into fragments, creating p smaller comparison problems, results in only a $\Theta(\sqrt{p})$ speedup but uses

only $\Theta(\sqrt{p}N)$ memory. Moreover, the total cost of the fragment comparison problems may be less than predicted if their smaller size substantially improves memory locality. The best space-time compromise in parallelizing LSH-ALL-PAIRS remains a topic for future study.

3. If we were doing serious repeat family identification in *Drosophila*, we would not want to do the entire comparative analysis at once. Rather, the most common repeats could be identified and removed using an inexpensive comparison with a high similarity threshold (say, 90%), after which increasingly more sensitive analyses would find more difficult repeats. Because the cost of LSH-ALL-PAIRS scales exponentially with the similarity threshold, doing a whole series of analyses at gradually decreasing levels of similarity would not cost much more than doing only the most sensitive analysis. Removing the most common repeat families early on would also reduce the effective sequence size for later comparisons.

2.5 Conclusions and Open Questions

In this chapter, we have introduced, analyzed, and implemented the LSH-ALL-PAIRS algorithm for ungapped all-pairs local alignment using filtering by random projection. By adopting a randomized approach to similarity search, we break through the curse of dimensionality to achieve substantially better performance than deterministic exclusion algorithms such as double filtration, at levels of similarity as low as 67% ungapped identity. In practice, LSH-ALL-PAIRS complements word match filtering techniques for annotation because it finds biologically meaningful similarities that are difficult to detect using even relatively short word lengths. Finally, we have shown that the algorithm does indeed scale to problems as large as 10^8 bases and outlined an approach to achieve close to an order of magnitude future speedup by observing how quickly the similarity search converges.

We raise three major open questions regarding future extensions of LSH-ALL-PAIRS. Firstly, is uniform random sampling with replacement the best way to choose random projections? Secondly, can LSH-ALL-PAIRS be extended to find gapped alignments directly, rather than relying on heuristic gapped postfiltering? Finally, can the framework described here

be extended to finding high-scoring *multiple* local alignments in long genomic sequences?

2.5.1 Better Distributions of Projections

For simplicity of analysis, LSH-ALL-PAIRS chooses each projection by picking k positions uniformly at random with replacement. In principle, projections could instead be drawn from an arbitrary distribution Ψ. An open question is:

> Does there exist a distribution Ψ on projections that lowers the total cost of LSH-ALL-PAIRS compared to the current distribution, for some (or all) values of the parameters ℓ and d, a given bound on ρ_{fn}, and similarities with adversarially distributed substitutions?

The last condition, adversarially distributed substitutions, is important because LSH-ALL-PAIRS guarantees its sensitivity even in the adversarial worst case. Lower-cost filtration methods, such as deterministic word matching, may be provably sensitive if the distribution of mutations is known in advance.

An important special case of the above question is whether choosing positions in each projection without replacement is better than choosing them with replacement. If LSH-ALL-PAIRS chooses k positions per projection without replacement, the probability that any d-similar pair (s_1, s_2) projects to the same value under a random projection f becomes

$$\Pr[f(s_1) = f(s_2)] \geq \frac{\binom{\ell - d}{k}}{\binom{\ell}{k}}.$$

For any fixed ℓ, d, and k, this probability is less than that of Inequality (2.1), so a larger number of iterations m is required to achieve the same bound on false negatives. However, each iteration produces fewer false positives.

We conjecture that, for sequences more than a few tens of kilobases long, choosing positions without replacement is at least slightly more efficient than picking them with replacement. If we fix ℓ, d, and k and choose the smallest m that achieves a given bound

on the false negative rate, the total false positive rate over all m iterations is empirically lower when choosing positions without replacement. Moreover, if we simultaneously vary both m and k, as in Section 2.3.4, the optimal parameterization when positions are chosen without replacement empirically appears less costly under the measured cost constants for partitioning and checking. We still seek a formal justification to back up these empirical observations.

A related question to the above is whether one can generate a set of m projections, either deterministically or at random, that jointly have a lower false negative rate than m independently chosen random projections while maintaining about the same expected algorithmic cost. It is trivial to satisfy the first condition – for example, the gapped k-tuples of double filtration have a false negative rate of zero – but more challenging to satisfy both conditions at once. As a less trivial example, choosing m projections (uniformly at random without replacement) that share no position in common is always more sensitive than choosing m projections independently. Comparing the probabilities of failure to find a d-similarity in any of m trials, we see that nonoverlapping projections are less likely to fail than independent projections:

$$\left[1 - \frac{\binom{l-d}{k}}{\binom{l}{k}}\right]^m \geq \prod_{i=0}^{m-1} \left[1 - \frac{\binom{l-d-i\cdot k}{k}}{\binom{l-i\cdot k}{k}}\right].$$

However, the total false positive rate for nonoverlapping projections is also higher because every pair of ℓ-mers with at most $\ell - k$ substitutions passes the filter more often on average. Whether choosing disjoint projections is actually beneficial depends on how many fewer iterations are needed to achieve a given false negative rate. Moreover, only $\lfloor \ell/k \rfloor$ nonoverlapping projections of size k can be chosen for similarities of length ℓ. For larger values of m, it will be necessary to devise and analyze the effectiveness of schemes in which projections have minimal but nonzero overlap.

The above theoretical questions attempt to maintain a sensitivity guarantee for LSH-ALL-PAIRS independently of how substitutions are arranged in a similarity. In practice, it may sometimes be useful to design a distribution of projections that is especially well-adapted to

finding certain kinds of sequence features. For example, to find conserved coding DNA, one could model its three-base codon structure and the fact that third base positions are poorly conserved by selecting only projections whose positions are 0 or 1 mod 3. Some offset of each projection against a coding sequence would not sample any third base positions. Projections could also be specially designed to find regulatory elements with constant-width spacers, which are often seen in yeast [89]. Such features consist of two short conserved regions separated by a completely unconserved spacer element of known width. To identify such features, projections should sample positions only from the ends of each ℓ-mer, not from its center.

2.5.2 Finding Similarities with Gaps

The core LSH-ALL-PAIRS algorithm only finds ungapped similarities. While postfiltering can produce gapped alignments, it does not maintain any formal guarantee of sensitivity. Some deterministic exclusion methods, such as Chang-Lawler, handle gaps with guaranteed sensitivity but share the poor scalability of their ungapped analogs. It remains an open question whether there exists a randomized algorithm for finding gapped similarities that both provides a formal sensitivity guarantee and scales as well as LSH-ALL-PAIRS to low similarity thresholds.

Random projection seems ill-suited to find alignments with gaps because each projection is assumed to index *corresponding* positions of every ℓ-mer. This correspondence is not fixed in the presence of indel mutations. Consider, for example, a search for alignments of an ℓ-mer to an $\ell + 1$-mer with a single deletion in the latter: there are $\ell + 1$ ways to delete a base from the longer string, each of which results in a different correspondence between the remaining aligned bases. To find a good correspondence, we must therefore hash $\ell+1$ ℓ-mers for every $\ell + 1$-mer in the input. For any fixed projection that samples k positions, these ℓ-mers produce at most $k + 1$ distinct projection values, but even with this observation the partitioning cost allowing a single gap is about k times the cost of LSH-ALL-PAIRS without gaps. The number of possible correspondences, and therefore the work factor, rises rapidly with the number of indels, so this straightforward approach to permitting gaps

seems unlikely to be practically efficient.

A projection-like technique that handles gaps efficiently is unlikely to rely on projections into subspaces of the Hamming space of ℓ-mers. More elaborate mappings might be needed to preserve gapped edit distance between sequences; one such mapping was recently found by Muthukrishnan and Ṣahinalp [66], but it corresponds to an edit distance based on operations, such as block reversals, that are not part of the usual biological definition of sequence similarity. Another property of sequences that is correlated to their gapped similarity is the set of k-mers present (at any position) in both strings [41]. LSH-ALL-PAIRS can be extended to work with this measure of similarity as follows. To specify all the k-mers present in a string s, first define an arbitrary bijection from k-mers to integers between 1 and 4^k, then create a vector of length 4^k, such that the ith position of the vector is nonzero iff s contains the ith k-mer. The more similar two ℓ-mers are, the more positions will match between their derived vectors. Pairs of similar vectors can be found using LSH-ALL-PAIRS by treating each vector as a 4^k-mer over the alphabet $\{0, 1\}$. The future challenge is to show that k-mer content, or any other property related to gapped edit distance between sequences, yields a search algorithm as sensitive and efficient for gapped similarities as projection is for ungapped similarities.

An interesting restriction of general gapped alignment would be to find alignments with one gap of arbitrary length. This restricted problem might be easier than handling arbitrary gaps and would still be useful for finding, e.g., palindromic regulatory sites with a variable-length spacer element [62] or tandem repeats with a block added or removed between the repeated sequences.

2.5.3 Long-Range Multiple Local Alignment

LSH-ALL-PAIRS is designed to find pairwise local alignments between sequences. In some applications, however, the problem is to find not pairs of similar sequences but sets of three or more sequences that are *mutually similar*. For example, most regulatory elements are so short – only a few tens of bases – that a pairwise alignment between two copies of the element does not score highly enough to pass even a generous significance threshold. If,

however, we could show that (say) three 20–30 base sequences are all similar, either to each other or to a common *consensus* or other abstract feature model, the sequences could be identified as occurrences of a significant feature.

While we will have more to say about multiple alignment of features in short background sequences in Chapter 3, this section is devoted to a different problem: long-range multiple local alignment. Whereas the motif finding methods of the next chapter do not scale well beyond a few tens of kilobases of background sequence, the long-range problem seeks multiple alignments in backgrounds hundreds or thousands of kilobases in length. The intended application of long-range multiple local alignment is to search long intronic or intergenic regions for features, especially regulatory elements, that may be too short to be identified as significant by pairwise alignment alone.

Finding the optimal multiple alignment of three or more sequences under any of a variety of different scoring criteria is an NP-hard problem [102]. The full cost of aligning three or more sequences is therefore even greater in practice than that of applying Smith-Waterman to pairs of sequences, and the need is even more acute for an efficient filtration scheme that minimizes the need for expensive multiple alignments. Developing such a scheme – including specifying the criteria for a biologically plausible multiple alignment, devising and analyzing a filter, implementing it efficiently, and finding biosequences to serve as test cases – is a major open avenue for research.

We end by briefly describing a heuristic extension of LSH-ALL-PAIRS to long-range ungapped multiple alignment. The pairwise algorithm's filter requires that each candidate pair of ℓ-mers match in k positions. To align n sequences, one could trivially extend the filtering criterion to require that *all* sequences match in k positions. The revised criterion, while not nearly as sensitive as those used by most motif finding algorithms, has the advantage that it is easy to apply even to long sequences: first apply the partitioning step of LSH-ALL-PAIRS to the input, then draw n-tuples rather than pairs of ℓ-mers from each class C_j as candidates.

The above filtering heuristic is admittedly crude, but it shows preliminary evidence of working in practice. We applied this filter to a set of three orthologous sequences from the prion protein (PrP) locus that were previously collected and annotated by Lee et al. [59].

The sequences, drawn from human, mouse, and sheep, ranged from 32 to 36 Kb in length. We filtered these sequences for matches of length $\ell = 30$ with at most $d = 10$ positions that were not conserved in all three organisms; such matches are unlikely to occur by chance alone in three sequences of these lengths. We somewhat arbitrarily chose parameters $m = 100$ and $k = 8$. The output of the filter included matches across all three organisms corresponding to three different noncoding regulatory sites annotated by Lee et al.. The fact that these features were previously found by *manual* multiple alignment strongly suggests that even a simple filter like ours has the potential to increase the automation and sensitivity of long-range multiple alignment.

Chapter 3

FINDING MOTIFS USING RANDOM PROJECTION

In this chapter, we show how to extend random projection from its original use in pairwise comparison to *multiple* alignment – comparisons among three or more sequences. We focus specifically on the widely studied problem of finding regulatory motifs in genomic sequence by ungapped multiple local alignment. We develop a new algorithm, the PROJECTION motif finder, and show that it solves problems that are intractable to existing tools based on the popular local search approach to finding motifs.

3.1 Problem Definition

Section 1.2.2 introduced transcription factor binding sites, an important class of feature in genomic DNA. These sites, which range up to a few tens of bases in length, enable transcription factors to bind near genes and alter their expression. Knowing which genes have nearby binding sites for which factors is key to understanding how the cell controls and coordinates gene expression to maintain homeostasis, to respond to environmental changes, and to carry out programs like replication or programmed cell death. Binding sites occurring inside genes, especially in the $5'$ and $3'$ *untranslated regions (UTRs)* surrounding the coding sequence, may also be recognized in the mRNA by molecules other than transcription factors, including parts of the spliceosome or ribosome.

A *motif* is a conserved DNA sequence pattern recognized by a transcription factor or by other cellular machinery[1]. As we saw in the human-mouse pairwise comparisons of Chapter 2, conservation of a regulatory motif across organisms or across genes allows us to identify it through similarity search. However, because regulatory motifs are so short and are

[1]The term "motif" is also commonly used to describe a pattern in protein sequence, but protein motifs may arise from many types of conserved functional domain, not just binding sites. Even DNA motifs may arise for reasons other than binding sites (e.g. intron branch sites).

imperfectly conserved, two occurrences of a motif by themselves may not provide significant evidence of conservation. For example, consider the problem of finding two occurrences of a conserved 20-mer motif that differ by only five substitutions, in a pair of 1-kb background sequences that are i.i.d. random with equal base frequencies. The expected number of 20-mer matches with at most five substitutions appearing by chance in the background is about 3.67, so two occurrences of the motif would be indistinguishable from the background. Unless we can localize the motif to a very much smaller region, the only way to demonstrate its significance is to find additional occurrences in other sequences.

3.1.1 Motif Models and Measures of Conservation

Before we can formulate the problem of finding a conserved motif in multiple sequences, we must specify just what it means for three or more sequences to be jointly conserved. Many different quantitative measures of conservation exist for multiple sequences [42, Chapter 14], but we focus on measures induced by two abstract models of a motif: the *consensus model* and the *profile* or *weight matrix model (WMM)*. These models induce measures of conservation as follows: given a multiset $S = \{s_1, \ldots s_t\}$ of t putative motif occurrences, first find the motif model $M(S)$ that best describes S, then measure how well the motif occurrences s_i conform to the model $M(S)$.

Both the consensus and weight matrix models assume that two occurrences of a motif may differ only by substitutions, not by indels. This assumption reflects in part the limitations of many computational technologies (including our own) for finding motifs and in part the fact that biologically interesting motifs are frequently ungapped. Some known motifs consist of a small number of ungapped segments with intervening variable-length spacers [89, 62]; such motifs can be modeled as a collection of ungapped consensus or weight matrix models whose occurrences always appear near each other with gaps of varying length in between.

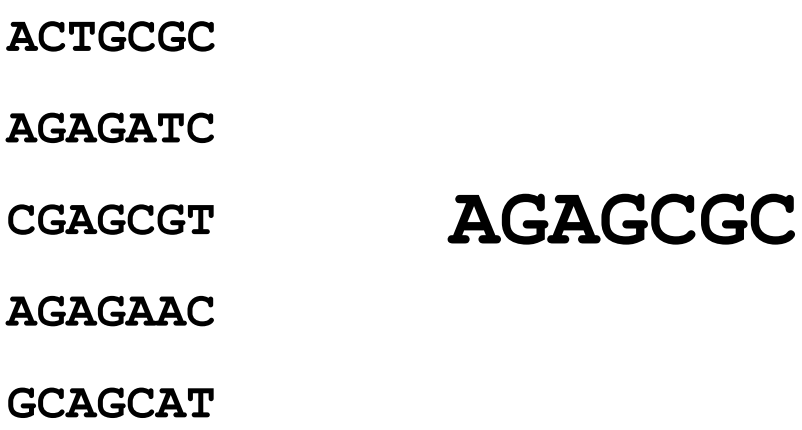

Figure 3.1: A consensus model inferred from five occurrences of a 7-mer motif. The most frequent base in position j of the occurrences becomes the jth base of the consensus. If two bases appear equally often in a given position, as with A and G in position six, the choice of the consensus base at that position is arbitrary.

The Consensus Model

The consensus model is a simple combinatorial description of a motif. In this model, the motif is assumed to arise from a *consensus sequence* χ. Each occurrence of the motif is a copy of χ, perhaps with a small number of substitutions. The consensus itself need never appear in a real biosequence – it is merely an abstraction that explains the pattern of conservation observed in a motif's occurrences.

One can form a consensus from any collection S of putative motif occurrences with common length ℓ. Define the consensus of a multiset B of bases to be that base which occurs most often[2] in B. Then the *consensus sequence of* S, denoted $\chi(S)$, is that ℓ-mer whose jth position contains the consensus of the jth bases of every $s_i \in S$, as illustrated in Figure 3.1. $\chi(S)$ is the consensus model that best describes S, since any other consensus sequence differs from a plurality of S's occurrences in some position.

A similarity score for S with respect to $\chi(S)$ can be defined in several ways. The usual score is the number of substitutions between each s_i and $\chi(S)$, summed over all $s_i \in S$. Alternatively, one could measure the conservation of a motif by the largest number of

[2]If there are two or more equally common bases, choose one of them arbitrarily.

substitutions d between its consensus and *any* of its occurrences. Thinking geometrically, the latter measure is the radius of the smallest sphere in Hamming space, centered on $\chi(S)$, that encloses all of S.

The Weight Matrix Model

A consensus sequence is a rather uninformative motif model because it says nothing about how strongly the consensus base in each position is conserved or about the distribution of any non-consensus bases. A more descriptive probabilistic model is the *weight matrix model (WMM)*, also called a *profile*. The WMM models a motif of length ℓ as a $4 \times \ell$ matrix W; the entry $W[i, j]$ gives the probability that an occurrence of the motif contains base i in its jth position[3]. Each column of W therefore sums to one. The bases in different positions of the motif appear independently at random according to the per-position distributions specified by W; hence, the probability that W produces a particular ℓ-mer s is given by

$$\begin{aligned} \Pr[s \mid W] &= \prod_{j=1}^{\ell} \Pr[s[j] \mid W[*, j]] \\ &= \prod_{j=1}^{\ell} W[s[j], j] \end{aligned}$$

Given a multiset S of motif occurrences, one can easily compute the WMM $W(S)$ that best describes S: set $W(S)[i, j]$ to be the frequency with which base i occurs among the jth positions of all occurrences in S. For example, Figure 3.2 shows the WMM corresponding to the motif occurrences of Figure 3.1. The matrix $W(S)$ is the best description of S in a *maximum likelihood* sense; that is, it is the WMM W that maximizes the likelihood

$$L[W \mid S] = \prod_{s \in S} \Pr[s \mid W].$$

The likelihood $L[W(S) \mid S]$ is a useful score by which to measure how well the motif S is conserved. If a particular position is completely unconserved, exhibiting equal base frequencies, it contributes a factor of $(0.25)^t$ (for motifs with t occurrences) to the likelihood score, whereas a position that always has a fixed base contributes a factor of 1. Positions

[3]By convention, bases are numbered in the order A, C, G, T.

ACTGCGC	A	**0.6**	0.0	**0.8**	0.0	0.2	**0.4**	0.0
AGAGATC	C	0.2	0.4	0.0	0.0	**0.6**	0.0	**0.6**
CGAGCGT	G	0.2	**0.6**	0.0	**1.0**	0.0	**0.4**	0.0
AGAGTAC	T	0.0	0.0	0.2	0.0	0.2	0.2	0.4
GCAGCAT								

Figure 3.2: A weight matrix model (WMM) inferred from the five motif occurrences in Figure 3.1. Entries corresponding to the consensus base at each position are identified in bold face. Unlike the consensus model, the WMM captures the frequencies of non-consensus bases and remains well-defined even when the consensus base is ambiguous, as in position six.

with intermediate degrees of conservation, e.g. those that always have a purine (A or G), contribute a factor between these two extremes. The reader seeking a more detailed discussion of the properties of the likelihood score should consult a good text on mathematical statistics, such as [48], or information theory, such as [28].

If the motif S occurs in i.i.d. random background sequences with a base distribution P, a better scoring function for S is the *likelihood ratio* $LR(S)$, defined as

$$LR(S) = \frac{L[W(S) \mid S]}{L[P \mid S]} \tag{3.1}$$

where

$$L[P \mid S] = \prod_{s \in S} \prod_{j=1}^{\ell} \Pr[s[j] \mid P].$$

The likelihood ratio, while not strictly a measure of conservation, is a principled way to account for the background base distribution when scoring a motif. The ratio adjusts for the background distribution by recognizing that, if base i appears frequently in the background, then a collection of strings with a high frequency of i's is more likely to occur purely by chance, and is therefore less significant as a putative motif, than one with few i's.

3.1.2 The Motif-Finding Problem

The motif-finding problem is a search in the same spirit as pairwise local alignment. We seek to ascertain whether a collection of genomic sequences contains substrings that constitute a high-scoring motif:

Problem 3.1 (Generic Motif Finding) Given t genomic sequences and a fixed length ℓ, find the highest-scoring ungapped motif S composed of t ℓ-mers, one from each input sequence.

Typically, t ranges between three and twenty sequences; considerations of both statistical significance and efficiency limit the lengths of these sequences to at most a few thousand bases each. Unlike pairwise local alignment, motif finding seeks the highest-scoring motif rather than setting a score threshold in advance. This difference arises because there is no one widely used theory of statistical significance for motifs comparable to Karlin-Altschul theory for pairwise alignment. Significance of motifs must in general be estimated in *ad hoc* fashion, as in Section 3.4.2, or empirically by simulation, though some general theoretical results are available for likelihood ratio scoring (see, e.g., [71, Section 8.2.3.3]).

The statement of Problem 3.1 contains several major assumptions: firstly, that the motif is ungapped; secondly, that it has fixed length ℓ; and thirdly, that it occurs exactly once per input sequence. In a probabilistic motif model, the latter two assumptions can be relaxed, albeit at some computational cost [11]. Even with all assumptions in place, however, the problem remains computationally challenging. Finding the best motif in a collection of input sequences, like many multiple alignment problems, is NP-hard [3].

A maximum-likelihood formulation using the WMM is perhaps the most popular, most broadly successful approach to motif finding in practice. Assuming this probabilistic model and scoring function, the motif-finding problem becomes:

Problem 3.2 (Maximum-Likelihood Motif Finding) Given t genomic sequences with bases distributed according to a background distribution P, find the motif S composed of t ℓ-mers, one from each input sequence, that maximizes the likelihood ratio $LR(S)$ defined by Equation (3.1).

Lawrence and Reilly [58] posed the motif finding problem in essentially this form[4]. With only minor changes to the scoring function, Problem 3.2 also encompasses close cousins of the maximum-likelihood formulation, such as the Bayesian approach of McCue et al. [62], that incorporate prior knowledge about the location and composition of the unknown motif.

3.2 Motif Finding Algorithms: Background and Limitations

3.2.1 Prior Work in Motif Finding

The motif-finding problem has spawned a plethora of different algorithmic solutions. These methods can be divided into two major approaches: *enumeration* and *local search*.

Enumerative algorithms exhaustively list and score all possible motifs in the input. The enumeration may be performed over collections of putative motif occurrences or over a countable set of models, such as the 4^ℓ possible ℓ-mer consensus sequences. Enumerative algorithms include methods by Blanchette et al. [16], Brāzma et al. [21], Galas et al. [36], Neuwald and Green [70], Sagot [82], Sinha and Tompa [89], Staden [95], Tompa [99], and van Helden et al. [100].

While enumerative algorithms are guaranteed to find the highest-scoring motif in the input, they require time exponential in the motif length ℓ and are considered impractical for motifs much above ten bases in length. One way to lower these methods' high cost is to enumerate *partial* motifs much smaller than the desired length, then try to assemble them into full-length motifs. This strategy is implemented by the TEIRESIAS algorithm of Rigoutsos and Floratos [79].

To avoid the exponential cost of enumeration for long motifs, motif finders resort to the heuristic approach of local search. Local search methods guess an initial model of the motif, then iteratively make small changes to the model that improve its score with respect to the input sequences. The model eventually converges to a *local maximum* whose score cannot be improved by further iteration but which is not guaranteed to be the globally highest-scoring motif. Local search methods increase their chances of finding the globally

[4]Technically, their problem statement asks for the motif induced by the maximum-likelihood WMM for the *entire* input and infers rather than fixes the distribution P. See Appendix C for details.

best motif by guessing many different initial models, iteratively improving each one, and finally reporting the highest-scoring motif resulting from any guess.

Local search is the technique of choice for motif finding in the popular maximum-likelihood formulation of Problem 3.2. Iterative improvement of the likelihood score can be performed numerically by expectation maximization (EM) or seminumerically by greedy search over models or by Gibbs sampling. Motif finders implementing one of these techniques include Bailey and Elkan's MEME [11], Hertz and Stormo's CONSENSUS [46], Lawrence et al.'s GibbsDNA [57], and the algorithms of Lawrence and Reilly [58] and of Rocke and Tompa [81].

3.2.2 Planted (ℓ,d)-Motifs and the Perils of Local Search

Despite the widespread use and notable success of motif finders using maximum-likelihood scoring and local search, there exists a class of highly significant motifs that such methods are largely unable to find. This limitation of existing motif finders was demonstrated by Pevzner and Sze [76], who studied a combinatorial formulation of motif finding, previously described by Sagot [82], in the consensus model. Their formulation, the *planted* (ℓ, d)*-motif problem*, is as follows:

Problem 3.3 (Planted (ℓ, d)-Motif Problem) Let χ be an (unknown) consensus sequence of length ℓ. Suppose that χ occurs once in each of t background sequences of common length n, but that each occurrence of χ is corrupted by exactly d substitutions in positions chosen independently at random. Given the t sequences, recover the motif occurrences and the consensus χ.

A particular parameterization of this problem, the so-called "challenge problem" of [76], plants a (15,4)-motif (that is, a motif of length $\ell = 15$ with exactly $d = 4$ substitutions per occurrence) in each of $t = 20$ background sequences of common length $n = 600$ composed of independent random bases chosen with equal frequencies. Such a well-conserved motif should be easy to identify because it has a very low probability ($< 10^{-14}$ by the estimate of Section 3.4.2 below) of occurring by chance in twenty random sequences of the specified length and composition. The values of n, t, and ℓ in the challenge problem are biologically

plausible, approximating those used to find, e.g., transcription factor binding sites in a collection of coregulated genes' promoter regions in yeast.

Pevzner and Sze showed that CONSENSUS, GibbsDNA, and MEME all perform poorly on the (15,4)-motif challenge problem. When presented with an instance of the challenge problem, these algorithms usually terminate at a local maximum of their score function corresponding to a randomly occurring pattern in the input, missing the planted motif despite its much higher score. The mere fact that planted motifs are generated by a combinatorial rather than probabilistic model does not excuse these failures, since any consensus-based motif can also be modeled as a WMM, and the problem parameters ensure that this WMM, if found, would be highly significant. The observed failures therefore suggest that **the performance of existing local search-based motif finders is strongly influenced by motif characteristics other than the degree of conservation**.

To illustrate why local search-based motif finders have difficulty solving planted (ℓ, d)-motif problems, consider the following example. Figure 3.3 shows two hexamer motifs, **A** and **B**, each of whose occurrences were produced by mutating the sequence `CCATAG` in exactly two positions. The numbers of mutations introduced in each motif are identical; however, given equal amounts of background sequence, motif **A** is in practice more likely than **B** to be found by local search.

The difference between motifs **A** and **B** is that the mutations in **B**, as specified in Problem 3.3, are distributed uniformly across its positions, while those in **A** are confined to its two center positions. This difference makes **B** harder to find for two reasons. First, despite having been constructed with the same number of mutations, and hence the same degree of conservation, as **A**, motif **B** scores worse than **A** under a variety of scoring functions. For example, **B** has only slightly more than half **A**'s information content[5] (4.4 versus 8.2 bits); algorithms that score motifs using statistical measures related to information content, including the likelihood ratio, will therefore have more trouble separating motif **B** from the background. Similarly, the best consensus model inferred from motif **A** differs from its occurrences by fewer than two substitutions on average, while every occurrence of **B** differs

[5]Information content for a WMM (in bits) is defined as 2ℓ minus the sum of the entropies of each position's base distribution. For a discussion of entropy, see [28].

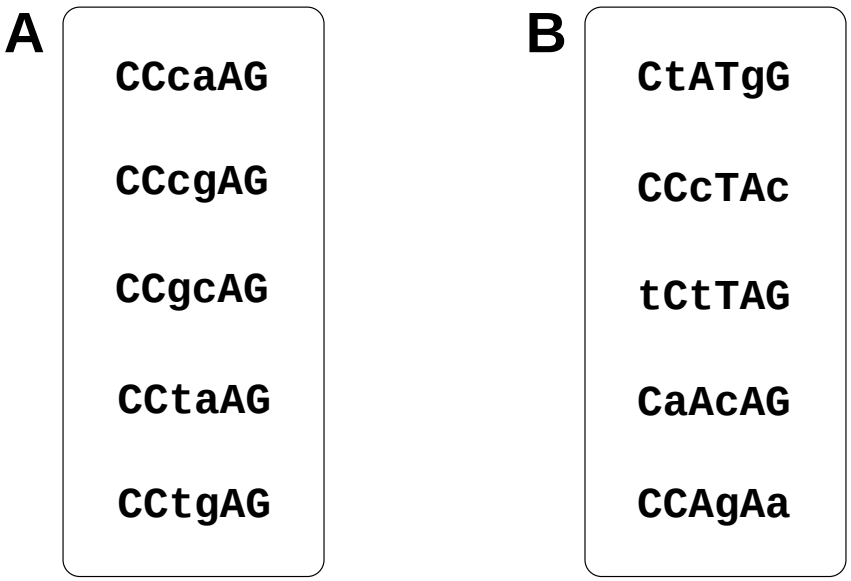

Figure 3.3: two hexamer motifs in which each occurrence differs by exactly two substitutions from the string `CCATAG`. Lower case letters indicate substitutions. Although the two motifs exhibit equal numbers of mutations, motif **B**'s substitutions are distributed uniformly throughout its occurrences (as in the challenge problem), while **A**'s substitutions are concentrated in its two center positions.

from its best consensus by exactly two bases.

A second, more insidious problem is the fact that the average Hamming distance between two occurrences of motif **B** is large – 3.6 substitutions versus only 1.6 for motif **A**. Local search methods typically start their search by guessing a single occurrence of the motif, then use iterative improvement to find additional occurrences by selecting ℓ-mers similar to the initial guess. Local search is likely to terminate at a local maximum different from the motif if the background contains substantial "noise," i.e. random ℓ-mers that are more similar to the initial guess than are other true occurrences of the motif. A larger average distance between motif occurrences increases the chance that the ℓ-mers most similar to an initially guessed occurrence are *not* other copies of the motif but random sequences from the background. For example, a DNA hexamer chosen uniformly at random has probability 0.038 of matching a fixed motif occurrence to within two substitutions, but it has probability 0.466 of matching it within four substitutions[6]. Hence, local search methods *using typical initialization strategies* encounter substantially more noise when finding motif **B** and so are more likely to fail.

Although motifs **A** and **B** represent an extreme dichotomy, they illustrate the ends of a continuum of practical interest. The more tightly a motif's mutations cluster in a small number of positions, the easier the motif is to find with existing algorithms; conversely, the more dispersed the mutations, the more subtle the motif. Biological motifs fall between the two extremes, but those close to the subtle end of the continuum are inaccessible to existing motif finders.

3.2.3 Recent Approaches to Planted (ℓ, d)-Motifs

Pevzner and Sze [76] developed two novel algorithms, WINNOWER and SP-STAR, to more reliably find subtle motifs, in particular to solve the (15,4)-motif challenge problem. Briefly, WINNOWER constructs a graph whose vertices correspond to all ℓ-mers present in the t input sequences, with an edge connecting two vertices if and only if the corresponding ℓ-mers

[6]Using more realistic parameters, a random 15-mer will match a fixed occurrence of a (15,4)-motif to within four substitutions with probability only 1.2×10^{-4}, but it has a greater than 5% chance of matching within eight substitutions.

differ in at most $2d$ positions and do not both come from the same sequence. WINNOWER then looks for a clique of size t in this graph. The second algorithm, SP-STAR, is a more conventional local search method that starts in turn from each individual ℓ-mer s in the input, chooses the closest match to s from every other input sequence to form an initial model, and uses iterative refinement to converge on a good motif. SP-STAR's motif scoring function is the *sum-of-pairs (SP)* score, defined for a motif S as

$$SP(S) = \sum_{i<j} D(s_i, s_j)$$

where $s_i, s_j \in S$ and D measures the Hamming distance between two strings.

WINNOWER and SP-STAR perform substantially better at finding planted (15,4)-motifs than previous algorithms [76]. However, they are less successful at solving more difficult planted motif problems. For example, the (14,4)-, (16,5)-, and (18,6)-motif problems, all with the same background length and composition as the challenge problem, prove intractable for the new algorithms as well as for existing local search techniques (see Table 3.1). The latter motifs, while not so well-conserved as a (15,4)-motif, are still highly significant; for example, an (18,6)-motif in the assumed background appears by chance with probability less than 10^{-7}. Hence, even the new algorithms leave a rather large gap between motifs that can be found computationally and the limits imposed by statistical significance. To close this gap, we turn once again to random projection.

3.3 The PROJECTION Algorithm

Figure 3.4 gives a new algorithm, PROJECTION, for the planted (ℓ, d)-motif finding problem. PROJECTION is designed to find planted (ℓ, d)-motifs that are intractable both to previous local search methods and to the WINNOWER and SP-STAR algorithms. Our algorithm combines random projection with a standard WMM-based local search motif finder, so it adapts easily to more practically useful maximum-likelihood motif finding as posed in Problem 3.2.

3.3.1 Intuition

To motivate the development of PROJECTION, observe that we can make local search more robust to a large average distance between motif occurrences by improving the way the

Figure 3.4: The PROJECTION algorithm. Subroutines referred to in the algorithm are: GEN-RANDOM-PROJ-NOREP(ℓ, k), which generates a random projection by picking k values uniformly at random *without* replacement between 1 and ℓ; THRESHOLD(h), which computes a bucket size threshold for bucket S_h using the Poisson approximation described in Section 3.3.2; REFINE(S), which performs iterative motif refinement (by EM in our implementation) initialized with the guess S; and SCORE(S), which computes the score of a motif S. The parameters m, k, and σ are chosen as described in Section 3.3.2. EM refinement and scoring are described in Section 3.3.3.

PROJECTION(C, ℓ, d, m, k, σ)

C: a collection of genomic sequences

ℓ: length of motif

d: max. substitutions per motif occurrence

m: number of random projections to perform

k: number of positions in each projection

σ: fixed bucket size threshold

S_h: bucket labeled with projection value h

M: set of all candidate motifs

$M \leftarrow \emptyset$

repeat the following m times

$f \leftarrow$ GEN-RANDOM-PROJ-NOREP(ℓ, k)

/* compute bucket of each ℓ-mer in C */

foreach sequence $c \in C$

for $1 \leq j \leq |c| - \ell + 1$ **do**

$s \leftarrow c\,[j \ldots j + \ell - 1]$

$S_{f(s)} \leftarrow S_{f(s)} \cup \{s\}$

end

end

/* refine all sufficiently large buckets */

foreach bucket S_h

if $|S_h| \geq$ MAX$(\sigma,$ THRESHOLD$(h))$

$S_h^* \leftarrow$ REFINE(S_h)

$M \leftarrow M \cup \{S_h^*\}$

end

end

end

return motif $S^* \in M$ s.t. SCORE(S^*) is maximal

search is initialized. Consider again the difficult motif **B** of Figure 3.3, and suppose we initially guess not one but $\sigma > 1$ occurrences of this motif. For example, suppose that $\sigma = 3$, and that we somehow guess the first, second, and fourth occurrences of **B** in the figure. The consensus of these three strings is `CAATAG` (with the second base chosen arbitrarily because of a tie), which differs from the true motif's consensus in only one position and from its occurrences by an average of 2.6 substitutions – substantially less than the average distance between one occurrence and another. Similarly, the WMM formed from these three strings describes the motif much more accurately than does any one of its occurrences. Starting with multiple motif occurrences permits an initial guess that more closely resembles the true motif and so decreases the opportunity for noise to confound subsequent local search.

The main problem with a more robust initialization strategy is computational cost. We do not know where the motif appears in the input, so to find σ occurrences we must naively guess every collection of σ ℓ-mers in the input as a starting point for local search, requiring $\binom{t}{\sigma}(n-\ell+1)^{\sigma}$ searches overall. Even for $\sigma = 2$, exhaustive enumeration and testing of all such guesses seems prohibitively expensive. PROJECTION takes a different approach: it *randomly samples* collections of at least σ ℓ-mers in a way that is biased toward picking sets of motif occurrences. Specifically, PROJECTION prefers multisets of ℓ-mers that are likely to be similar to a common consensus.

3.3.2 Better Initialization through Random Projection

In accordance with the intuition of the previous section, PROJECTION attempts to correctly guess at least σ occurrences of an (unknown) planted motif. To this end, the algorithm performs m independent trials, each of which may generate multiple guesses. In each trial, it chooses a random projection f and hashes each ℓ-mer s in the input to a *bucket* $S_{f(s)}$. We call the projection value $f(s)$ the *label* of $S_{f(s)}$. Any bucket receiving sufficiently many ℓ-mers is explored as a potential motif, using a local search-based refinement procedure described in the next section.

As in Section 2.3.1, the projections f are constructed by choosing k positions uniformly at random from the set $\{1 \ldots \ell\}$, for a value of k to be determined later. Rather than try to

deal with the vagaries of a variable projection size, which are practically more difficult to work around in motif finding than in pairwise alignment, PROJECTION selects the positions of each projection *without* replacement.

Let χ be the unknown motif's consensus, and define the *planted bucket* to be that bucket labeled with value $f(\chi)$. The fundamental idea underlying our use of random projection is that, if k is small enough, there is a good chance that at least σ occurrences of χ will hash together into the planted bucket. At the same time, if k is not too small, it is unlikely that many random ℓ-mers from the background sequence will hash to the planted bucket, because each such ℓ-mer must agree with χ in all k sampled positions. Thus, the ℓ-mers in the planted bucket will likely be highly enriched for the planted motif.

The planted bucket depends on the unknown motif, so PROJECTION cannot identify it *a priori*. However, by setting the parameters m and k appropriately, one can arrange for the planted bucket to receive at least σ motif occurrences in at least one iteration of the algorithm with high probability. Hence, PROJECTION refines only those buckets with at least σ ℓ-mers, confident that one such bucket will be the planted one[7].

To fully specify the PROJECTION algorithm, we must describe how to choose both the projection size k and the number of trials m as a function of the algorithm parameters ℓ, d, n, t, and σ and a bound q on the probability that the algorithm successfully hashes at least σ motif occurrences to the planted bucket. We must also describe how to choose σ, which while notionally an input to the algorithm is not a "natural" parameter for motif finding.

Choice of Parameters k and m

As with LSH-ALL-PAIRS, many different combinations of m and k will achieve the goal of producing a large planted bucket with high probability. However, very small values of k cause many background ℓ-mers hash to the planted bucket; in the degenerate case $k = 0$, the entire input would end up in one bucket that would be useless for motif finding. We therefore choose a projection size k large enough to minimize contamination of the planted bucket by random background sequences. Suppose we require at most E background ℓ-mers

[7]On occasion, PROJECTION actually succeeds in recovering the correct motif by refining some bucket other than the planted bucket, which is an added bonus.

per bucket on average. PROJECTION hashes $t(n-\ell+1)$ ℓ-mers into 4^k buckets, so if we set

$$k \geq \log_4 \left(\frac{t(n-\ell+1)}{E} \right),$$

then the expected number of background ℓ-mers per bucket is at most E. We normally fix $E < 1$ so that the planted bucket is expected to contain less than one ℓ-mer from the background sequence.

The number of trials m must be set large enough so that with probability at least q, some trial produces a planted bucket containing at least σ motif occurrences. The probability q is computed over both the random choices of projections and the random distribution of mutations in the problem instance. We set $q = 0.95$, which is high enough that PROJECTION often produces planted buckets with at least σ motif occurrences in several trials, providing some robustness against unsuccessful refinements.

Because PROJECTION chooses its projections uniformly at random, each motif occurrence in the planted model hashes to the planted bucket with probability $\hat{p}(\ell, d, k)$, defined by

$$\hat{p}(\ell, d, k) = \frac{\binom{\ell - d}{k}}{\binom{\ell}{k}}.$$

In particular, those planted occurrences for which the d mutated positions are disjoint from the k hash positions will hash to the planted bucket. Let $\hat{t}$ be an estimate of the number of input sequences containing a planted motif occurrence ($\hat{t} = t$ for our synthetic challenge problems and promoter examples). Then the probability that fewer than σ planted occurrences hash to the planted bucket in a given trial is $B_{\hat{t},\hat{p}(\ell,d,k)}(\sigma)$, where $B_{t,p}(x)$ is the probability that there are fewer than x successes in t independent Bernoulli trials, each trial having probability p of success. We may assume that the trials for different motif occurrences are independent because the problem formulation states that mutations appear independently at random in each occurrence.

If PROJECTION is run for m trials, the probability that σ or more motif occurrences hash to the planted bucket in at least one trial is

$$1 - \left[B_{\hat{t},\hat{p}(\ell,d,k)}(\sigma) \right]^m \geq q.$$

In order to satisfy this inequality, choose

$$m = \left\lceil \frac{\log(1-q)}{\log(B_{\hat{t},\hat{p}(\ell,d,k)}(\sigma))} \right\rceil . \tag{3.2}$$

This bound on m is similar in character to that derived in Inequality 2.3 of the previous chapter.

Choice of Threshold σ

The choice of the bucket size threshold σ is determined by a combination of sensitivity and efficiency concerns. The need for sensitivity dictates a threshold large enough that a bucket with at least σ true motif occurrences is likely to produce the planted motif after refinement. Unfortunately, we know of no formal properties of local search refinement from which to determine a lower bound on σ, though the intuition of Section 3.3.1 and the fact that a bucket may receive one or more background ℓ-mers both argue strongly for $\sigma > 1$. Empirically, we have found that in experiments on problems containing 4–20 motif occurrences in background sequences of 600–1000 bases, setting $\sigma = 3$ or 4 usually works well. This empirical value of σ is the one used to determine m in Equation 3.2. Using these criteria for m, k, and σ, finding even subtle motifs of the sizes considered here requires at most thousands of trials, and usually many fewer, to produce a bucket containing enough motif occurrences for effective refinement.

An alternative lower bound on the bucket size threshold derives from a practical consideration of efficiency: we wish to discard buckets composed mostly or entirely of background ℓ-mers, which are unlikely to be useful as starting points for refinement. We assume for simplicity that in a collection of random background sequences, the number of ℓ-mers that project to the bucket with label h is approximately Poisson-distributed with a mean μ_h given by

$$\mu_h = t(n - \ell + 1) \Pr_s [f(s) = h]$$

where the latter probability is computed over the distribution of ℓ-mers s in the background sequences. The Poisson assumption derives from the fact that we are counting occurrences of a short fixed pattern, namely the bases of h appearing in the positions read by f, in

the background. In the case of nonuniform background base frequencies, the probability $\Pr[f(s) = h]$ is different for each bucket S_h, producing larger thresholds for buckets likely to contain many background ℓ-mers purely by chance.

PROJECTION sets the size threshold for each bucket S_h to the larger of the empirical σ determined by sensitivity and the 90th percentile value of the bucket's size distribution, as determined by its label h. The per-bucket size computation is represented in Figure 3.4 by the function THRESHOLD(h). Excluding buckets with more than σ ℓ-mers could in theory cause the algorithm to miss the planted bucket when it appears; in this work, however, the empirical value of σ is larger than the Poisson-derived size threshold for most buckets, so the practical effect on sensitivity is small.

3.3.3 Motif Refinement and Scoring

PROJECTION refines each sufficiently large bucket in hopes of recovering the planted motif. If the bucket being refined is the planted bucket, its ℓ-mers already match the motif's consensus in at least k positions. These positions plus the information in the remaining $\ell - k$ positions of the bucket's ℓ-mers provide a strong signal, starting from which a few iterations of refinement should lead to the correct motif.

The primary function of PROJECTION is to pick good starting guesses for refinement; it should in principle be agnostic as to which local search technique is used to refine those guesses into motifs. Our implementation refines candidate motifs using expectation maximization as formulated by Lawrence and Reilly [58]. Their formulation makes several important simplifying assumptions:

1. The motif is well-modeled by an ungapped WMM of fixed length ℓ.

2. The motif occurs exactly once per input sequence.

3. The background sequences are composed of independent, identically distributed bases.

Appendix C reviews this formulation and how we implement it.

Although Lawrence and Reilly's motif model is greatly simplified compared to motifs in real biosequences, it is sensitive enough to identify meaningful motifs in practice. Bailey

and Elkan [11] describe a more elaborate mixture model EM that permits any number of motif occurrences per sequence, but fitting its parameters from sequence data requires significantly more computation than the simpler Lawrence and Reilly model.

To form an initial guess for EM from a bucket S_h, we compute its maximum-likelihood WMM $W(S_h)$. As a practical matter, it is wise to avoid zero entries in this initial matrix because they are as likely due to the small number of sampled motif occurrences as to the true motif composition. To avoid avoid zeroes in $W(S_h)$, we add to each entry $W(S_h)[i,j]$ a Laplace correction of $P[i]$, the probability of base i in the background sequence.

Because EM converges only linearly, running it to convergence for every starting guess would be computationally prohibitive. Fortunately, just a few iterations of EM (five in our implementation) can significantly improve a well-chosen starting model to the point where it identifies the planted motif. Let W_h^* be the candidate motif model refined from $W(S_h)$, and let P be the background sequence distribution. We form a refined guess S_h^* at the planted motif by selecting from each input sequence the ℓ-mer s with the largest likelihood ratio $\Pr[s \mid W_h^*] / \Pr[s \mid P]$. The multiset S_h^* represents the motif in the input that is most consistent with the model W_h^*.

PROJECTION generates a refined guess for every sufficiently large bucket from every trial, but we wish to pick a single best motif to report to the user. For the biological examples of Section 3.4.5, we score each refined motif S_h^* by its likelihood ratio score $LR(S_h^*)$. We report the motif S^* that maximizes this score over all buckets from all m trials.

To maximize the number of motif occurrences recovered in the synthetic challenge problems of Section 3.4.1, we perform a further combinatorial refinement of each S_h^*. This further refinement process is similar to SP-STAR [76] but uses a different, combinatorial score function. Define the score $r(S_h^*, d)$ to be the number of ℓ-mers in S_h^* within radius d of its consensus $\chi(S_h^*)$, and let S_h' contain the ℓ-mer from each input sequence that is closest in Hamming distance to $\chi(S_h^*)$. If $r(S_h', d) > r(S_h^*, d)$, replace S_h^* by S_h' and repeat. This refinement usually converges within a few iterations. Again, we report the motif S^* for which $r(S^*, d)$ is maximum over all buckets from all m trials.

3.4 Experimental Results

In this section, we investigate PROJECTION's ability to find both synthetic and biological motifs. The synthetic motifs, described in Section 3.4.1, are (naturally) the (15,4)-motif challenge and more difficult planted motif problems for which the algorithm was designed. Sections 3.4.3 and 3.4.4 respectively determine the performance impact of introducing background sequences with nonuniform distributions and longer lengths than in the original challenge problem, while Section 3.4.2 estimates the statistical limits of (ℓ, d)-motif finding to quantify the remaining gap between problems that PROJECTION can solve and problems that cannot be solved at all.

In Sections 3.4.5 and 3.4.6, we complement these synthetic results by addressing realistic biological motif finding problems, first identifying transcription factor binding sites in the promoter regions of eukaryotic genes, then tackling the problem of finding ribosome binding sites in prokaryotes. The promoter data sets contain just a few motif occurrences in an equal number of sequences, while the ribosome binding site data sets consist of thousands of sequences, only a fraction of which contain the motif. We validate the motifs found by PROJECTION by comparing them to published sites from the literature and, for ribosome binding sites, to their complementary 16S rRNA sequences.

3.4.1 Challenge Problems on Synthetic Data

We first tested PROJECTION on synthetic problem instances generated according to the planted (ℓ, d)-motif model of Pevzner and Sze [76]. Such problems have been shown to be intractable to most existing motif finders, even for $\ell = 15$ and $d = 4$, so they are natural test cases for our algorithm. We generate problem instances as follows: first, a motif consensus χ of length ℓ is chosen by picking ℓ bases at random. Second, $t = 20$ occurrences of the motif are created by randomly choosing d positions per occurrence (without replacement) and mutating the base at each chosen position to a different, randomly chosen base. Third, we construct t background sequences of length $n = 600$ using $n \cdot t$ bases chosen at random. Finally, we assign each motif occurrence to a random position in a background sequence, one occurrence per sequence. All random choices are made uniformly and independently

with equal base frequencies. This generation procedure corresponds to the "FM" model used in the challenge problem described by Pevzner and Sze.

We report the performance of PROJECTION using the performance coefficient of [76], defined as follows. Let K denote the set of $t \cdot \ell$ base positions in the t occurrences of a planted motif, and let $\hat{K}$ denote the corresponding set of base positions in the t occurrences predicted by an algorithm. Then the algorithm's *performance coefficient* on the motif is defined to be $|K \cap \hat{K}|/|K \cup \hat{K}|$. When all occurrences of the motif are found correctly, the performance coefficient achieves its maximum value of one.

Table 3.1 compares the performance of PROJECTION with that of previous motif discovery algorithms on sets of twenty random problem instances, each generated as described above. All experiments used projection size $k = 7$ and bucket size threshold $\sigma = 4$, which combined with the problem parameters requires numbers of trials m as shown in the table. The values of m are determined by Equation (3.2). For each set of problems, we give the average performance coefficient for PROJECTION as well as the number of problem instances (out of twenty) for which it correctly recovered the planted motif's consensus. For comparison, we provide corresponding average performance coefficients for three other algorithms: GibbsDNA, WINNOWER ($k = 2$), and SP-STAR. The data for previous algorithms was collected by Pevzner and Sze and summarized in [76, Figures 1 and 2].

In every line of Table 3.1, the average performance of PROJECTION is at least as great as that of any of the previous algorithms. PROJECTION almost always correctly solved planted (11,2)-, (13,3)-, (15,4)-, (17,5)-, and (19,6)-motif problems, with performance coefficients less than one appearing only because our algorithm, like any motif finder, sometimes picked as a motif occurrence a background ℓ-mer that was at least as similar to the correct consensus as was the true occurrence. The (11,2)-, (13,3)-, and (15,4)-motif problems were also accessible to WINNOWER and to SP-STAR.

PROJECTION's improved performance is more striking on the more difficult planted (14,4)-, (16,5)-, (17,5)-, (18,6)-, and (19,6)-motif problems. Our algorithm's performance on these problems substantially exceeds that of previous algorithms, including those of Pevzner and Sze, which generally fail to find the planted motifs. Finding each synthetic motif in the most difficult (18,6) problem required about one hour on a 667 MHz Alpha workstation;

Table 3.1: performance of PROJECTION and previous motif finding algorithms on synthetic (ℓ, d)-motif problems. Each problem instance consists of $t = 20$ sequences each of length $n = 600$. Average performance coefficients of GibbsDNA, WINNOWER ($k = 2$), and SP-STAR are from Pevzner and Sze (personal communication), who averaged over eight random instances. For PROJECTION, averages were taken over twenty random instances, with projection size $k = 7$ and threshold $\sigma = 4$.

ℓ	d	GibbsDNA	WINNOWER	SP-STAR	PROJECTION	Correct	m
10	2	0.20	0.78	0.56	0.82	20	72
11	2	0.68	0.90	0.84	0.91	20	16
12	3	0.03	0.75	0.33	0.81	20	259
13	3	0.60	0.92	0.92	0.92	20	62
14	4	0.02	0.02	0.20	0.77	19	647
15	4	0.19	0.92	0.73	0.93	20	172
16	5	0.02	0.03	0.04	0.70	16	1292
17	5	0.28	0.03	0.69	0.93	19	378
18	6	0.03	0.03	0.03	0.74	16	2217
19	6	0.05	0.03	0.40	0.96	20	711

easier problems like (15,4) were typically solved in only a few minutes. The vast majority of PROJECTION's running time was spent performing local search, with only a small fraction attributable to its random projection-based initialization phase.

3.4.2 Limitations on Solvable (ℓ, d)-Motif Problems

Although PROJECTION performs well on the planted motifs of Table 3.1, it generally fails to find motifs with slightly different parameters, such as (9,2)-, (11,3)-, (13,4)-, (15,5)-, or (17,6)-motifs (again for $t = 20$ and $n = 600$). We naturally investigated why our algorithm tends to fail on problems that seem quite similar to the original challenge.

A probabilistic analysis shows that problems involving planted motifs with the above parameters are quantitatively different from the problems in Table 3.1. For example, twenty random sequences of length 600, with no planted motif, are expected to contain more than one spurious (9,2)-motif by chance, whereas the expected number of (10,2)-motifs that they contain is approximately 6.1×10^{-8}. We derive these estimates as follows. Let

$$p_d = \sum_{i=0}^{d} \binom{l}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{l-i}$$

be the probability that a fixed ℓ-mer occurs with up to d substitutions at a given position of a random sequence. Then the expected number of length-ℓ motifs that occur with up to d substitutions at least once in each of t random length-n sequences is approximately

$$E(\ell, d) = 4^l \left(1 - (1 - p_d)^{n-l+1}\right)^t .$$

This expectation is only an estimate because overlapping occurrences of a given consensus string χ do not occur independently in the background.

Table 3.2 lists relevant values of $E(\ell, d)$ and $E(\ell + 1, d)$ for comparison. In each line of the table, the expected number of spurious (ℓ, d)-motifs is around 1–5, whereas the expected number of spurious $(\ell + 1, d)$-motifs is negligible. We therefore expect that on the specified (ℓ, d)-problems, PROJECTION, or for that matter any other algorithm, is likely to report a spurious motif as good as the planted motif, even if it usually succeeds on the corresponding $(\ell + 1, d)$-motif problems.

Table 3.2: statistics of spurious motifs in synthetic (ℓ, d) problems. Parameters used were $k = 7$, $\sigma = 4$, m as shown. Column headings: "a.p.c" = average performance coefficient over twenty problem instances; "Correct" = instances yielding correct motif consensus; "Spurious" = instances yielding equally good but spurious consensus; "19/20" = instances yielding a consensus with nineteen occurrences.

ℓ	d	$E(\ell, d)$	$E(\ell+1, d)$	a.p.c.	Correct	Spurious	19/20	m
9	2	1.6	6.1×10^{-8}	0.28	11	5	4	1483
11	3	4.7	3.2×10^{-7}	0.026	1	13	6	2443
13	4	5.2	4.2×10^{-7}	0.062	2	15	3	4178
15	5	2.8	2.3×10^{-7}	0.018	0	7	13	6495
17	6	0.88	7.1×10^{-8}	0.022	0	8	12	9272

Although the expectations of Table 3.2 are only estimates, we know from an exhaustive enumeration of 9-mers and an exact calculation of their probabilities in twenty random 600-mers that the expected number of spurious (9,2)-motifs is 1.621. (The probability calculation was done using an algorithm described in [99, Section 3.1].) Thus, the estimates may not be too inaccurate in practice.

To further corroborate our analysis, we ran PROJECTION on sets of twenty random instances of the planted (ℓ, d)-motif problems of Table 3.2 generated as described in the previous section. The algorithm's performance on these sets is also reported in Table 3.2, including the average performance coefficient, the number of problem instances in which the correct consensus was found, and the number of instances where we instead found a spurious (ℓ, d)-motif appearing in all twenty input sequences. Where PROJECTION failed to find either the correct or an equally good spurious motif, it found a motif (again not the planted one) occurring in nineteen of the twenty input sequences. These experiments provide further evidence that the (ℓ, d)-motif problems that PROJECTION consistently fails to solve are often intractable to any algorithm, simply because there is no way to distinguish the planted motif from an equally high-scoring but spurious motif.

Finally, we note that the distinction between solvable and unsolvable motif finding problems assumes that, as in our experiments on synthetic data, a motif finder gets only one

chance to find the planted motif. If, as in the biological examples of Section 3.4.5 below, a motif finder may return multiple high-scoring motifs, it may be still able to find the true motif (though not to distinguish it from equally high-scoring spurious motifs in the background). Future work should therefore explore the challenging parameterizations of Table 3.2 to assess whether returning both real and spurious motifs with equal scores is computationally feasible. However, if such motifs arise often, they could better be found by incorporating into the algorithm additional information, such as the expected positions of their occurrences, that better distinguishes them from the background sequence.

3.4.3 Performance vs. Background Base Distribution

Real biosequences frequently have base compositions different than the equal base frequencies used in the experiments of Section 3.4.1. We therefore tested the performance of PROJECTION on synthetic motif-finding problems with background sequences of varying composition. These problems were constructed and solved identically to those of Section 3.4.1, except that we chose a background GC fraction θ_{GC}, then generated the background sequence from a distribution with $\Pr[G] = \Pr[C] = \theta_{GC}/2$ and $\Pr[A] = \Pr[T] = (1 - \theta_{GC})/2$. We continued to generate planted motifs from a distribution with equal base frequencies.

Figure 3.5 shows the performance of PROJECTION on both (15,4)- and more challenging (14,4)-motifs at different background compositions. The algorithm's performance, as measured by the average performance coefficient, was essentially unchanged on (15,4)-motifs down to 35% GC and on (14,4)-motifs down to 40% GC. Below these thresholds, recovery of the motif drops off precipitously, to essentially zero at 35% GC for (14,4)-motifs and 20% GC for (15,4)-motifs. This performance drop occurs because a biased background is more likely to produce problem instances that, like those of Section 3.4.2, by chance contain spurious motifs at least as good as the planted motif. Indeed, in every example where the algorithm failed to find the planted motif, it found a collection of twenty ℓ-mers that were at least as well conserved. In the absence of spurious motifs, PROJECTION's ability to find the planted motif is robust to moderate variations in GC content.

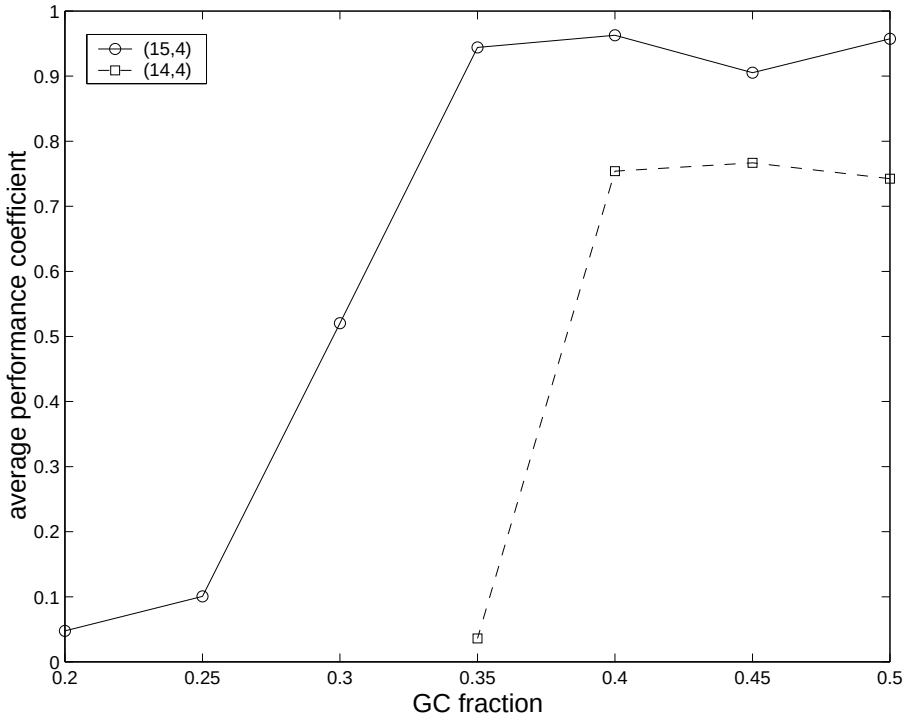

Figure 3.5: performance of PROJECTION on synthetic motif-finding problems with unequal background base frequencies. Problems were generated as in Section 3.4.1, except that the background GC fraction was set to values different from 0.5 as shown. Problem instances contained either (15,4)-motifs (solid line) or (14,4)-motifs (dashed line) generated with equal base frequencies.

3.4.4 Performance vs. Background Sequence Length

We tested PROJECTION's ability to handle increasingly noisy problems by finding planted motifs in increasingly large amounts of background sequence. Longer backgrounds contain more random ℓ-mers similar to real motif occurrences, as well as more collections of ℓ-mers scoring almost as highly as the true motif. Both phenomena increase the chance that local search will terminate at a suboptimal local maximum instead of finding the true motif.

Figure 3.6 shows the performance of PROJECTION on (15,4)- and (14,4)-motifs for background lengths ranging from $n = 600$ to $n = 2000$. Other than the increased length n, problem instances were generated identically to those of Section 3.4.1. All experiments used the parameters given in Table 3.1 for (15,4)- and (14,4)-motifs. In these experiments, we retained the projection size $k = 7$ at all lengths, even though for backgrounds longer than 800 bases, setting $k = 7$ causes the average bucket size to exceed one ℓ-mer. Setting $k = 8$ would have kept the average bucket size below one but would have required an order of magnitude more iterations ($m = 1987$ for (15,4) and $m = 14860$ for (14,4)) to maintain a 95% probability of producing a planted bucket with at least $\sigma = 4$ occurrences of the motif.

PROJECTION's performance on (15,4)-motifs degrades gracefully with increasing length, from a performance coefficient of nearly 1.0 for $n = 600$ to 0.6 at $n = 2000$. As predicted, some of the decay can be attributed to failures to find the planted motif at all (one failure in twenty problem instances at $n = 1600$, 4/20 at $n = 2000$), while the rest is attributable to admixture of background ℓ-mers into otherwise correct motifs. The observed failures, unlike those of the previous section, are *not* attributable to spurious motifs as good as the planted motif; rather, they are consistent with algorithm failures caused by a proliferation of suboptimal local maxima. For example, at $n = 2000$ the four failed problem instances all yielded spurious (15,4)-motifs with sixteen or seventeen (rather than twenty) occurrences, which by the estimates of Section 3.4.2 are expected to occur frequently by chance at this background size.

Finding (14,4)-motifs proved much harder at increased sequence lengths. Failures to find the planted motif's consensus increased dramatically between $n = 700$ and $n = 800$, from two to ten failures in twenty trials. The failures at these lengths were again caused

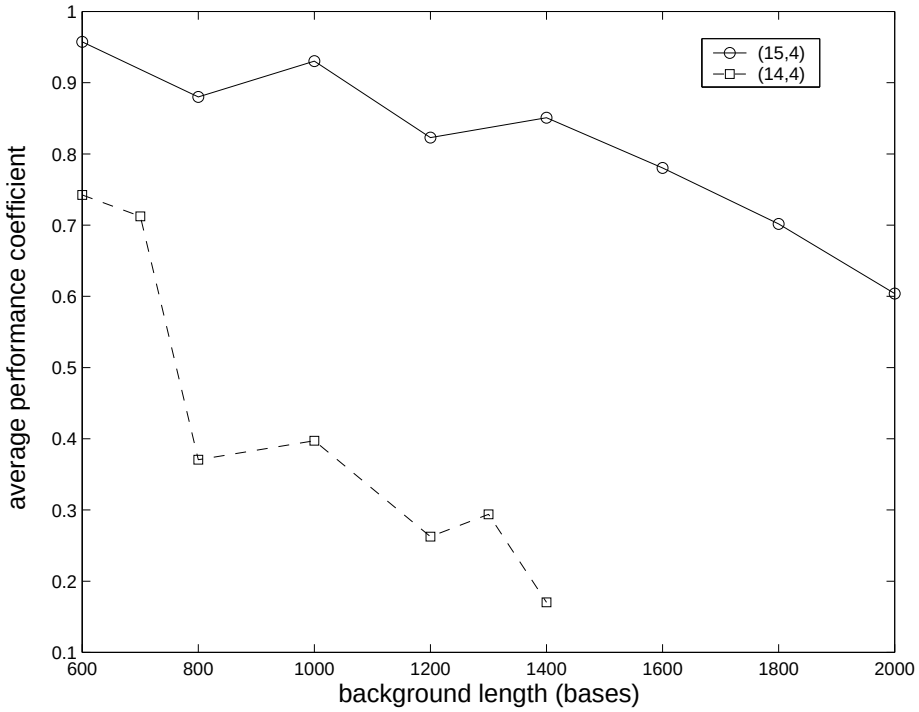

Figure 3.6: performance of PROJECTION on synthetic motif-finding problems with increasing background sequence lengths. Problems were generated as in Section 3.4.1, except that the background sequence length was scaled from $n = 600$ to $n = 2000$ as shown. The number of sequences was maintained at $t = 20$. Problem instances contained either (15,4)-motifs (solid line) or (14,4)-motifs (dashed line).

by local search terminating at a suboptimal local maximum, usually a (14,4)-motif with sixteen or seventeen occurrences. The nonmonotonicity of the performance curves suggests that estimates from only twenty problem instances have substantial variance, which may also explain in part the large magnitude of the performance drop observed between 700 and 800 bases. We did not extend the (14,4) curve past $n = 1400$ because at this length, spurious (14,4)-motifs with a full twenty occurrences are already frequent enough that they cause most of the observed failures.

Increased length testing is useful not only to determine the absolute performance limits of PROJECTION but also to compare its performance to previous algorithms, including the specialized methods of Pevzner and Sze. Previous motif finders usually fail to find (15,4)-motifs when $n = 1000$, exhibiting a performance coefficient of 0.23 or less [76, Table 1]. Only WINNOWER, a non-local-search-based motif finder, maintains an average performance of at least 0.8 at this background size, and that only with parameter $k = 3$. PROJECTION's comparatively high performance, even for $n = 2000$, again demonstrates the power of augmenting ordinary local search with intelligent initialization.

3.4.5 Transcription Factor Binding Sites

To test PROJECTION on realistic biological data, we used it to find known transcriptional regulatory elements upstream of eukaryotic genes. We examined orthologous sequences from a variety of organisms taken from regions upstream of four types of gene: preproinsulin, dihydrofolate reductase (DHFR), metallothioneins, and c-*fos*[8]. These sequences are known to contain binding sites for specific transcription factors. We also tested a collection of promoter regions[9] from the yeast *S. cerevisiae* that are known to contain a common cell-cycle-dependent promoter, the ECB element [63].

The motifs in these data sets are much better conserved than those in our synthetic problem instances, with little variation and a structure more like the simple motif **A** of Figure 3.3 than like the more subtle **B**. In general, we have been unable to locate published

[8]Sequences were kindly provided by M. Blanchette; see [15] for a list of organisms used and an alternative approach to finding motifs in these sequences.

[9]*Genes used*: SWI4, CLN3, CDC6, CDC46, and CDC47.

examples of biological motifs as subtle as those of Section 3.4.1, but the dearth of such examples need not imply that subtle motifs do not exist biologically. Motifs like those in the planted (ℓ, d) model are inaccessible to existing computational search techniques, and a high degree of conservation is necessary to detect a motif at all given only four or five occurrences. For example, a (15,4)-motif occurring only five times with uniformly distributed mutations would be statistically meaningless in a background like those of our promoter data sets. Many published motifs were likely inferred using few enough sequences that a subtle (ℓ, d)-motif, if present, would have appeared insignificant and so would not have been reported.

In all experiments, we set $\ell = 20$ and $d = 2$, which worked well despite the fact that the actual motifs varied considerably in length. We chose a uniform projection size $k = 7$ for all experiments, which implies less than one expected background ℓ-mer per bucket even for our largest problem instance. Because the motif occurs only 4–5 times in the input, we set a smaller than usual bucket size threshold $\sigma = 3$ so as not to demand that all occurrences end up in one bucket. Given the high expected amount of conservation and the low size threshold, the numbers of iterations m computed from Equation (3.2) proved quite small, requiring only a few seconds' running time. Motifs were scored by likelihood ratio score as described in Section 3.3.3.

Table 3.3 gives for each experiment the consensus of the highest-scoring motif found by PROJECTION, along with a published motif that closely matches a substring of that consensus. The locations of motif occurrences were not known *a priori*, so we do not give performance coefficients. Analysis of the preproinsulin promoter region yielded a motif known from the TRANSFAC database [105], while the other four experiments all produced motifs corresponding to experimentally verified transcription factor binding sites (see Table 3.3).

In cases where a data set contained several distinct known binding sites, we attempted to find additional motifs beyond that of highest score. To find multiple motifs, we simply masked the motif of highest score, replacing each base in its occurrences with X, and reran the motif finder on the masked sequences. This procedure proved effective in finding additional documented motifs. In the preproinsulin data, the second motif found was the well-known CT-II promoter element [17], while the second and third motifs from the

Table 3.3: performance of PROJECTION on eukaryotic promoter sequences. All motifs were found using parameters $\ell = 20$, $d = 2$, $k = 7$, and $\sigma = 3$. Underlined portions of motifs indicate matches to known sequence features. "Start points" counts total number of buckets used as start points for EM refinement.

Sequence	Input Size (seqs / bases)	m	Start Points	Best (20,2) Motif from PROJECTION	Published Reference Motif
preproinsulin	4 / 7689	15	5147	GAAATTGCAG<u>CCTCAGCCCC</u>	CCTCAGCCCC [a]
DHFR	4 / 800	15	158	TGCA<u>ATTTCGCGCCA</u>AACTT	ATTTCnnGCCA [b]
metallothionein	4 / 6823	15	3188	CTC<u>TGCGCCCGG</u>ACCGGTTC	TGCRCYCGG [c]
c-*fos*	4 / 3695	8	1043	<u>ATATTAGGACATCT</u>GCGTCA	...CCATATTAGGACATCT [d]
yeast ECB	5 / 5000	8	1204	GGAAA<u>TTTCCCGTTTAGGAA</u>	TTtCCcnntnaGGAAA [e]

[a] TRANSFAC signal [105]

[b] non-TATA transcription start signal [64]

[c] MREa promoter [6]

[d] 3′ end of c-*fos* serum response element [68]

[e] yeast early cell cycle box [63]

metallothionein data were respectively the MREd and GC-box elements of [6].

In addition to the promoter sequences listed in Table 3.3, we ran PROJECTION on a set of twenty 1000-base *C. elegans* promoter regions containing occurrences of the "X box" motif `RYYNYYATRRNRAC`, the target site for the DAF-19 transcription factor [98]. The genes from which these sequences are taken were chosen by P. Swoboda (personal communication) because their expression is likely regulated by DAF-19. Some genes exhibit empirical evidence of such regulation, while the remainder were chosen because they exhibit an occurrence of the X box motif between 50 and 300 bases upstream of the translation start site.

The X box looks somewhat more like the subtle motifs for which PROJECTION was designed. Only four of the fourteen positions in the motif are perfectly conserved across all twenty occurrences; of the remaining positions, one is poorly conserved, while the rest exhibit a strong preference for either purine or pyrimidine, with one base appearing in 13–19 occurrences. PROJECTION easily found nineteen of twenty known motif occurrences (perf. coeff. = 0.90) using parameters $\ell = 14$, $d = 2$, $k = 8$, $m = 6$, and $\sigma = 4$. The twentieth annotated occurrence was not found, but the ℓ-mer reported by PROJECTION had a higher likelihood ratio than the annotated occurrence and, based on its position in the sequence, could conceivably be a second copy of the X box site.

Although the motifs we found are not particularly subtle and indeed have previously been found by existing local search methods [15], the results of these experiments are noteworthy for two reasons. First, we achieved good performance even with a fairly primitive refinement strategy that did not include, e.g., score corrections for motif length or iteration of EM to convergence. We expect that random projection would yield even better performance if adjoined to a more sophisticated local search procedure. Second, because PROJECTION selectively samples good starting points for local search, it uses fewer restarts than the usual approach of starting from each ℓ-mer in the input in turn. As shown in Table 3.3, the number of starting points in the various experiments ranged from 20% to 67% of the input sequence length – substantially fewer than the number of starts required with the usual search initialization. For the X box, the number of starts was 2607, just 13% of the input length. Moreover, the reported motifs were invariably found during several different iterations, so we could have been even more aggressive in reducing m and therefore the

number of starts. Future work should determine how aggressively m can be reduced for "easy" motifs like those of this section without sacrificing sensitivity.

3.4.6 Ribosome Binding Sites

To test PROJECTION's robustness on a very different sort of biological example, we applied it to the problem of finding prokaryote ribosome binding sites. A ribosome binding site problem instance consists of thousands of short DNA sequences ($n = 20$) taken from just upstream of the translation start site of each of an organism's genes. The goal is to identify the site ($\ell \approx 6$) at which the 16S rRNA of the ribosome binds to mRNAs transcribed from the genes. It is known that this binding site is approximately complementary to a short sequence near the $3'$ end of the 16S rRNA [56].

The ribosome binding site problem poses challenges to PROJECTION not encountered in previous examples. First, because of incorrect gene annotation and other limitations, only a fraction of sequences in any problem instance actually contain a 16S rRNA binding site. To model this phenomenon, we set $\hat{t} = t/3$ in Equation (3.2) when determining the number of iterations to perform. Second, the total amount of sequence in this problem is sufficiently large that we cannot choose k to simultaneously satisfy $k < \ell - d$ and achieve a contamination threshold of fewer than tens or hundreds of background ℓ-mers. Instead, we set $k = \ell - d - 1$, as large as possible, and set the bucket size threshold σ to twice the average bucket size $t(n - \ell + 1)/4^k$. This bound should on average select buckets in which motif occurrences (which are numerous in these examples) outnumber background ℓ-mers, that is, buckets with more signal than noise. Because prokaryote genomes often have highly biased composition, some buckets may still be much larger than the threshold σ, but these buckets are discarded by the Poisson filtering heuristic described in Section 3.3.2.

For all ribosome binding site experiments, we chose $\ell = 6$, $d = 1$, and projection size $k = 4$. Table 3.4 shows the problem sizes t, thresholds σ, and numbers of iterations m for each experiment. The motif predicted by PROJECTION is shown in the column labeled "Motif." Each experiment finished in under three minutes on an 800 MHz Intel Pentium III workstation.

Table 3.4: All experiments were performed with projection size $k = 4$. Column headings: t = number of sequences; σ = bucket size threshold, m = number of iterations; "Occurrences" = number of input sequences containing motif with up to one substitution; "16S rRNA" = reverse complement to 3′ end of organism's 16S rRNA, which should be similar to true binding site; "Best z-score" = 5-mer with greatest z-score using the algorithm of [99].

Organism	t	σ	m	Motif	Occurrences	16S rRNA	Best z-score
M. jannaschii	1679	196	14	`AGGTGA`	606	`GGAGGTGATCC`	`GGTGA`
H. influenzae	1716	202	17	`AGGAAA`	639	`TAAGGAGGTGA`	`AAGGA`
T. maritima	1846	216	13	`GGAGGT`	1198	`GAAAGGAGGTG`	`AGGTG`
B. subtilis	4099	480	35	`AGGAGG`	2742	`TAGAAAGGAGG`	`AGGAG`
E. coli	4287	502	35	`AAGGAG`	1306	`TAAGGAGGTGA`	`AGGAG`

Neither likelihood ratio scoring nor combinatorial refinement produced entirely satisfactory results on ribosome binding site problems. The likelihood ratio favored motifs with unusual base composition (e.g. `TCAGGA` for *E. coli*) that were relatively infrequent in the input, while combinatorial refinement and scoring as described at the end of Section 3.3.3 chose very common strings without regard for their composition (e.g. `TAAAAT` for *T. maritima*). In an effort to compromise between the importance of high frequency and meaningful composition, we ultimately chose the motifs in Table 3.4 using the combinatorial score $r(S, d)$ but without performing combinatorial refinement after EM.

Many pieces of evidence corroborate PROJECTION's predicted motifs in Table 3.4. The first is the complementarity of these motifs to the 3′ end of the 16S rRNA sequences (with the possible exception of *H. influenzae*), as shown in Table 3.4. More corroboration follows from the well-known fact that in many bacteria, the binding site for the 16S rRNA during translation initiation is the Shine-Dalgarno sequence `AAGGAGG` or a large substring of it [56, 60]. The reported motifs for the four bacteria in Table 3.4 agree quite well with this sequence. In archaea such as *M. jannaschii*, the 3′ end of the 16S rRNA is missing a few terminal nucleotides compared to the bacterial rRNA sequences, and the 16S rRNA binding site is instead `AGGTGAT` or a large substring of it (Woese, personal communication). Hayes and Borodovsky [44] discovered the motif `GGTGA` in *M. jannaschii* using a Gibbs sampler, and

Tompa [99] discovered similar binding sites in both this and three other archaeal genomes.

Tompa used a very different enumerative statistical algorithm to solve the ribosome binding site problem, ranking motifs by their z-scores. All the motifs found by PROJECTION are in good agreement with the highest-scoring motifs that his algorithm reported. For example, the last column of Table 3.4 shows for each problem instance the pentamer motif, allowing *no* substitutions, with highest z-score. Note the strong overlap between each of these 5-mers and the corresponding PROJECTION prediction.

Randomization is not strictly necessary to find good starting points for refinement in the ribosome binding site problem. There are only fifteen different projections of a hexamer into four dimensions, so one could efficiently test all possible projections rather than picking them at random. Indeed, because the embedded motifs are so short, this particular problem has been addressed enumeratively without resorting to iterative search techniques at all [99]. The significance of PROJECTION's success in finding 16S rRNA binding sites is rather to show that the algorithm is capable of solving motif-finding problems that are quite different both from the typical applications of Section 3.4.5 and from the formal motif model for which it was designed.

3.5 Conclusions and Open Questions

In this chapter, we have described PROJECTION, a new algorithm for finding motifs based on random projection. PROJECTION was designed to efficiently solve problems from the planted (ℓ, d)-motif model, which it does more reliably and for substantially more difficult problem instances than previous motif finding algorithms. PROJECTION is robust to changes in background sequence composition and, to some extent, to long background sequences that create noisier motif finding problems. For $t = 20$ and $n = 600$, our algorithm achieves performance close to the best possible, being limited primarily by the statistical considerations of Section 3.4.2.

Despite its development in a particular formal model, PROJECTION performs well on real biological motif-finding problems, even cases as dissimilar from the model as the ribosome binding site problem. As a general sampling technique for initializing local search, our

method can extend a variety of existing motif-finding algorithms, both increasing their sensitivity to difficult motifs and reducing the number of searches required to find easier motifs. Even so, we continue to seek biological motifs that are more subtle than those described in Sections 3.4.5 and 3.4.6.

While PROJECTION can perhaps be incrementally extended to better handle the extremely difficult synthetic problems of Section 3.4.4, the major open questions about the algorithm focus on how useful it is in practice. In particular, we pose the following three questions. Firstly, is PROJECTION compatible with other, more elaborate local search approaches to motif finding? Secondly, have we any hope of finding motifs with indels? Finally, how can we choose the parameters k and m *a priori* to achieve the considerable reductions in the cost of motif finding observed in Section 3.4.5?

3.5.1 Extensions to Local Search Refinement

The present implementation of PROJECTION uses a rather unsophisticated form of EM refinement. Industrial-strength motif finders based on local search, such as MEME [11] and the recently updated GibbsDNA [62], implement a variety of enhancements to basic iterative refinement, including models that allow motifs to occur multiple times per sequence, inference of the motif length, prior weights on the positions and other properties of motif occurrences, and more refined control of the number of iterations. Fortunately, these enhancements to refinement are largely orthogonal to the process by which starting points for local search are chosen. As the examples of Section 3.4.5 show, even the initial choice of motif length need only be approximately correct to produce good starting points.

Enhancements to the motif model that require fitting additional parameters besides the entries of a WMM should be approached cautiously, both because of their computational cost and because they increase the effective dimension of the search problem, raising the specter of overfitting. However, our experience with the ribosome binding site problem in Section 3.4.6 argues strongly for at least one such extension. In those experiments, neither likelihood ratio scoring nor combinatorial refinement seemed to strike the proper balance in valuing both the frequency with which the motifs occurred and the unusualness

of their composition. We believe that this difficulty stems from the fact that Lawrence and Reilly's EM formulation does not properly account for sequences lacking an occurrence of the motif. Scoring by likelihood is a common and well-founded technique, but by choosing an occurrence from *every* input sequence, we include a large number of background ℓ-mers that corrupt the consensus reported for the motif. A logical future enhancement to our implementation of PROJECTION is to use a probabilistic motif model that accounts for sequences with no motif occurrence, specifically the ZOOPS model [11] used by MEME.

Given the weak dependence between initialization and refinement in local search, the best strategy for enhancing PROJECTION may be not to implement a better local search ourselves but to add our initialization methods to an existing motif finder. The ready availability of MEME's source code makes it a logical candidate for future integration with PROJECTION.

3.5.2 Handling Gaps

A major open question for PROJECTION is how to extend it to find motifs whose occurrences contain indels as well as substitutions. The previous chapter discussed the problems of using random projection for gapped pairwise alignments; the same issues raised there apply to multiple alignment as well. The problem is further complicated by the fact that the WMM itself does not permit indels, so that refinement as well as initialization presents barriers to gapped motif finding.

A potentially more tractable restriction of gapped motif finding is the following: find a motif consisting of $x \geq 2$ ungapped segments separated by $x - 1$ intervening spacers whose length may vary in each occurrence. Such motifs, which are known (with $x = 2$) in yeast [89] and bacteria [62], arise because transcription factor molecules are often flexible dimers or multimers, each part of which separately contacts a particular pattern in the DNA sequence. Even if random projection cannot be extended to handle motifs composed of gapped segments directly, it may be possible to find promising ungapped segments, in the style of Rigoutsos and Floratos [79], then attempt to paste these segments together into a full motif during refinement.

Finding and assembling motif segments seems feasible for, e.g., the motifs reported by McCue et al. in *E. coli* [62], each of which consists of two ungapped segments of seven to eight bases with an intervening variable-length spacer of up to seven bases. In experiments to find motifs conserved between *E. coli* and related bacteria, PROJECTION frequently found motifs reported by McCue et al. by identifying one ungapped segment. Hence, ungapped segments of these motifs appear sufficiently well-conserved that PROJECTION can identify them for use as inputs to a refinement algorithm that assembles the segments.

3.5.3 Better Parameter Selection

In Section 3.4.5, we observed that PROJECTION succeeded in finding biologically meaningful motifs using many fewer starting points for refinement than the total length of the input sequence. Our initialization strategy therefore holds promise for reducing the number of starting points used in existing local search algorithms such as MEME. However, to put this idea into practice, future work must determine how to choose the number of iterations m and projection size k when the desired motif does not follow the planted (ℓ, d) model.

The primary goal of PROJECTION – causing at least σ motif occurrences to hash to the same bucket – remains the same regardless of the motif model. The probability of succeeding at this goal is easily computed in the planted (ℓ, d) model, both because each motif occurrence has a fixed number of mutations and because the mutations occur independently in each occurrence. In principle, the same probability could be computed for motifs with *any* given distribution of mutations; in practice, such computations may be difficult when mutations in different occurrences do not appear independently. More realistic motif models therefore present computational challenges for parameter selection.

Real biological motifs typically show strong conservation at certain positions, which may act as critical contact points for a transcription factor, and lesser conservation elsewhere. A formal model that captures this knowledge yet may be tractable for parameter selection is the following. A motif is composed of several different types of positions: strongly conserved positions, with a very low rate ρ_1 of mutation from their consensus bases; moderately conserved positions, with a larger deviation rate ρ_2; and wholly non-conserved positions

with base frequencies similar to that of the background sequence. The model would specify the proportions of strongly, weakly, and non-conserved positions in the motif, which provides sufficient information to estimate (numerically if not analytically) the chance that a given projection succeeds in hashing σ motif occurrences together. The desired probability must be summed over all partitions of a projection's k positions into the three types, but this computation seems feasible for the small values of k used in the algorithm.

While the above motif model is much closer to reality than the planted (ℓ, d) model, it is still consensus-based; in particular, it offers no easy way to model the common case of positions that always contain, e.g., a purine but do not favor A over G or vice versa. Indeed, PROJECTION itself cannot easily discover conserved purine or pyrimidine residues, since it only projects together ℓ-mers with a single, fixed base at a given position. To work around this limitation, the basic projection operation could be modified as follows. Each time a position is chosen for a projection, with some probability that position samples only the purine-pyrimidine class of every base; in other words, if position p of an ℓ-mer contains A, then the projection's value at p is "purine" rather than "A". This revised form of projection directly captures conservation of a class of bases, but it is not yet clear how it affects PROJECTION's balance between sensitivity and efficiency for real motif finding problems.

The idea of extending similarity beyond a simple match-or-mismatch decision is useful to LSH-ALL-PAIRS as well as to PROJECTION. In fact, extensions like the one described above can be formally related to well-known measures of sequence similarity. We will explore the relationship between extensions of random projection and general similarity score functions more fully in the next chapter.

Chapter 4

SIMULATION OF ALIGNMENT SCORING FUNCTIONS

Random projection as we have described it is closely tied to the number of substitutions between two sequences. While this measure of similarity has proven practically effective in LSH-ALL-PAIRS and PROJECTION, it is only one of many ways to score ungapped alignments. In this chapter, we pose the problem of *score simulation*, or how to implement more complex alignment score functions using random projection. We provide some basic results on this problem, including a simulation of the DNA-PAM-TT family of scoring functions [96], and sketch an approach to simulating arbitrary ungapped similarity scores.

4.1 Problem Definition

Biologically informed alignment score functions assign unequal penalties to different kinds of substitution mutations. The different penalties reflect the fact that some substitutions are more common than others, either because they are biochemically more likely or because they are not under strong negative selective pressure. Score functions with unequal substitution penalties are commonly used in both DNA and protein sequence comparisons, as the following examples illustrate:

- Substitutions in DNA sequence are of two types: *transitions*, in which one purine (A or G) or pyrimidine (C or T) changes to another base of the same class, and *transversions*, in which a purine becomes a pyrimidine or vice versa. Transitions are biochemically more probable than transversions and, when they occur in the third base positions of codons, are less likely to alter an encoded protein sequence. Hence, alignment scoring functions like the DNA-PAM-TT family assign a larger penalty to a transversion than to a transition.

- The Dayhoff PAM matrices [29] for protein comparison model the constraints imposed

by natural selection on substitutions of one amino acid for another. These constraints arise from chemical properties, such as an amino acid's physical size and charge, that determine the three-dimensional structure of a protein. For example, leucine and isoleucine are quite similar in size and other properties, so substitutions between them are usually not subject to strongly negative selection and hence receive a small penalty. In contrast, tryptophan has a much larger side chain than leucine, so replacing one amino acid by the other is more likely to disrupt a protein's structure. Leucine-tryptophan substitutions therefore receive a larger penalty.

By incorporating scoring functions with unequal substitution penalties, algorithms such as LSH-ALL-PAIRS can perform a more biologically informed similarity search. As we shall shortly see, however, building such score functions into random projection poses technical difficulties.

4.1.1 Limitations of Projection

Random projection is closely tied to the *Hamming distance*, or number of substitutions, between sequences. In particular, the fundamental result of Inequality (2.1) states that sequences at a smaller Hamming distance from each other are more likely to project to the same value. Unfortunately, Hamming distance is often a poor approximation to real alignment score functions, as the following example illustrates.

Let $H(s, \bar{s})$ denote the Hamming distance between two sequences s and $\bar{s}$. Consider the three 7-mer sequences $s_1 = \texttt{AAAAAAA}$, $s_2 = \texttt{AGAGAGA}$, and $s_3 = \texttt{ATATATA}$, and suppose they are compared using an alignment score function $\mathcal{F}$ that assigns a bonus of $+1$ to each match and unequal penalties of 0 and -1 to a transition and transversion respectively. Then

$$\begin{aligned} \mathcal{F}(s_1, s_2) &= 4 \\ \mathcal{F}(s_1, s_3) &= 1. \end{aligned}$$

However, Hamming distance fails to distinguish these pairs of sequences:

$$H(s_1, s_2) = H(s_1, s_3) = 3.$$

This example has serious implications for random projection and algorithms based on it. The Hamming distance between two ℓ-mers (and hence, the probability that they project to the same value under a random projection) may be only weakly correlated to their score. For arbitrary thresholds in Hamming distance, we can parameterize LSH-ALL-PAIRS to find *only* those similarities above the threshold and, as the number of iterations m grows large, *all* such similarities with probability one. In contrast, no Hamming distance threshold, regardless of the number of iterations, suffices to find all and only those similarities above a threshold under a more general score function $\mathcal{F}$. For such scores, there exist thresholds (e.g. $\theta = 2$ in the example) for which LSH-ALL-PAIRS necessarily commits either false positive or false negative errors, however it is parameterized and even in the limit of large m.

One obvious way to remedy the disconnect between projection and non-Hamming score functions is to postfilter LSH-ALL-PAIRS' output to remove similarities below the desired score threshold. We took this approach in Chapter 2, but it only solves half the problem: the algorithm may still miss any number of similarities above the threshold whose Hamming distances are too large to report, regardless of how many iterations it performs. In general, we can make no guarantee as to LSH-ALL-PAIRS' sensitivity under postfiltering without setting the distance bound d for reported similarities so high as to be impractically expensive.

A second approach to integrating general scoring with projection is to extend the definition of candidate ℓ-mer pairs, allowing candidates whose projection values are similar but not equal. Rigoutsos and Califano take this approach in their FLASH algorithm [78]. Let h be a projection on ℓ-mers; FLASH treats two ℓ-mers s_1 and s_2 as a candidate pair under a score function $\mathcal{F}$ whenever $\mathcal{F}(h(s_1), h(s_2))$ exceeds some threshold θ'. This approach works fine for comparing a short query sequence to a large database but does not scale to the all-pairs problem on large sequences. Simple equality of projection values is an equivalence relation that lets LSH-ALL-PAIRS implicitly generate all candidate ℓ-mer pairs in time linear in the input size. In contrast, FLASH's approach requires quadratic time to produce all candidate pairs.

Extending random projection to unequal substitution penalties with rigorous sensitivity guarantees *and* reasonable efficiency requires a more powerful tool than simple postfiltering. To find such a tool, we turn to the theory of *isometric embeddings*.

4.1.2 Isometries and Score Simulation

Let X be a set equipped with a metric distance d. We take from geometry the idea that X can sometimes be *embedded* in a Hamming space of sufficiently high dimension k. In other words, each point $x \in X$ can be mapped to some $\phi(x) \in \{0,1\}^k$ such that, for any $x, y \in X$,

$$H(\phi(x), \phi(y)) = d(x, y).$$

The mapping ϕ is said to be an *isometry* on X because it preserves distances between X's elements. For sequences, it will be more convenient to use isometries into a *generalized Hamming space* whose points are vectors over some alphabet Σ rather than zero and one; below, the term "Hamming space" always refers to such a generalized space.

Isometric embeddings have been widely studied; see, for example, results of Linial et al. [61, Section 5] and applications of random projection to points in the ℓ_1 metric by Gionis et al. [38]. We apply these tools to alignment score functions by introducing the idea of *score simulation*.

To begin, recall the formal definition of an alignment score function from Section 1.3.1. Because this chapter considers only ungapped alignments, we slightly restrict the definition here. Let $\sigma(x, y)$ be a function from the columns of an alignment to integers. A single column is a pair of characters from the sequence alphabet Σ, so that $\sigma(x, y)$ is specified by its $|\Sigma|^2$ possible values. In practice, σ is often written as a square matrix of dimension $|\Sigma|$. Each column score σ induces a family of alignment score functions $\mathcal{F}_\sigma^\ell : \Sigma^\ell \times \Sigma^\ell \to \mathcal{Z}$ on pairs of ℓ-mers defined by

$$\mathcal{F}_\sigma^\ell(s_1, s_2) = \sum_{i=1}^{\ell} \sigma(s_1[i], s_2[i]).$$

Let $\Delta^\ell : \mathcal{Z} \to \mathcal{Z}^{0+}$ be a function mapping alignment scores on ℓ-mer pairs to non-negative distances, and let $\Phi^\ell : \Sigma^\ell \to (\Sigma')^k$ be an embedding of ℓ-mer strings into a Hamming space with dimension k and alphabet Σ'.

Definition 4.1 The pair $\left\langle \Delta^\ell, \Phi^\ell \right\rangle$ *simulates* a score function $\mathcal{F}_\sigma^\ell$ if, for all $s_1, s_2 \in \Sigma^\ell$ and all $\theta \in \mathcal{Z}$, the following three properties hold:

1. $\mathcal{F}_\sigma^\ell(s_1, s_2) > \theta$ iff $\Delta^\ell(\mathcal{F}_\sigma^\ell(s_1, s_2)) < \Delta^\ell(\theta)$.

2. $\mathcal{F}_\sigma^\ell(s_1, s_2) < \theta$ iff $\Delta^\ell(\mathcal{F}_\sigma^\ell(s_1, s_2)) > \Delta^\ell(\theta)$.

3. Φ^ℓ is an isometry on Σ^ℓ with respect to the distance $\Delta^\ell \cdot \mathcal{F}_\sigma^\ell$, i.e.

$$H\left(\Phi^\ell(s_1), \Phi^\ell(s_2)\right) = \Delta^\ell(\mathcal{F}_\sigma^\ell(s_1, s_2)).$$

Note that the fact that $\Delta^\ell \cdot \mathcal{F}_\sigma^\ell$ is equivalent to a Hamming distance implies that it is a metric distance.

If we can find a pair $\left\langle \delta^\ell, \Phi^\ell \right\rangle$ that simulates $\mathcal{F}_\sigma^\ell$, we can implement pairwise local alignment using $\mathcal{F}_\sigma^\ell$ via random projection as follows. For any collection of ℓ-mers $S = s_1 \ldots s_n$, map each s_i to its image $\Phi^\ell(s_i)$. Call the resulting collection of k-mers $\Phi^\ell(S)$. If a problem calls for finding all (and only) pairs of ℓ-mers $s_i, s_j \in S$ such that $\mathcal{F}_\sigma^\ell(s_i, s_j) \geq \theta$, we can equally well find all pairs of k-mers $\Phi^\ell(s_i), \Phi^\ell(s_j) \in \Phi^\ell(S)$ such that $H\left(\Phi^\ell(s_i), \Phi^\ell(s_j)\right) \leq \Delta^\ell(\theta)$. The latter computation can be implemented directly by LSH-ALL-PAIRS and can be parameterized to return *only* and (in the limit of large m) *all* those k-mers corresponding to ℓ-mer pairs above the threshold θ.

Although simulation is defined with respect to a particular sequence length ℓ, it would be convenient if a single construction could produce simulations for *every* ℓ. The following lemma gives a sufficient condition for the existence of such a construction, which we will use repeatedly in the remainder of this chapter.

Lemma 4.2 Suppose $\langle \delta, \phi \rangle$ simulates $\mathcal{F}_\sigma^1 = \sigma$. If δ is an affine linear function, then for every $\ell \geq 1$, there exists a simulation $\left\langle \Delta^\ell, \Phi^\ell \right\rangle$ of $\mathcal{F}_\sigma^\ell$.

Proof: If δ is affine linear, it must be that $\delta(z) = a - bz$ for $b > 0$, since higher scores map to lower distances and vice versa. Define Δ^ℓ in terms of δ as follows:

$$\Delta^\ell(\mathcal{F}_\sigma^\ell(s_1, s_2)) = \sum_{i=1}^{\ell} \delta(\sigma(s_1[i], s_2[i])).$$

By this definition, Δ^ℓ has the form $\Delta^\ell(z) = a\ell - bz$. For any ℓ-mer s, let $\Phi^\ell(s)$ be the concatenation of $\phi(s[i])$ for $1 \leq i \leq \ell$.

We claim that $\left\langle \Delta^\ell, \Phi^\ell \right\rangle$ simulates $\mathcal{F}_\sigma^\ell$. We first show that Δ^ℓ satisfies the inequality constraints of Definition 4.1, in particular that

$$\mathcal{F}_\sigma^\ell(s_1, s_2) > \theta \text{ iff } \Delta^\ell(\mathcal{F}_\sigma^\ell(s_1, s_2)) < \Delta^\ell(\theta)$$

and vice versa. If $\mathcal{F}_\sigma^\ell(s_1, s_2) > \theta$, then by the definition of Δ^ℓ and the assumption that δ is affine linear,

$$\begin{aligned} \Delta^\ell(\mathcal{F}_\sigma^\ell(s_1, s_2)) &= a\ell - b\mathcal{F}_\sigma^\ell(s_1, s_2) \\ &< a\ell - b\theta \\ &= \Delta^\ell(\theta). \end{aligned}$$

Conversely, we have that, if $\Delta^\ell(\mathcal{F}_\sigma^\ell(s_1, s_2)) < \Delta^\ell(\theta)$, then $a\ell - b\mathcal{F}_\sigma^\ell(s_1, s_2) < a\ell - b\theta$, and so $\mathcal{F}_\sigma^\ell(s_1, s_2) > \theta$. The other required constraint is proved similarly.

It remains to show that the distance $\Delta^\ell \cdot \mathcal{F}_\sigma^\ell$ is embeddable in Hamming space. Observe that for any $s_1, s_2 \in \Sigma^\ell$,

$$\begin{aligned} \Delta^\ell(\mathcal{F}_\sigma^\ell(s_1, s_2)) &= \sum_{i=1}^{\ell} \delta(\sigma(s_1[i], s_2[i])) \\ &= \sum_{i=1}^{\ell} H\left(\phi(s_1[i]), \phi(s_2[i])\right) \\ &= H\left(\Phi^\ell(s_1), \Phi^\ell(s_2)\right). \end{aligned}$$

We conclude that $\Delta^\ell \cdot \mathcal{F}_\sigma^\ell$ is indeed Hamming-embeddable, and so the lemma is proven. ∎

In the remainder of this chapter, we address the problem of finding simulations for various families of alignment score functions:

Problem 4.1 (Score Simulation Problem) Given a column score function σ, find $\langle \delta, \phi \rangle$ that simulates σ, such that δ is affine linear.

We will show that simulations exist for a number of practically important alignment scoring functions and discuss how to implement score simulation in practice as part of algorithms based on random projection.

4.2 Simulation of DNA-PAM and DNA-PAM-TT

We begin by giving simulations for the DNA-PAM and DNA-PAM-TT families of score functions, devised by States et al. in [96]. DNA-PAM can actually be simulated with only the trivial identity embedding $\phi(x) = x$, but describing its simulation introduces the argument structure used in the more elaborate simulation of DNA-PAM-TT. We also describe how to modify LSH-ALL-PAIRS, including the procedure for choosing parameters m and k, to work with the DNA-PAM-TT embedding.

4.2.1 Simulating DNA-PAM

The DNA-PAM family of score functions is based on a discrete-time Markov model of mutation in a DNA sequence composed of independent bases. During each time step, each base remains unchanged with probability 0.99; with probability 0.01, the base mutates to a different base chosen uniformly at random. Let M^k_{xy} be the probability that base x changes to base y in k steps, and let $p(x)$ and $p(y)$ be the background frequencies of x and y respectively. The DNA-PAM-K score matrix is an integer-valued approximation of a log likelihood ratio scoring matrix LR given by

$$LR(x, y) = \log \frac{p(x)M^k_{xy}}{p(x)p(y)}.$$

The numerator of $LR(x, y)$ reflects the probability that, if two sequences initially share base x in a given position, one of the sequences will mutate from x to y; in contrast, the denominator reflects the chance of a match between two unrelated bases. DNA-PAM assumes a background with equal base frequencies, so $p(x) = p(y) = 1/4$.

Let σ^k denote the column scoring function of DNA-PAM-K. Every σ^k has the following properties:

1. $\sigma^k(x, x)$ is the same for every x, and $\sigma^k(x, y)$ is the same for every $x \neq y$. These properties hold because all bases have the same probabilities of conservation and of mutation to each other base in the Markov model, and because the background frequency of each base is the same.

2. $\sigma^k(x,x) > \sigma^k(x,y)$ for $x \neq y$. This property holds because, if M is the matrix describing the Markov process of DNA-PAM for a single time step, M^k has diagonal probabilities equal to $(0.98^k + 1)/2$, which exceeds $\frac{1}{2}$ for any finite k.

Because all match bonuses and all substitution penalties in DNA-PAM-K are identical, it is easily simulated:

Lemma 4.3 Every score function in the DNA-PAM family can be simulated using an affine linear mapping and the identity embedding $\phi(x) = x$.

Proof: The following construction simulates σ^k for any k. Let $\alpha = \sigma^k(x,x)$ and $\beta = \sigma^k(x,y)$ for bases $x \neq y$, and define $\delta^k(z) = (\alpha - z)/(\alpha - \beta)$. The function δ^k, and indeed *any* affine linear function δ with a negative linear term, satisfies the inequality constraint

$$z > \theta \text{ iff } \delta(z) < \delta(\theta)$$

and vice versa.

It remains to show that the base alphabet can be embedded in Hamming space while preserving the distance $\delta^k \cdot \sigma^k$. Such an embedding is trivial because, by construction,

$$\delta^k(\sigma^k(x,y)) = H(x,y).$$

In other words, $\delta^k \cdot \sigma^k$ assigns matching bases a distance of zero and mismatched bases a distance of one, so it is equivalent to Hamming distance. Hence, the identity mapping $\phi(x) = x$ suffices. ∎

Note that LSH-ALL-PAIRS can simulate DNA-PAM-K with no changes: if the user specifies a score threshold θ, simply set $d = \delta^k(\theta)$ and proceed as usual.

4.2.2 Simulating DNA-PAM-TT

The DNA-PAM-TT family of score functions derives from the same evolutionary model as DNA-PAM, except that each transition occurs with probability 0.006 per time step, while each transversion has probability 0.002. Let DNA-PAM-TT-K be the score matrix defined for these probabilities by a log-likelihood ratio construction and uniform background analogous

to that used for DNA-PAM-K, and let σ^k_{tt} be the corresponding column scoring function. The key properties listed for σ^k also apply to σ^k_{tt}, with the exception that there are *two* possible values of $\sigma^k_{tt}(x, y)$ when $x \neq y$: a penalty $-\pi_s$ for a transition, and a larger penalty $-\pi_v$ for a (less probable) transversion. As we saw in Section 4.1.1, one-dimensional Hamming distance cannot satisfy the constraints of Definition 4.1 for DNA-PAM-TT; however, a higher-dimensional embedding is equal to the task.

Theorem 4.4 Every score function σ^k_{tt} in the DNA-PAM-TT family can be simulated using an affine linear mapping and a Hamming embedding of dimension $\sigma^k_{tt}(x, x) + \pi_v$.

Proof: Define the score-to-distance mapping $\delta^k_{tt}(z) = \sigma^k_{tt}(x, x) - z$. As in Lemma 4.3, this affine linear function satisfies the required inequality constraints, so it suffices to exhibit a Hamming embedding of the base alphabet that preserves the distance $\delta^k_{tt} \cdot \sigma^k_{tt}$. In what follows, define the images of the two substitution penalties under δ^k_{tt} to be

$$\begin{aligned} \alpha_s &= \delta^k_{tt}(-\pi_s) \\ \alpha_v &= \delta^k_{tt}(-\pi_v). \end{aligned}$$

The higher probability of transition vs. transversion in DNA-PAM-TT ensures that $\alpha_v \geq \alpha_s$.

Let Σ_b be the base alphabet. Define an embedding $\phi_0 : \Sigma_b \rightarrow (\Sigma_b \cup \{0, 1\})^2$ into a *weighted Hamming space* as shown in Figure 4.1B. Each base x is mapped to a two-dimensional vector $\phi_0(x) = \langle b_x, t_x \rangle$, where $b_x = x$ and t_x is zero if x is a purine or one if it is a pyrimidine. The two dimensions are assigned weights α_s and $\alpha_v - \alpha_s$, respectively; because $\alpha_v \geq \alpha_s$, both dimensions have nonnegative weights.

In a weighted Hamming space, the distance $D(v, w)$ between two vectors v and w is the sum of the weights of all dimensions in which they do not match. Under this interpretation,

Lemma 4.5 The embedding ϕ_0 is an isometry on Σ_b; that is,

$$D(\phi_0(x), \phi_0(y)) = \delta^k_{tt}(\sigma^k_{tt}(x, y)).$$

Proof: If $x = y$, the vectors $\phi_0(x)$ and $\phi_0(y)$ match in both dimensions and so

$$\begin{aligned} D(\phi_0(x), \phi_0(y)) &= 0 \\ &= \delta^k_{tt}(\sigma^k_{tt}(x, y)). \end{aligned}$$

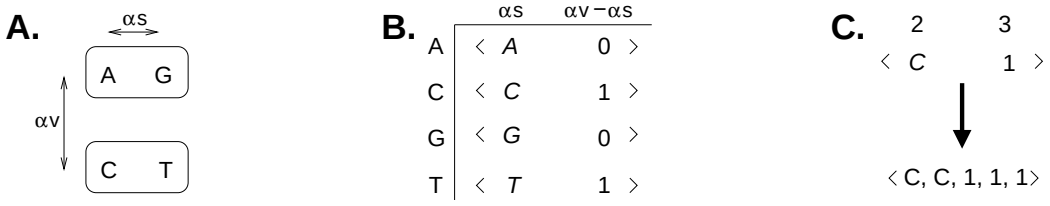

Figure 4.1: An isometric embedding that simulates the DNA-PAM-TT family of score functions. (A) a pair of bases representing a transition or transversion are respectively placed at distances α_s and α_v. (B) a two-dimensional weighted Hamming embedding that reflects the distances in part A. (C) An embedding of a weighted Hamming space in an unweighted space.

If the mutation $x \to y$ is a transition, then x and y are either both purines or both pyrimidines, in which case only their second dimensions match and

$$\begin{aligned} D(\phi_0(x), \phi_0(y)) &= \alpha_s \\ &= \delta_{tt}^k(-\pi_s) \\ &= \delta_{tt}^k(\sigma_{tt}^k(x, y)). \end{aligned}$$

If the mutation is a transversion, neither dimension matches, so

$$\begin{aligned} D(\phi_0(x), \phi_0(y)) &= \alpha_s + (\alpha_v - \alpha_s) \\ &= \alpha_v \\ &= \delta_{tt}^k(-\pi_v) \\ &= \delta_{tt}^k(\sigma_{tt}^k(x, y)). \end{aligned}$$

∎

To complete the Hamming-space embedding, observe that every weighted Hamming space can be isometrically embedded in an unweighted space as follows:

Lemma 4.6 Let V be a weighted Hamming space with non-negative integer weights over an alphabet Σ. Let W be the total weight of all dimensions in V. Then V can be isometrically embedded in an unweighted Hamming space of dimension W.

Proof: Define the embedding $\phi_1 : V \to \Sigma^W$ as follows. If the ith dimension of V has weight w_i, ϕ_1 replicates this dimension w_i times. For example, Figure 4.1C illustrates an

embedding of a two-dimensional weighted Hamming space with dimension weights 2 and 3. The first dimension of each vector in V is replicated twice, while the second is replicated three times.

Because each dimension i is repeated w_i times with unit weight in the unweighted Hamming space, the embedding ϕ_1 is an isometry on V. ∎

Corollary 4.7 The weighted Hamming space described in Figure 4.1B can be embedded in an unweighted Hamming space of dimension

$$\begin{aligned} \alpha_s + (\alpha_v - \alpha_s) &= \alpha_v \\ &= \delta_{tt}^k(-\pi_v) \\ &= \sigma_{tt}^k(x,x) + \pi_v. \end{aligned}$$

We conclude that $\phi_1 \cdot \phi_0$ isometrically embeds the base alphabet with distance $\delta_{tt}^k \cdot \sigma_{tt}^k$ in unweighted Hamming space, which completes the proof of the theorem. ∎

4.2.3 Adapting LSH-ALL-PAIRS *to* DNA-PAM-TT

We have shown that the DNA-PAM-TT family can be simulated, which implies that any score function from this family can in theory be implemented by the LSH-ALL-PAIRS algorithm. However, the naive implementation described in Section 4.1.2 is needlessly inefficient in practice. In this section, we describe a more realistic implementation of DNA-PAM-TT in LSH-ALL-PAIRS and show how to compute the parameters m and k for any threshold value of these score functions.

Implementation Strategy

The major implementation cost of adapting LSH-ALL-PAIRS to DNA-PAM-TT is the increase in dimension that occurs when a sequence is embedded: for commonly used DNA-PAM-TT matrices, the embedded representation of a sequence is ten to thirty times larger than the original. A practical implementation should therefore *never explicitly store* a sequence's embedded representation.

LSH-ALL-PAIRS operates directly on sequences at two points: once when it computes the projection of each ℓ-mer in the input, and again when it computes the actual distances between pairs of ℓ-mers during the checking phase. In the naive implementation, both of these operations work on ℓ-mers' embedded representations. However, simulation is needed only for computing projections, not for checking ℓ-mers. Rather than comparing the Hamming distance $H\left(\Phi^\ell(s_1), \Phi^\ell(s_2)\right)$ to the threshold $\Delta^\ell(\theta)$ to check each candidate pair (s_1, s_2), an implementation can equivalently test the original score function $\mathcal{F}^\ell_\sigma(s_1, s_2)$ against the original threshold θ.

The filtering phase of LSH-ALL-PAIRS must be able to compute random projections of an ℓ-mer's embedded representation, but it may do so implicitly rather than storing the embedded representation explicitly. For example, the procedure GET-EMBED-TT-K shown in Figure 4.2 takes an ℓ-mer s and returns any single character from its embedded representation $\Phi^\ell(s)$ under the simulation of DNA-PAM-TT-K. Each character of the original ℓ-mer maps to a run of α_v contiguous dimensions in the embedding, and each run contains a fixed number of copies of each dimension from Figure 4.1B, so the procedure can easily determine the index i of the character being sampled and whether the dimension d sampled from that character contains the character itself or its purine/pyrimidine class. Much of the computation in this procedure need be done only once when a particular projection is chosen, so projection values can still be computed efficiently without explicitly representing $\Phi^\ell(s)$.

Choosing Parameters m and k

The LSH-ALL-PAIRS algorithm is parameterized by the similarity length ℓ, the maximum allowed number of substitutions d, and a number of iterations m and projection size k chosen to upper-bound the chance of missing a d-similarity. When the algorithm is adapted to DNA-PAM-TT, however, several changes occur: the substitution bound d is replaced by a minimum score threshold θ; the projection size k and the positions in each projection are chosen with respect to the embedded representation, not the original ℓ-mer; and the optimal balance between projection and checking costs must be recomputed. To implement

GET-EMBED-TT-K(s, j)

s: sequence being projected

j: position to be read from $\Phi^{\ell}(s)$

$\alpha_v \leftarrow \sigma^k_{tt}(x, x) + \pi_v$

$i \leftarrow \lfloor j/\alpha_v \rfloor$

$d \leftarrow j \bmod \alpha_v$

if $d < \sigma^k_{tt}(x, x) + \pi_s$

  return $s[i]$

else

  return 0 if $s[i] \in \{A, G\}$, 1 otherwise

end

Figure 4.2: a procedure GET-EMBED-TT-K to compute the character at the jth position in the embedded representation of the sequence s, as described for DNA-PAM-TT-K in Section 4.2.2. To simplify the pseudocode, all indices are assumed to be zero-based rather than one-based.

the procedure for parameter selection described in Section 2.3.4, we must compute LSH-ALL-PAIRS' false positive and false negative rates for sequences in the embedded representation. In what follows, $\left\langle \Delta^\ell, \Phi^\ell \right\rangle$ refers to the simulation of $\mathcal{F}^\ell_{\sigma^k_{tt}}$, constructed from the single-base simulation of σ^k_{tt} according to the procedure of Lemma 4.2.

Let (s_1, s_2) be a pair of ℓ-mers with score $\mathcal{F}^\ell_{\sigma^k_{tt}}(s_1, s_2) \geq \theta$. The Hamming distance $H\left(\Phi^\ell(s_1), \Phi^\ell(s_2)\right)$ between these ℓ-mers' embedded representations is by construction at most $\Delta^\ell(\theta)$. Hence, if the algorithm uses projection size k and runs for m iterations using projections with positions chosen independently at random with replacement, its false negative rate ρ_{fn} for similarities scoring at least θ is bounded by

$$\rho_{fn} \leq \left[1 - \left(1 - \frac{\Delta^\ell(\theta)}{\alpha_v \ell}\right)^k\right]^m \tag{4.1}$$

where α_v, the distance between the two bases of a transversion, is the dimensionality of each base's embedded representation. Inequality (4.1) is exactly analogous to Inequality (2.2), the false negative rate estimated in Chapter 2; like that earlier inequality, it determines the smallest feasible m for any value of k.

The false positive rate ρ_{fp}, derived analogously to Equation (2.4), is somewhat more difficult to compute explicitly. For fixed m and k and a score threshold θ, the same argument used in Section 2.3.4 shows that this rate is given by

$$\rho_{fp} = m \sum_{\theta' < \theta} \Pr\left[\mathcal{F}^\ell_{\sigma^k_{tt}}(s_1, s_2) = \theta'\right] \left(1 - \frac{\Delta^\ell(\theta')}{\alpha_v \ell}\right)^k \tag{4.2}$$

where the probability is computed for an ℓ-mer pair (s_1, s_2) drawn at random from the background sequence. The discrete sum is possible because the alignment score function is integer-valued, and it is finite because any two ℓ-mers must have an alignment score of at least $-\ell\pi_v$. However, the probability that two random ℓ-mers have pairwise score θ' is difficult to compute explicitly, as one score can arise from multiple combinations of transitions and transversions.

An alternative, more explicit formula for ρ_{fp} is derived as follows. The probability that a pair of unrelated ℓ-mers in the background sequence exhibits exactly d_s transitions and

d_v transversions is given by the trinomial density

$$\tau_{p_s,p_v,\ell}[d_s,d_v] = \frac{\ell\,!}{d_s!d_v!(\ell-d_s-d_v)!}\; p_s^{d_s} p_v^{d_v} (1-p_s-p_v)^{\ell-d_s-d_v}$$

where p_s and p_v are the probabilities that two randomly chosen background bases form a transition or a transversion, respectively. These probabilities are easily computable from the background distribution; for example, when all bases are equally frequent, $p_s = 0.25$ and $p_v = 0.5$. Let α_s and α_v be the distances between bases in a transition and a transversion respectively according to $\delta_{tt}^k \cdot \sigma_{tt}^k$. If the ℓ-mers s_1 and s_2 have d_s transitions and d_v transversions, then a random projection h sampling k positions with replacement projects them together with probability

$$\Pr\left[h(\Phi^\ell(s_1)) = h(\Phi^\ell(s_2))\right] = \left(1 - \frac{\alpha_s d_s + \alpha_v d_v}{\alpha_v \ell}\right)^k.$$

Let $d_v^*(\theta)$ be the fewest possible transversions in any ℓ-mer pair with score less than θ, and let $d_s^*(\theta, d_v)$ be the fewest possible transitions in such a pair if it has exactly d_v transversions. Then we may write ρ_{fp} for given m, k, and θ as follows:

$$\rho_{fp} = m \sum_{d_v=d_v^*(\theta)}^{\ell} \sum_{d_s=d_s^*(\theta,d_v)}^{\ell-d_v} \tau_{p_s,p_v,\ell}[d_s,d_v] \left(1 - \frac{\alpha_s d_s + \alpha_v d_v}{\alpha_v \ell}\right)^k. \tag{4.3}$$

The lower bounds on the summation derive from the constraint that each of its terms reflect an ℓ-mer pair that does not pass the score threshold θ, or equivalently that $\alpha_s d_s + \alpha_v d_v > \Delta^\ell(\theta)$. Solving this inequality for d_s, the lower bound on the inner sum is given by

$$d_s^*(\theta, d_v) = \left\lfloor \frac{\Delta^\ell(\theta) - \alpha_v d_v}{\alpha_s} \right\rfloor + 1.$$

The lower bound for the outer sum is the smallest d_v for which $d_s^*(\theta, d_v) \le \ell$. Plugging the above expression for d_s^* into this inequality and solving for d_v gives

$$d_v^*(\theta) = \left\lfloor \frac{\Delta^\ell(\theta) - \alpha_s \ell}{\alpha_v} \right\rfloor + 1.$$

Both of these bounds should be clamped to be ≥ 0.

Using the revised false positive and false negative rates above, the optimal m and k for LSH-ALL-PAIRS may be computed as described in Section 2.3.4. For example, Figure 4.3

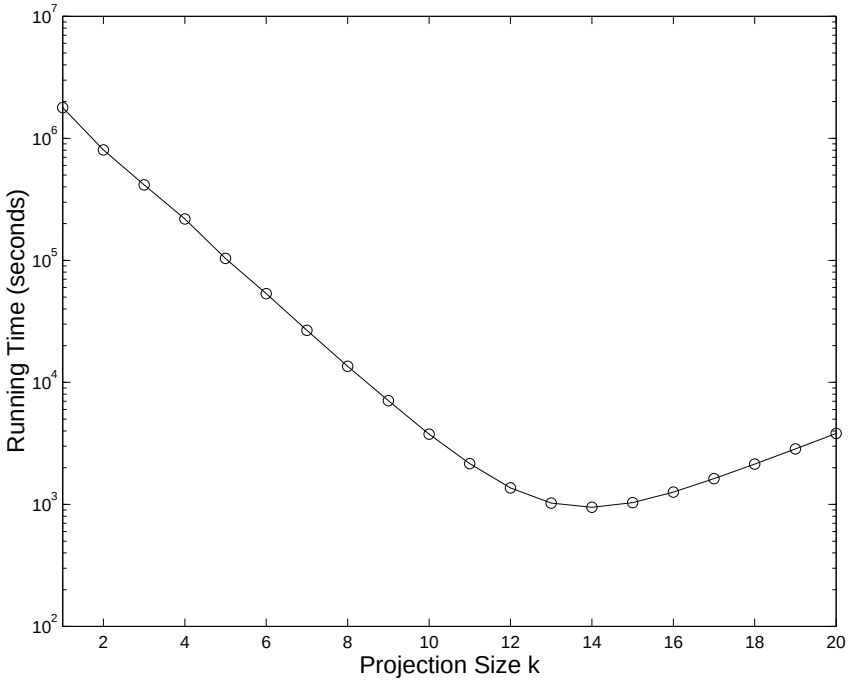

Figure 4.3: total computational cost as a function of projection size k for the score function DNA-PAM-TT-20 with $\ell = 75$ and $\theta = 324$. We assume a comparison between two one-megabase sequences with equal base frequencies using the timings of Figure 2.4. Minimal cost is achieved for $k = 14$, $m = 185$.

shows the optimal cost curve for comparing two one-megabase sequences with equal base frequencies, assuming $\ell = 75$, the score function DNA-PAM-TT-20, the per-iteration and per-comparison timings of Figure 4.3, and a score threshold $\theta = 324$ (the $p = 0.05$ significance level for ungapped alignments under DNA-PAM-TT-20 as predicted by Karlin-Altschul theory). The predicted cost is minimized for $k = 14$ and $m = 185$.

4.3 Simulating Arbitrary Score Functions

The DNA-PAM and DNA-PAM-TT families represent useful but highly specialized classes of score function. In particular, they have the property that, while different substitutions may be scored differently, every matching base pair is scored the same. This property is by no means required of arbitrary similarity scores. For example, if the background model

of DNA-PAM is extended to allow unequal base frequencies, the diagonal elements of the log likelihood ratio matrix become unequal, reflecting the fact that certain matches are now more likely than others to occur by chance in the absence of any conservation. The PAM matrices for protein comparison and the empirically derived score function used by PipMaker on DNA sequences also have unequal base frequencies.

The following lemma shows that, unfortunately, score simulation as we have defined it cannot cope with score functions whose match bonuses are unequal.

Lemma 4.8 Let $\sigma : \Sigma \times \Sigma \to \mathcal{Z}$ be a column scoring function, and suppose there exist $x, y \in \Sigma$ for which $\sigma(x,x) \neq \sigma(y,y)$. Then $\mathcal{F}_\sigma^\ell$ cannot be simulated for any ℓ.

Proof: We will show that no function $\Delta^\ell : \mathcal{Z} \to \mathcal{Z}^{0+}$ can map $\mathcal{F}_\sigma^\ell$ to a distance that both admits a Hamming embedding and preserves the inequalities of Definition 4.1. Let $s_x = x^\ell$ and $s_y = y^\ell$ be ℓ-mers composed entirely of characters x and y respectively. If $\Delta^\ell(\mathcal{F}_\sigma^\ell(s_x, s_x)) \neq \Delta^\ell(\mathcal{F}_\sigma^\ell(s_y, s_y))$, then at least one of these distances is nonzero; hence, $\Delta^\ell \cdot \mathcal{F}_\sigma^\ell$ cannot be a metric on $\Sigma^\ell \times \Sigma^\ell$ and so cannot be preserved by any Hamming-space embedding of Σ^ℓ.

Suppose instead that $\Delta^\ell(\mathcal{F}_\sigma^\ell(s_x, s_x)) = \Delta^\ell(\mathcal{F}_\sigma^\ell(s_y, s_y)) = 0$, and suppose without loss of generality that $\mathcal{F}_\sigma^\ell(s_x, s_x) < \mathcal{F}_\sigma^\ell(s_y, s_y)$. Then, for $\theta = \mathcal{F}_\sigma^\ell(s_y, s_y)$, we have

$$\mathcal{F}_\sigma^\ell(s_x, s_x) < \theta$$

but

$$\begin{aligned} \Delta^\ell(\mathcal{F}_\sigma^\ell(s_x, s_x)) &= 0 \\ &\not> \delta(\theta). \end{aligned}$$

Hence, $\Delta^\ell \cdot \mathcal{F}_\sigma^\ell$ violates one of the inequality constraints of Definition 4.1. We conclude that *no* pair $\left\langle \Delta^\ell, \Phi^\ell \right\rangle$ simulates $\mathcal{F}_\sigma^\ell$. ∎

When $\sigma(x,x)$ is the same for all x, it is not clear whether σ can always be simulated. Not every finite metric space has an isometric embedding in $\{0,1\}^k$ for some k; in general, determining whether such an embedding exists is an NP-complete problem [27]. However, generalized Hamming spaces can isometrically embed metric spaces that ordinary Hamming

space cannot (e.g. three points mutually at distance one). We know of no result on the existence of generalized Hamming embeddings of arbitrary metric spaces. Furthermore, it is not clear whether every biologically meaningful score function with $\sigma(x, x)$ equal for all x can be mapped to a metric distance that preserves the inequality properties of Definition 4.1. Rather than try to address the above theoretical questions directly, the next section instead shows how to construct an embedding that, while not technically a simulation of σ, in practice achieves the desired end of allowing random projection to model arbitrary alignment score functions.

4.3.1 Extended Simulation

We can work around the failure of simulation for arbitrary score functions by removing the requirement that a score function be mapped to a metric distance on $\Sigma \times \Sigma$. The following construction, which we call *extended simulation*, shows how to remove this restriction without sacrificing the ability to embed the sequence alphabet isometrically in Hamming space.

We first expand the sequence alphabet Σ into two new alphabets Σ_1 and Σ_2 that contain two distinct copies, denoted x_1 and x_2, of every base $x \in \Sigma$. For example, the base alphabet Σ_b would expand into the alphabets

$$\begin{aligned} \Sigma_{b1} &= \{A_1, C_1, G_1, T_1\} \\ \Sigma_{b2} &= \{A_2, C_2, G_2, T_2\}. \end{aligned}$$

We use these expanded alphabets to break the symmetry of sequence comparison. A column score is now a function $\sigma : \Sigma_1 \times \Sigma_2 \to \mathcal{Z}$ on the cross product of two distinct alphabets (that happen to be written with similar symbols). The alignment score function $\mathcal{F}_\sigma^\ell$ on ℓ-mers is defined as before, except that it is now a function on $\Sigma_1^\ell \times \Sigma_2^\ell$.

For any column score function σ, define the following values:

$$\begin{aligned} W(\sigma) &= \max_{x \in \Sigma_1, y \in \Sigma_2} |\sigma(x, y)| \\ Z(\sigma) &= \left(|\Sigma|^2 - 1\right) W(\sigma) + \sum_{x \in \Sigma_1, y \in \Sigma_2} \sigma(x, y) \end{aligned}$$

Let $\Sigma_{12} = \Sigma_1 \cup \Sigma_2$, and define an *extended score function* $\psi[\sigma] : \Sigma_{12} \times \Sigma_{12} \to \mathcal{Z}$ as follows:

$$\psi(x,y) = \begin{cases} \sigma(x,y) & \text{if } x \in \Sigma_1, y \in \Sigma_2 \\ \sigma(y,x) & \text{if } x \in \Sigma_2, y \in \Sigma_1 \\ Z(\sigma) & \text{if } x = y \\ -W(\sigma) & \text{otherwise} \end{cases}$$

The extended function ψ coincides with σ on $\Sigma_1 \times \Sigma_2$, so a simulation of ψ would *a fortiori* preserve σ's score thresholds on those character pairs in $\Sigma_1 \times \Sigma_2$. This fact does not contradict Lemma 4.8 because σ is no longer defined for any pair (x,x) of identical characters, and the extended function ψ has $\psi(x,x)$ equal for all $x \in \Sigma_{12}$.

Theorem 4.9 Let σ be an arbitrary score function on $\Sigma_1 \times \Sigma_2$. Then $\psi[\sigma]$ can be simulated with an affine linear mapping and an embedding of dimension $Z(\sigma) + W(\sigma)$.

Proof: The proof is similar to that used for DNA-PAM-TT. Define $\delta(z) = Z(\sigma) - z$; as before, δ is affine linear and so preserves the score thresholds of ψ as required by Definition 4.1. It remains only to show an embedding of Σ_{12} that preserves the distance $\delta \cdot \psi$.

As before, we construct an isometry ϕ_0 into a weighted Hamming space and rely on Lemma 4.6 to complete the mapping to an unweighted space. ϕ_0 maps each $x \in \Sigma_{12}$ to a vector of length $|\Sigma|^2$ whose dimensions are labeled by pairs (a_1, b_2) of characters from $\Sigma_1 \times \Sigma_2$. The vector $\phi_0(x)$ is defined as follows:

$$\phi_0(x)[a_1, b_2] = \begin{cases} 0 & \text{if } x = a_1 \text{ or } x = b_2 \\ x & \text{otherwise} \end{cases}$$

Dimension (a_1, b_2) of the space has weight $W(\sigma) + \sigma(a_1, b_2)$, which is by construction nonnegative. Figure 4.4 illustrates this construction for a score function σ originally defined on the two-character alphabet $\{P, Q\}$.

To see that ϕ_0 is an isometry on Σ_{12} with distance $\delta \cdot \psi$, consider the following three cases:

- ϕ_0 is isometric on pairs (x,x) because

$$D(\phi_0(x), \phi_0(x)) \quad = \quad 0$$

	P	Q
P	3	-7
Q	-7	9

	(P_1,P_2)	(P_1,Q_2)	(Q_1,P_2)	(Q_1,Q_2)
P_1	[0,	0,	,	P_1
P_2	[0,	P_2,	0,	P_2]
Q_1	,	Q_1	0,	0]
Q_2	[Q_2,	0,	Q_2,	0]
Weight	12	2	2	18

Figure 4.4: an example of the embedding ϕ_0 for extended score simulation. The score function σ (at left) on the alphabet $\{P, Q\}$ is extended to form $\psi[\sigma]$ on the alphabet $\{P_1, Q_1, P_2, Q_2\}$, after which the weighted Hamming embedding (at right) is created as described in Theorem 4.9. The embedding preserves the distance $\delta \cdot \psi[\sigma]$, where $\delta(z) = Z(\sigma) - z = 25 - z$.

$$\begin{aligned} &= Z(\sigma) - Z(\sigma) \\ &= \delta(\psi(x,x)). \end{aligned}$$

- No dimension of a vector has a label containing two characters both from Σ_1 or both from Σ_2. Hence, for any $x, y \in \Sigma_1$ with $x \neq y$, no dimension of the vectors $\phi_0(x)$ and $\phi_0(y)$ contains the same character in both vectors, and so

$$\begin{aligned} D(\phi_0(x), \phi_0(y)) &= \sum_{z \in \Sigma_1, w \in \Sigma_2} (W(\sigma) + \sigma(z,w)) \\ &= |\Sigma|^2 W(\sigma) + \sum_{z \in \Sigma_1, w \in \Sigma_2} \sigma(z,w) \\ &= Z(\sigma) + W(\sigma) \\ &= \delta(-W(\sigma)) \\ &= \delta(\psi(x,y)). \end{aligned}$$

  A similar argument proves that ϕ_0 is an isometry on pairs $x, y \in \Sigma_2$.

- If $x \in \Sigma_1$ and $y \in \Sigma_2$, the vectors $\phi_0(x)$ and $\phi_0(y)$ coincide in the single dimension

with label (x, y), where both contain "0". Hence,

$$\begin{aligned} D(\phi_0(x), \phi_0(y)) &= \left[\sum_{z \in \Sigma_1, w \in \Sigma_2} (W(\sigma) + \sigma(z, w)) \right] - (W(\sigma) + \sigma(x, y)) \\ &= Z(\sigma) - \sigma(x, y) \\ &= \delta(\sigma(x, y)) \\ &= \delta(\psi(x, y)). \end{aligned}$$

The symmetry of ψ means that the embedding is also isometric on $(x, y) \in \Sigma_2 \times \Sigma_1$.

We conclude that ϕ_0 is indeed an isometry, and so there exists an isometric Hamming embedding of $\delta \cdot \psi$. The dimension of this embedding is, as before, the sum of the dimension weights defined for ϕ_0, or

$$\sum_{z \in \Sigma_1, w \in \Sigma_2} (W(\sigma) + \sigma(z, w))$$

which is equal to the claimed dimension $Z(\sigma) + W(\sigma)$. ∎

4.3.2 Implementing LSH-ALL-PAIRS *for Arbitrary Score Functions*

The extended score simulation of Theorem 4.9 may be implemented in LSH-ALL-PAIRS using the same ideas used to implement the simulation of DNA-PAM-TT. However, an implementation must accommodate not only the changes required to perform projection on the embedded representations of ℓ-mers but also the effects of breaking the symmetry of sequence comparison. In particular, it must address the fact that the character x may map to two distinct embedded representations, depending on whether it is interpreted as $x_1 \in \Sigma_1$ or $x_2 \in \Sigma_2$.

For alignments between orthologous sequences, such as the human-mouse comparisons of Chapter 2, the similarities of interest always align strings from two conceptually disjoint sets, each containing the ℓ-mers of one input sequence. LSH-ALL-PAIRS can therefore implement extended simulation for these analyses by treating the two input sequences as members of Σ_1^* and Σ_2^*, respectively. Provided the original score function σ on $\Sigma \times \Sigma$ is symmetric[1],

[1]This condition always holds in practical pairwise alignment, where neither sequence in a comparison is an ancestor of the other.

the choice of which sequence is assigned which alphabet is arbitrary. When projections are computed, each sequence's ℓ-mers are mapped to the embedded representations specific to its alphabet, and the projection values from both sequences are sorted together to produce candidate pairs. The modified version of LSH-ALL-PAIRS, like the original, must still discard candidate pairs in which both ℓ-mers are drawn from the same sequence.

For self-alignment problems such as our repeat finding experiments on human chromosome 22 and *Drosophila*, both ℓ-mers in a similarity are drawn from the same input sequence. However, treating the entire input as a string from Σ_1^* or Σ_2^* alone does not implement the desired simulation. For such comparisons, LSH-ALL-PAIRS must (at least conceptually) create two copies of its input, one from each of the alphabets Σ_1 and Σ_2. In practice, the filtering phase of the algorithm implements this duplication by mapping each ℓ-mer in the input to two distinct projection values – one for each alphabet. The checking phase, which needs only a single copy of the sequence, must of course take care to discard candidate pairs derived from a single ℓ-mer's two projection values.

Selecting the parameters m and k for LSH-ALL-PAIRS with extended score simulation also presents new difficulties. The false negative rate ρ_{fn} for any ℓ, m, and k and any threshold θ remains straightforward to compute:

$$\rho_{fn} = \left[1 - \left(1 - \frac{\Delta^\ell(\theta)}{(Z(\sigma) + W(\sigma))\ell}\right)^k\right]^m.$$

However, determining the false positive rate becomes more challenging. An analog of Equation (4.2) still applies:

$$\rho_{fp} = m \sum_{\theta' < \theta} \Pr\left[\mathcal{F}_\sigma^\ell(s_1, s_2) = \theta'\right] \left(1 - \frac{\Delta^\ell(\theta')}{(Z(\sigma) + W(\sigma))\ell}\right)^k \tag{4.4}$$

but the corresponding explicit computation of the false positive rate in terms of the numbers of each type of substitution (and, in this case, each type of match) is both complex and likely to be inefficient. This difficulty is actually not confined to extended score simulation: it applies to the simulation of any score function with complexity greater than that of simple functions like the DNA-PAM-TT family.

We previously chose not to explicitly evaluate the false positive rate in the form of Equation (4.4) because of the difficulty of computing $\Pr\left[\mathcal{F}_\sigma^\ell(s_1, s_2) = \theta'\right]$ for every $\theta' < \theta$.

However, it is possible to approximate these probabilities efficiently when the similarity length ℓ is large. Let P be the background character distribution; P induces a distribution F on the score $\sigma(x, y)$ of two characters chosen at random according to P, with mean μ_F and (to avoid confusion with the score function σ) variance v_F given by

$$\begin{aligned} \mu_F &= \sum_{x,y \in \Sigma} P(x)P(y)\sigma(x,y) \\ v_F &= \sum_{x,y \in \Sigma} P(x)P(y)\sigma(x,y)^2 - \mu_F^2. \end{aligned}$$

Recall that the background sequence model used to derive ρ_{fp} assumes that the background is composed of characters chosen independently at random. To compute the false positive rate, we must therefore estimate the probability that ℓ independent random character pairs have total score θ', or equivalently estimate the density of a sum of ℓ random variables, each with distribution F. When ℓ is large, the Central Limit Theorem implies that this sum is approximately normally distributed with mean $\ell\mu_F$ and variance ℓv_F.

For practical DNA sequence comparisons, significant similarities have lengths ℓ between 50 and 100 bases, large enough that the normal approximation of score probabilities is roughly accurate. For smaller lengths, as might be encountered in comparing protein sequences, the score distribution may instead be estimated empirically. Either approach permits explicit approximation of the sum in Equation (4.4) and hence an estimate of the true false positive rate.

4.4 Conclusions and Open Problems

In this chapter, we have shown that random projection, in particular the LSH-ALL-PAIRS algorithm, can be adapted to work with score functions more general than percent identity. We have introduced the idea of *score simulation* and constructed simulations for DNA-PAM and DNA-PAM-TT as well as an extended simulation that works for any ungapped alignment score function. Random projections drawn from sequences embedded according to these constructions are useful for more than just LSH-ALL-PAIRS; they provide a provably sensitive way to construct sparse indices allowing fast similarity-based queries against a database of DNA or protein sequences. In this form, score simulation enhances the utility of previous

work by Gionis et al. [38] for bioinformatics and provides a formal underpinning to the sparse sequence indexing schemes of Rigoutsos and Califano [78].

While we have not formally settled the question of whether simulations exist for arbitrary score functions with $\sigma(x, x)$ equal for all x, extended simulation at least enables such functions to be implemented in LSH-ALL-PAIRS. However, extended simulation requires an embedding of high dimensionality; for example, applying it to the DNA-PAM-TT-20 column score function yields an embedding ϕ with 180 dimensions per character, while the special-purpose embedding of Section 4.2.2 requires only 34 dimensions. A complete characterization of algorithmic cost versus simulation dimension remains a topic for future work, but we conjecture that higher-dimensional embeddings always entail a greater cost. The construction of Theorem 4.9 is quite sparse in the sense that, unless the values of σ are extremely skewed, two characters' embedded representations never match in more than a small fraction of their positions. Many projections are therefore required to ensure a reasonable probability of finding positions that *do* match, even in highly similar sequences.

If lower-dimensional projections are indeed more efficient, the following question becomes important:

Problem 4.2 (Efficient Simulation) Given a score function σ, find a simulation (possibly extended) for $\mathcal{F}_\sigma^\ell$ using an isometry into a Hamming space of the smallest possible dimension.

Finding a simulation of minimal dimension may be difficult in general, but the construction of Theorem 4.9 admits some obvious improvements that lower its dimension. Consider, for example, the modification shown in Figure 4.5. Let $x_1 \in \Sigma_1$ and $y_2, z_2 \in \Sigma_2$ be characters; let $\alpha = \sigma(x_1, y_2)$ and $\beta = \sigma(x_1, z_2)$, with $\alpha \leq \beta$. The weighted Hamming-space isometry ϕ_0 of the theorem contains two dimensions labeled (x_1, y_2) and (x_1, z_2), with respective weights $W(\sigma) + \alpha$ and $W(\sigma) + \beta$ and the contents shown at left for $\phi_0(x_1)$, $\phi_0(y_2)$, and $\phi_0(z_2)$.

Suppose we replace the dimension (x_1, y_2) by a new dimension $(x_1, y_2 + z_2)$ and adjust the weight of dimension (x_1, z_2) as shown at right in the figure. Call this modified weighted embedding ϕ_0', and let $\phi' = \phi_1 \cdot \phi_0'$, where ϕ_1 is the mapping from weighted to unweighted Hamming space defined in Lemma 4.6. Note that the total dimension of the new embedding

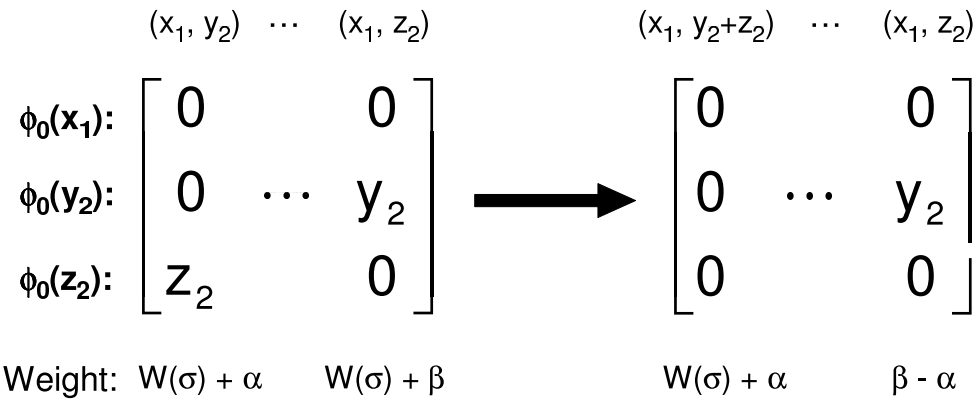

Figure 4.5: a technique for reducing the dimension in the extended simulation construction of Theorem 4.9. Two dimensions in the weighted Hamming embedding ϕ_0 of total weight $2W(\sigma) + \alpha + \beta$, with $\alpha \leq \beta$, are converted to two dimensions of total weight $W(\sigma) + \beta$, forming a new, lower-dimensional extended simulation that preserves the score function σ on $\Sigma_1 \times \Sigma_2$.

ϕ' is $Z(\sigma) - \alpha$, which is less than that of the original ϕ.

Lemma 4.10 There exist an extended score function $\psi'[\sigma]$ and an affine linear mapping δ' for which $\langle \delta', \phi' \rangle$ simulates ψ'. In particular, the new simulation still preserves σ on $\Sigma_1 \times \Sigma_2$.

Proof: Set $Z'(\sigma) = Z(\sigma) - W(\sigma) - \alpha$, so that the dimension of ϕ' is $Z'(\sigma) + W(\sigma)$. Define the extended score function $\psi'[\sigma]$ as follows:

$$\psi'(x,y) = \begin{cases} \sigma(x,y) & \text{if } x \in \Sigma_1, y \in \Sigma_2 \\ \sigma(y,x) & \text{if } x \in \Sigma_2, y \in \Sigma_1 \\ Z'(\sigma) & \text{if } x = y \\ \alpha & \text{for the pairs } (y_2, z_2) \text{ and } (z_2, y_2) \\ -W(\sigma) & \text{otherwise} \end{cases}$$

Note that ψ' preserves σ on $\Sigma_1 \times \Sigma_2$.

Define $\delta'(z) = Z'(\sigma) - z$. It is straightforward to check that for every $x, y \in \Sigma_{12}$,

$$H\left(\phi'(x), \phi'(y)\right) = \delta'(\psi'(x,y)).$$

The function δ' is affine linear and hence threshold-preserving, so the pair $\langle \delta', \phi' \rangle$ indeed simulates ψ'. ∎

Analogs of the dimension-reducing operation illustrated above exist for sets of three, four, and more dimensions, indeed for *any* set D of dimensions such that the positions denoted by the labels in D together form a combinatorial rectangle in σ. It remains an open problem to devise a general scheme of reductions to minimize the dimension of a given simulation; however, this task appears closely related to known NP-hard optimization problems such as finding the smallest biclique decomposition of a bipartite graph.

Another potential way to reduce the dimensionality of a simulation is to relax slightly the constraints of Definition 4.1, allowing an embedding that is not an isometry but distorts alignment scores by only a small amount. For example, if the precision of a score function σ is reduced by dividing each $\sigma(x, y)$ by 2^k for some $k \geq 1$, the dimension of the constructions for both DNA-PAM-TT and extended simulation are also reduced by a factor of 2^k. The resulting score function is equivalent to the original σ with its k low-order bits removed. Linial et al. provide additional results [61, Section 3] on the relationship between an embedding's distortion and its dimension that may be useful in devising efficient approximate simulations of arbitrary score functions.

BIBLIOGRAPHY

[1] Adams MD, Kelley JM, Gocayne JD, Dubnick M, Polymeropoulos M, Xiao H, Merril CR, et al. (1991). Complementary DNA sequencing: expressed sequence tags and human genome project. *Science* 252:1651–6.

[2] Adams MD et al. (2000). The genome sequence of *Drosophila melanogaster*. *Science* 287:2185–95.

[3] Akutsu T (1998). Hardness results on gapless local multiple sequence alignment. Technical Report 98-MPS-24-2, Information Processing Society of Japan.

[4] Altschul SF and Gish W (1996). Local alignment statistics. *Methods in Enzymology* 266:460–80.

[5] Altschul SF, Madden TL, Schäffer AA, Zhang J, Zhang Z, Miller W, and Lipman DJ (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research* 25(17):3389–402.

[6] Andersen RD, Taplitz SJ, Wong S, Bristol G, Larkin B, and Herschman HR (1987). Metal-dependent binding of a factor in vivo to the metal-responsive elements of the metallothionein 1 gene promoter. *Molecular and Cellular Biology* 7:3574–81.

[7] Arabidopsis Genome Initiative (2000). Analysis of the genome sequence of the flowering plant *Arabidopsis thaliana*. *Nature* 408:796–815.

[8] Arlazarov VL, Dinic EA, Kronrod MA, and Faradzev IA (1970). On economic construction of the transitive closure of a directed graph. *Soviet Mathematics Doklady* 11:1209–10. (English translation).

[9] Baeza-Yates RA and Navarro G (1999). Faster approximate string matching. *Algorithmica* 23:127–58.

[10] Baeza-Yates RA and Perleberg C (1992). Fast and practical approximate string matching. In *Proceedings of the 3rd Symposium on Combinatorial Pattern Matching*, pages 185–92.

[11] Bailey TL and Elkan C (1995). Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning* 21(1–2):51–80.

[12] Batzoglou S, Pachter L, Mesirov JP, Berger B, and Lander ES (2000). Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome Research* 10:950–8.

[13] Bedell JA, Korf I, and Gish W (2000). MaskerAid: a performance enhancement to RepeatMasker. *Bioinformatics* 16:1040–1.

[14] Blackwell TW, Rouchka E, and States DJ (1999). Identity by descent genome segmentation based on single nucleotide polymorphism distributions. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 54–59. AAAI Press, Heidelberg, Germany.

[15] Blanchette M (2001). Algorithms for phylogenetic footprinting. In *RECOMB01: Proceedings of the Fifth Annual International Conference on Computational Molecular Biology*, pages 49–58. Montreal, Canada.

[16] Blanchette M, Schwikowski B, and Tompa M (2000). An exact algorithm to identify motifs in orthologous sequences from multiple species. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 37–45. AAAI Press, San Diego, CA.

[17] Boam DSW, Clark AR, and Docherty K (1990). Positive and negative regulation of the human insulin gene by multiple trans-acting factors. *Journal of Biological Chemistry* 265:8285–96.

[18] Bonneau R, Tsai J, Ruczinski I, and Baker D (2001). Functional inferences from blind ab initio protein structure predictions. *Journal of Structural Biology* To appear.

[19] Bourgain J (1985). On Lipschitz embedding of finite metric spaces in Hilbert space. *Israeli Journal of Mathematics* 52:46–52.

[20] Boysen C, Simon MI, and Hood LE (1997). Analysis of the 1.1-Mb human alpha/delta T-cell receptor locus with bacterial artificial chromosome clones. *Genome Research* 7:330–8.

[21] Brāzma A, Jonassen I, Vilo J, and Ukkonen E (1998). Predicting gene regulatory elements *in silico* on a genomic scale. *Genome Research* 15:1202–1215.

[22] Buhler J and Tompa M (2001). Finding motifs using random projections. In *RECOMB01: Proceedings of the Fifth Annual International Conference on Computational Molecular Biology*. Montreal, Canada.

[23] Burge C and Karlin S (1997). Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology* 268:78–94.

[24] *C. elegans* Genome Sequencing Consortium (1998). Genome sequence of the nematode *C. elegans*: a platform for investigating biology. *Science* 282:2012–18.

[25] Chang WI and Lawler EL (1994). Sublinear expected time approximate string matching and biological applications. *Algorithmica* 12:327–44.

[26] Chen FC and Li WH (2001). Genomic divergences between humans and other hominoids and the effective population size of the common ancestor of humans and chimpanzees. *American Journal of Human Genetics* 68:444–56.

[27] Chvatal V (1980). Recognizing intersection patterns. In M Deza and IG Rosenberg, eds., *Annals of Discrete Mathematics - Combinatorics 79*, volume 8, pages 249–51. North Holland.

[28] Cover TM and Thomas JA (1991). *Elements of Information Theory*. John Wiley & Sons.

[29] Dayhoff MO, Schwartz R, and Orcutt BC (1978). A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure* 5:345–52.

[30] Delcher AL, Kasif S, Fleischmann RD, Peterson J, White O, and Salzberg SL (1999). Alignment of whole genomes. *Nucleic Acids Research* 27(11):2369–76.

[31] Dempster AP, Laird NM, and Rubin DB (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society* 39(1):1–38.

[32] Duda RO and Hart PE (1973). *Pattern Classification and Scene Analysis*. John Wiley & Sons.

[33] Dunham I, Shimuzu N, Roe BA, Chissoe S, et al. (1999). The DNA sequence of human chromosome 22. *Nature* 402:489–95.

[34] Fleischmann RD, Adams MD, White O, Clayton RA, Kirkness EF, Kerlavage AR, Bult CJ, et al. (1995). Whole-genome random sequencing and assembly of *Haemophilus influenzae rd. Science* 269:496–512.

[35] Fraser P and Grosveld F (1998). Locus control regions, chromatin activation, and transcription. *Current Opinion in Cell Biology* 10(3):361–5.

[36] Galas DJ, Eggert M, and Waterman MS (1985). Rigorous pattern-recognition methods for DNA sequences: Analysis of promoter sequences from Escherichia coli. *Journal of Molecular Biology* 186(1):117–128.

[37] Genome International Sequencing Consortium (2001). Initial sequencing and analysis of the human genome. *Nature* 409:860–921.

[38] Gionis A, Indyk P, and Motwani R (1999). Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Databases*. Edinburgh, Scotland.

[39] Gonick L and Wheelis M (1991). *The Cartoon Guide to Genetics*. Harper Perennial.

[40] Green P (1994). Phrap and Crossmatch. http://bozeman.mbt.washington.edu/phrap.docs/swat.html.

[41] Grossi R and Luccio F (1990). Simple and efficient string matching with k mismatches. *Information Processing Letters* 33:113–20.

[42] Gusfield D (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, New York, NY.

[43] Hattori M, Fujiyama A, Taylor TD, Watanabe H, et al. (2000). The DNA sequence of human chromosome 21. *Nature* 405:311–9.

[44] Hayes WS and Borodovsky M (1998). Deriving ribosomal binding site (RBS) statistical models from unannotated DNA sequences and the use of the RBS model for N-terminal prediction. In *Pacific Symposium on Biocomputing*, pages 279–290.

[45] Henikoff S and Henikoff JG (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Science USA* 89(22):10915–10919.

[46] Hertz GZ and Stormo GD (1995). Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps. In HA Lim and CR Cantor, eds., *Proceedings of the Third International Conference on Bioinformatics and Genome Research*, pages 201–216. World Scientific Publishing Co., Ltd., Singapore.

[47] Ho IC, Bhat NK, Gottschalk LR, Lindsten T, Thompson CB, Papas TS, and Leiden JM (1990). Sequence-specific binding of human ets-1 to the T cell receptor a gene enhancer. *Science* 250:814–8.

[48] Hogg RV and Craig AT (1970). *Introduction to Mathematical Statistics*. Macmillan, 3rd edition.

[49] Huang X and Madan A (1999). CAP3: a DNA sequence assembly program. *Genome Research* 9:868–77.

[50] Huang X and Miller W (1991). A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics* 12:337–57.

[51] Indyk P and Motwani R (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. Dallas, TX.

[52] Jang W, Hua A, Spilson SV, Miller W, Roe BA, and Meisler MH (1999). Comparative sequence of human and mouse BAC clones from the mnd2 region of chromosome 2p13. *Genome Research* 9:53–61.

[53] Johnson W and Lindenstrauss J (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics* 26:189–206.

[54] Karlin S and Altschul SF (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Science USA* 87:2264–8.

[55] Koop BF and Hood L (1994). Striking sequence similarity over almost 100 kilobases of human and mouse T-cell receptor DNA. *Nature Genetics* 7:48–53.

[56] Kozak M (1983). Comparison of initiation of protein synthesis in procaryotes, eucaryotes, and organelles. *Microbiological Reviews* 47(1):1–45.

[57] Lawrence CE, Altschul SF, Boguski MS, Liu JS, Neuwald AF, and Wootton JC (1993). Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science* 262:208–214.

[58] Lawrence CE and Reilly AA (1990). An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins: Structure, Function, and Genetics* 7:41–51.

[59] Lee IY, Westaway D, Smit AF, Wang K, Seto J, Chen L, Acharya C, et al. (1998). Complete genomic sequence and analysis of the prion protein gene region from three mammalian species. *Genome Research* 8:1022–37.

[60] Lewin B (1997). *Genes VI*. Oxford University Press.

[61] Linial N, London E, and Rabinovich Y (1994). The geometry of graphs and some of its algorithmic applications. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 577–91. Los Alamitos, CA.

[62] McCue L, Thompson W, Carmack C, Ryan MP, Liu JS, Derbyshire V, and Lawrence CE (2001). Phylogenetic footprinting of transcription factor binding sites in proteobacterial genomes. *Nucleic Acids Research* 29:774–82.

[63] McInerny CJ, Partridge JF, Mikesell GE, Creemer DP, and Breeden LL (1997). A novel Mcm1-dependent element in the SWI4, CLN3, CDC6, and CDC47 promoters activates M/G_1-specific transcription. *Genes & Development* 11:1277–1288.

[64] Means AL and Farnham PG (1990). Transcription initiation from the dihydrofolate reductase promoter is positioned by *hip1* binding at the initiation site. *Molecular and Cellular Biology* 10:653–61.

[65] Mewes HW, Albermann K, Bahr M, Frishman D, Gleissner A, Hani J, Heumann K, et al. (1997). Overview of the yeast genome. *Nature* 387(6632 suppl.):7–65.

[66] Muthukrishnan S and Ṣahinalp SC (2000). Approximate nearest neighbors and sequence comparison with block operations. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 416–24.

[67] Myers EW (1994). A sublinear algorithm for approximate keyword searching. *Algorithmica* 12:345–74.

[68] Natsan S and Gilman M (1995). Yy1 facilitates the association of serum response factor with the c-fos serum response element. *Molecular and Cellular Biology* 15:5975–82.

[69] Needleman SB and Wunsch CD (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48:443–53.

[70] Neuwald AF and Green P (1994). Detecting patterns in protein sequences. *Journal of Molecular Biology* 239:698–712.

[71] NIST Engineering Statistics Handbook. http://www.itl.nist.gov/div898/handbook/.

[72] Oeltjen JC, Malley TM, Muzny DM, Miller W, Gibbs RA, and Belmont JW (1997). Large-scale comparative sequence analysis of the human and murine Bruton's tyrosine kinase loci reveals conserved regulatory domains. *Genome Research* 7:315–29.

[73] Pachter L, Alexandersson M, and Cawley S (2001). Applications of generalized pair hidden Markov models to alignment and gene finding problems. In *RECOMB01: Proceedings of the Fifth Annual International Conference on Computational Molecular Biology*. Montreal, Canada.

[74] Paracel GeneMatcher. http://www.paracel.com/products/genematcher.html.

[75] Pearson WR and Lipman DJ (1988). Improved tools for biological sequence analysis. *Proceedings of the National Academy of Science USA* 85:2444–8.

[76] Pevzner P and Sze SH (2000). Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 269–278. San Diego, CA.

[77] Pevzner P and Waterman MS (1995). Multiple filtration and approximate pattern matching. *Algorithmica* 13:135–54.

[78] Rigoutsos I and Califano A (1993). FLASH: a fast look-up algorithm for string homology. In *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*, pages 56–64.

[79] Rigoutsos I and Floratos A (1998). Motif discovery without alignment or enumeration. In *RECOMB98: Proceedings of the Second Annual International Conference on Computational Molecular Biology*, pages 221–227. New York, NY.

[80] Rivas E and Eddy SR (2000). Secondary structure alone is generally not statistically significant for the detection of noncoding RNAs. *Bioinformatics* 16:573–85.

[81] Rocke E and Tompa M (1998). An algorithm for finding novel gapped motifs in DNA sequences. In *RECOMB98: Proceedings of the Second Annual International Conference on Computational Molecular Biology*, pages 228–233. New York, NY.

[82] Sagot MF (1998). Spelling approximate repeated or common motifs using a suffix tree. In CL Lucchesi and AV Moura, eds., *Latin '98: Theoretical Informatics*, volume 1380 of *Lecture Notes in Computer Science*, pages 111–127. Springer.

[83] Sagot MF and Myers EW (1998). Identifying satellites in nucleic acid sequences. In *RECOMB98: Proceedings of the Second Annual International Conference on Computational Molecular Biology*. New York, NY.

[84] Sanger F, Coulson AR, Hong GF, Hill DF, and Petersen GB (1982). Nucleotide sequence of bacteriophage lambda DNA. *Journal of Molecular Biology* 162(4):729–73.

[85] Schwartz S, Zhang Z, Frazer KA, Smit AF, Riemer C, Bouck J, Gibbs RA, et al. (2000). PipMaker – a web server for aligning two genomic DNA sequences. *Genome Research* 10:577–86.

[86] Sheridan SD, Benham CJ, and Hatfield GW (1998). Activation of gene expression by a novel DNA structural transmission mechanism that requires supercoiling-induced DNA duplex destabilization in an upstream activating sequence. *Journal of Biological Chemistry* 273:21298–308.

[87] Sheridan SD, Benham CJ, and Hatfield GW (1999). Inhibition of DNA supercoiling-dependent transcriptional activation by a distant B-DNA to Z-DNA transition. *Journal of Biological Chemistry* 274:8169–74.

[88] Shpaer EG, Robison M, Yee D, Candlin JD, Mines R, and Hunkapiller T (1996). Sensitivity and selectivity in protein similarity searches: Comparison of Smith-Waterman in hardware. *Genomics* 38:179–91.

[89] Sinha S and Tompa M (2000). A statistical method for finding transcription factor binding sites. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 344–354. AAAI Press, San Diego, CA.

[90] Smit AF (1996). The origin of interspersed repeats in the human genome. *Current Opinion in Genetics and Development* 6(6):743–8.

[91] Smit AF (1999). Interspersed repeats and other mementos of transposable elements in mammalian genomes. *Current Opinion in Genetics and Development* 9(6):657–63.

[92] Smit AF and Green P (1999). Repeatmasker. http://ftp.genome.washington.edu/RM/RepeatMasker.html.

[93] Smit AF and Riggs AD (1995). MIRs are classic, tRNA-derived SINEs that amplified before the mammalian radiation. *Nucleic Acids Research* 23(1):98–102.

[94] Smith TF and Waterman MS (1981). Identification of common molecular subsequences. *Journal of Molecular Biology* 147(1):195–97.

[95] Staden R (1989). Methods for discovering novel motifs in nucleic acid sequences. *Computer Applications in the Biosciences* 5(4):293–298.

[96] States DJ, Gish W, and Altschul SF (1991). Improved sensitivity of nucleic acid database searches using application-specific scoring matrices. *Methods: a Companion to Methods in Enzymology* 3(1):66–70.

[97] Strachan T and Read AP (1996). *Human Molecular Genetics*. Books International, Inc., Bios Scientific.

[98] Swoboda P, Adler HT, and Thomas JH (2000). The RFX-type transcription factor DAF-19 regulates sensory neuron cilium formation in *C. elegans*. *Molecular Cell* 5:411–21.

[99] Tompa M (1999). An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 262–271. AAAI Press, Heidelberg, Germany.

[100] van Helden J, André B, and Collado-Vides J (1998). Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology* 281(5):827–842.

[101] Venter JC et al. (2001). The sequence of the human genome. *Science* 291:1304–51.

[102] Wang L and Jiang T (1994). On the complexity of multiple sequence alignment. *Journal of Computational Biology* 1:337–348.

[103] Wilson C, Hilyer L, and Green P (1994). Genefinder. http://repeatmasker.genome.washington.edu/cgi-bin/GeneFinder.

[104] Wilson RK, Lai E, Concannon P, Barth RK, and Hood LE (1988). Structure, organization, and polymorphism of murine and human T-cell receptor α and β chain gene families. *Immunological Reviews* 101:149–72.

[105] Wingender E, Dietze P, Karas H, and Knüppel R (1996). TRANSFAC: a database on transcription factors and their DNA binding sites. *Nucleic Acids Research* 24(1):238–241. http://transfac.gbf-braunschweig.de/TRANSFAC/.

[106] Yuh CH, Bolouri H, and Davidson EH (1998). Genomic cis-regulatory logic: experimental and computational analysis of a sea urchin gene. *Science* 279:1871–2.

[107] Zhu J, Liu JS, and Lawrence CE (1998). Bayesian adaptive sequence alignment algorithms. *Bioinformatics* 14:25–39.

Appendix A

THE CHANG-LAWLER EXCLUSION ALGORITHM

Chang and Lawler [25] devised a filtering scheme for the corpus-pattern matching problem (Problem 2.2) using matching statistics. Let C be a corpus, and let P be a fixed pattern string. Define the *ith matching statistic* $\mu(i)$ of C versus P to be the length of the longest exact prefix match between any substring of P and the ith suffix of C, i.e. $C[i \ldots |C|]$. After preprocessing P, the matching statistics of C versus P can be computed in time $O(|C|)$ using a suffix tree [42, Section 7.8].

We describe the ungapped version of Chang-Lawler, following Gusfield's presentation in [42, Chapter 12]. Let ℓ be the pattern length, and d be the maximum number of allowed mismatches between corpus and pattern. The algorithm first divides the corpus into nonoverlapping regions of size $\ell/2$, then attempts to eliminate all regions that cannot be part of a d-mismatch to the pattern. Regions that are not eliminated become candidates for checking.

Let $r = C[j \ldots j']$ be a region of the corpus, with $j' = j + \ell/2 - 1$. If the following test fails, r is eliminated; otherwise, it becomes a candidate:

```
TEST-CANDIDACY(r)
    posn ← j
    repeat d + 1 times
        posn ← posn + μ(posn) + 1
        if posn > j'
            return success
        endif
    end

    return failure
```

Lemma A.1 Suppose $r = C[j \ldots j']$ is part of a d-mismatch to the pattern. Then TEST-CANDIDACY(r) will succeed.

Proof: If r is part of a d-mismatch to P, then it matches some substring π of P with at most d mismatches. Hence, the alignment of r with π can be divided into at most $d+1$ word matches (some possibly of zero length), each separated by a single mismatched pair. These words have total length at least $\ell/2 - d$.

Claim A.1A Consider the operation of TEST-CANDIDACY on r. After t iterations of the loop, we claim that the *posn* pointer must have passed both the first t words of the alignment and the tth mismatched pair.

Proof: Proceed by induction on t. The base case $t = 0$ is trivial. At the start of the $t+1$st iteration, we know inductively that *posn* points either to a character of the $t+1$st word or to some character after that word. In the latter case, we always add at least one to *posn*, so the claim holds. In the former case, suppose that *posn* points to $r[i]$, and that there are k characters between *posn* and the end of the word. Each word in r has a corresponding match in π, so by definition $\mu(j+i)$ must be at least k. Hence, the $t+1$st increment will be at least $k+1$, and the pointer will be advanced past the end of the word and following mismatch. ∎

It follows from the claim that after at most $d+1$ iterations of the loop, *posn* must have advanced by the total length of all words plus d intervening mismatched pairs and one terminal pair, for a total distance of at least $\ell/2 + 1$. Since *posn* is initialized to j, this distance puts it past j', and so the test succeeds. ∎

Intuitively, Chang-Lawler uses matching statistics as a rough substitute for actually finding the consecutive word matches that constitute an ungapped match between C and P. The test is only approximate because the words in P that give rise to the matching statistics need not be consecutive or within a common interval of size $\ell/2$. However, the filter is efficient because the expected size of a matching statistic between the pattern and an i.i.d. uniform sequence is small, only $\log(\ell)$. For small values of d, it is unlikely that adding up the lengths of at most d consecutive word matches between a region and any part of P would produce a total length as long as $\ell/2$.

As discussed in Section 2.2.3, the naive extension of Chang-Lawler to the all-pairs problem requires $|C|$ passes over the corpus. A more clever extension would compare each region against a string longer than one pattern length. For instance, we could first divide the input into m nonoverlapping "chunks," then perform a single pass of Chang-Lawler per chunk, using the entire chunk to compute matching statistics[1]. The number of passes would be reduced at the cost of more regions spuriously passing the filter.

[1]In the full all-pairs formulation, the region itself would have to be excluded from the matching statistics.

Appendix B

THE DOUBLE FILTRATION ALGORITHM

Pevzner and Waterman's double filtration algorithm [77] extends the idea of word match filtering to obtain a more efficient strategy. Define a *gapped k-tuple with gapsize t* to be a series of k equally spaced indices with a distance t between successive indices. For example, a gapped 3-tuple with gapsize 2 has indices $\{j, j+2, j+4\}$. We say that a *gapped tuple match* exists between two strings if they share a gapped tuple whose positions contain only matching characters (an exact word match of length k is thus a k-tuple match with gapsize 1).

Double filtration requires that every candidate for checking contain both a word match and a gapped tuple match of sufficient length. Adding the tuple match requirement makes the filter more stringent than word matching alone and so improves efficiency. The following theorem shows that adding the tuple match criterion does not affect the correctness of the algorithm – it still finds all ungapped similarities with sufficiently few substitutions:

Theorem B.1 Every pair of ℓ-mers that match to within d substitutions contain both a word match of length $k = \lfloor \ell/(d+1) \rfloor$ and a gapped k-tuple match with gapsize $d+1$ on the same diagonal. Furthermore, if x and y are the starting indices of the word match and tuple match respectively, then $-d \leq x - y \leq \ell - k$.

Proof: We first give an alternate counting argument that proves Lemma 1.1, then give the corresponding proof for the existence of a gapped k-tuple match.

Let s_1 and s_2 be ℓ-mers that match to within d substitutions. s_1 contains at least $d+1$ nonoverlapping k-mers, each of which aligns to some region of s_2 with zero or more mismatches. But there are only d total mismatches between the ℓ-mers, so some k-mer in s_1 must align to s_2 with no mismatch. This proves the existence of a k-mer word match.

A gapped k-tuple spans $(k-1)(d+1)+1$ sequence positions; hence, a sequence of length

ℓ can hold at least $d+1$ such tuples starting at positions $1, 2, \ldots d+1$. None of these tuples share any position between them. Now consider the $d+1$ pairs of gapped k-tuples starting at positions $1, 2, \ldots d+1$ in both strings. Again, these tuple pairs cover disjoint pairs of characters. s_1 and s_2 differ by at most d substitutions; hence, at most d of the tuple pairs contain a mismatch. We conclude that at least one k-tuple pair contains only matches, which is what we want.

It remains to show that the word and tuple matches between s_1 and s_2 are within the claimed distance bound. Let x be the starting position (common to s_1 and s_2) of the exact match, and let y be the starting position of the tuple match. In proving the existence of the word match, we showed that

$$1 \leq x \leq \ell - k + 1.$$

The corresponding proof for the tuple match showed that

$$1 \leq y \leq d + 1.$$

We therefore conclude that

$$-d \leq x - y \leq \ell - k$$

which was the original claim. ∎

Appendix C

EM FOR THE WEIGHT MATRIX MOTIF MODEL

Lawrence and Reilly [58] describe an expectation maximization (EM) formulation for finding maximum-likelihood motifs in a weight matrix model. The following is a simplified version of that formulation as used in our implementation of the PROJECTION algorithm.

Let $C = \{c_1 \ldots c_t\}$ be a collection of t sequences of common length n containing a motif of length ℓ. The motif is assumed to occur exactly once in each c_i, with the remaining bases of C distributed independently at random according to a background distribution P. The motif's occurrences are generated according to a WMM W, which we must estimate from the sequences C. For simplicity, we assume that the background is large compared to the motif, so that P can be inferred with minimal error by setting the probability $P[b]$ to the frequency of base b in *all* of C. This approximation fixes P during the EM algorithm, whereas the original formulation simultaneously estimates W and P.

Our goal is to find the WMM W that maximizes the likelihood

$$L\left[W \mid C, P\right] = \Pr\left[C \mid W, P\right]$$

(Henceforth, we will leave P implicit in the likelihood rather than writing it each time.) If the locations of the motif's occurrences were fixed *a priori*, a maximum-likelihood estimate of W could trivially be computed as described in Section 3.1.1. However, the problem is complicated by the fact that the motif occurs at an *unknown* position in each c_i. To compute the likelihood of any given W, we must therefore sum over all possible locations of the motif. Let x_{ij} be a 0-1 indicator which is one iff the motif occurs in c_i starting at position j. Then

$$L\left[W \mid C\right] = \sum_{j_1 \ldots j_t} \Pr\left[c_i \mid W, x_{1,j_1} = 1, \ldots, x_{t,j_t} = 1\right]$$

where the x_{ij}'s not written explicitly in each term are all zero.

Simply computing the likelihood for a single W requires an expensive summation, so it is not clear how to efficiently and stably *maximize* W. Fortunately, the EM algorithm, due to Dempster, Laird, and Rubin [31], is a relatively inexpensive and stable procedure for finding the model W^* that locally maximizes the likelihood in the vicinity of an initial guess W_1. Briefly, the algorithm alternates *E-steps*, which compute the expected values $\hat{x}_{ij}$ of the unknown motif positions x_{ij}, with *M-steps*, which re-estimate the parameters of the model W using the $\hat{x}_{ij}$'s. It can be shown that iterating alternate E-steps and M-steps causes the estimates of W to converge linearly to the locally maximum-likelihood estimate W^*.

Below, we detail the computations of the E-step and M-step in PROJECTION's version of EM for motifs.

C.1 E-Step

In the E-step, we compute $\hat{x}_{ij} = E[x_{ij} \mid C, W]$ for each sequence c_i and starting position j. Using the fact that x_{ij} is 0-1 and applying Bayes' theorem, we have that

$$\begin{aligned} E[x_{ij} \mid C, W] &= \Pr[x_{ij} = 1 \mid c_i, W] \\ &= \frac{\Pr[c_i \mid x_{ij} = 1, W] \Pr[x_{ij} = 1 \mid W]}{\Pr[c_i \mid W]} \end{aligned}$$

If we place equal prior weight on the motif occurring at any position in sequence c_i and recognize that, by the assumptions of the model,

$$\sum_j \hat{x}_{ij} = 1, \tag{C.1}$$

then computing the desired expectations for x_{i*} requires only that we first compute $\Pr[c_i \mid x_{ij} = 1, W]$ for $1 \leq j \leq n - \ell + 1$, then normalize these values to sum to one over all j.

Let $s_{ij} = c_i[j \ldots j + \ell - 1]$ be the putative motif occurrence starting at position j in c_i. Then we have that

$$\Pr[c_i \mid x_{ij} = 1, W] = \Pr[c_i[1 \ldots j-1] \mid P] \Pr[s_{ij} \mid W] \Pr[c_i[j + \ell \ldots n] \mid P]$$

Removing a constant factor of $\Pr[c_i \mid P]$, we are left with

$$\hat{x}_{ij} \propto \frac{\Pr[s_{ij} \mid W]}{\Pr[s_{ij} \mid P]},$$

with the normalization of Equation (C.1) yielding the correct expectations.

C.2 M-Step

In the M-step, we re-estimate the parameters of the model W using the $\hat{x}_{ij}$'s. Exploiting the fact that the x_{ij}'s are 0-1, we may write the complete-data likelihood as

$$L[W \mid C, x_{**}] = \prod_i \prod_j (L[W \mid c_i, x_{ij} = 1])^{x_{ij}}$$

Taking logs and rearranging terms, it follows that

$$\log L[W \mid C, x_{**}] = \sum_{i,j} \sum_{y=0}^{\ell-1} x_{ij} \log W[s_{ij}[y], y] + Z \tag{C.2}$$

where Z is a constant term independent of W. By monotonicity of the log, the same W^* maximizes both the first and second expressions.

EM re-estimates W by substituting the expectation $\hat{x}_{ij}$ for each x_{ij} on the right-hand side of Equation (C.2) and maximizing the resulting expression with respect to the probabilities $W[b, y]$. It can be shown that this expression is maximized when

$$W[b, y] = \frac{\sum_{i,j} \hat{x}_{ij} \delta(s_{ij}[y], b)}{\sum_{i,j} \hat{x}_{ij}} \tag{C.3}$$

where $\delta(b', b)$ is one if $b = b'$ and zero otherwise. The new estimate of W derived by Equation (C.3) becomes the input to the next E-step.

Intuitively, the estimate of W given in Equation (C.3) is the same as that of Section 3.1.1. The only difference is that, rather than estimating W from a motif with exactly t ℓ-mers, we instead let *every* ℓ-mer in the input contribute to the estimate according to its weight $\hat{x}_{ij}$. Note that by Equation (C.1), the denominator of (C.3) is exactly t, so the total weight of occurrences used is the same whether or not the occurrences are fixed. If the nearest local maximum to the starting guess is the the true motif, a few iterations of EM should accumulate most of the weight on its true occurrences, so that the estimate of W rapidly converges to the W^* that would be computed from the true motif alone.

VITA

Jeremy Buhler did his undergraduate work in computer science at Rice University in Houston, Texas (B.A. 1996) and his graduate work, also in computer science, at the University of Washington in Seattle, Washington (M.S. 1998, Ph.D. 2001). He is presently an assistant professor in the Department of Computer Science at Washington University in St. Louis, Missouri.

Dr. Buhler became interested in computational biology as an undergraduate, working with Dr. Alejandro Schäffer on CASPAR, a program for affected sibpair analysis of complex diseases. After moving to Seattle, he worked for some time on data quality assurance in fluorescent cDNA microarrays, producing the Dapple software for microarray quantitation. Most recently, he has studied problems of sequence comparison and similarity search.

As a graduate student, Dr. Buhler was supervised by Drs. Richard Karp and Martin Tompa, and occasionally by his cat Figaro.