AN EUCLIDEAN METRIC FOR GENETIC SEQUENCE COMPARISON

K. Balasubramanian
(Thesis)

CS-TR-332-91

October 1991

An Euclidean Metric for Genetic Sequence Comparison

K. Balasubramanian

A Dissertation

Presented to the Faculty of

Princeton University

in Candidacy for the Degree of

Doctor of Philosophy

Recommended for Acceptence

by the Department of

Computer Science

October 1991

Acknowledgements[†]

I would like to express my gratitude to my advisor, Prof. Richard Lipton, for his ideas and encouragement teaching me the importance of taking the broad view, to Prof. Douglas Welsh for providing a molecular biologists perspective and to Prof. Ken Steiglitz for reading this thesis and his helpful comments on improving its clarity.

I would also like to express my appreciation the technical and office staff, and Sharon Rodgers in particular, for all their support and help during my stay at Princeton.

My thanks too to all my collegues and friends at Princeton, for making my time there the memorable and broadening experience that it turned out to be. It would be impossible to acknowledge all the wonderful friends I made but I must at least try.

My thanks to Arvin Park for a fruitful partnership, for his advice and encouragement, for golf lessons but most of all for being a friend.

To S. V. Krishnan, Mohanram Sivaraja, A. Hariharan, Ehtesham Hyder, for making a house a home.

To A. Narayanan and Jyoti Shukla, for so many reasons.

To Chris Clifton, Alvaro Campos, Burt Rosenberg, Mark Greenstreet, Matt Blumrich, Jim Plank, Karin Petersen, Csaba Gabor, Bill Lin, Mike Laszlo, Luen Heng, E. S. Panduranga and Rob Abbott, my officemates over the years who made coming to work a distinct pleasure.

[†]This work was partially supported by NSF Grant #COOP AGMT DCR 8420948 and DARPA Grant #N00014 - 85 - C - 456 P00005.

To the Cache Hitters.

My apologies to anyone whose names I may have omitted. While they may be absent from this page, they will remain in my memory.

My affectionate gratitude to my family both here in the United States and back home in India for their love and understanding. Most of all to my parents for always encouraging me to think for myself and to my sister. It is to them I dedicate this work.

Abstract

This thesis introduces a new representation for genetic sequences in the form of geometric points or vectors. It is based on the relative frequencies of the various (small) fixed length substrings or *k-tuples* of the DNA or protein sequences. This effectively transforms the sequence from a variable sized string to fixed size vector. We show that this transformation preserves, under certain circumstances, the *edit distance* between sequences, a widely used measure for comparing genetic sequences. This fact is used to develop a linear time heuristic for sequence comparison, an improvement over the quadratic time dynamic programming based algorithms currently in widespread use.

The transformation from a variable sized sequence to a fixed size vector representation allows computational geometric techniques to be applied to the study of genetic sequences. In particular, we develop a method of comparing several sequences simultaneously without having to compare each pair of sequences separately. This results in a substantial reduction in the complexity of the problem of multiple sequence comparison and clustering. This can be applied as a filter to extract interesting sets of sequences from a large database for more thorough study as well as an indexing method, to locate the database sequences most likely to be related to a query sequence.

Table of Contents

I. Introduction

This thesis deals with certain computational problems arising in the context of Molecular Biology. The field of molecular biology has seen an increasing number of computer applications in recent years. Most such applications center around the *Genetic Sequence* data object. Genetic sequences refer to sequences of *Nucleic Acids*, which form the genetic material and are responsible for transmitting information from generation to generation, or to sequences of *Amino Acids*, which are the building blocks of proteins and which are responsible for *Gene Expression*. In particular the type of problems we shall be particularly be concerned with in this thesis involve comparison of genetic sequences - locating sequences which are similar or quantifying the similarity between two sequences.

This thesis introduces a new representation for genetic sequences in the form of geometric points or vectors. This representation results in a linear (in sequence length) time heuristic for sequence comparison, an improvement over the quadratic time dynamic programming algorithms currently in widespread use. It is based on the relative frequencies of the various (small) fixed length substrings or *k-tuples* of the DNA or protein sequences. Moving away from a variable sized sequence to a fixed size vector representation allows computational geometric techniques to be applied to the study of genetic sequences. In particular, we develop a method of comparing several sequences simultaneously without having to compare each pair of sequences separately. This results in a substantial reduction in the complexity of the problem of multiple sequence

comparison and clustering.

Sequence as Data Object

Handling of sequences or strings is a very wide ranging problem occurring in many diverse areas of computer applications. This is because in a large number of computational activities the data objects in question cannot be represented as simple numerical values or structures but only as strings or variable length linear sequences of characters from a given alphabet. This is obvious when dealing with actual English (or natural language) text, as in the case of text editing or word processing, but may also be implicitly true in many other cases as well. The computational problems, and the type of queries involved with respect to sequences are many and varied. The simplest string related activity involves only storage and retrieval. We wish to be able to store a string and retrieve it on demand without regard to its content. An example of this would be a "synopsis" or "abstract" field in a bibliographic database. The system need only be able to associate a particular string (the text of the abstract) with a particular record (the book): the actual contents are of no consequence.

A common type of computational operation with respect to sequences involves checking for identicality of two strings, *ie.*, whether they match character for character. Often we are interested in comparing one string with a *substring* of another as opposed to the entire string. This would include text indexing or searching, where we are interested in locating where a particular "target" string may be found in a body of "source" text. This is widely used in text editors, word processors, searching for entries in a dictionary, searching for quotations in text, looking up telephone or other directory entries etc., and is often an integral part of many database

applications. The type of queries involved may also be distinguished according to whether the source is fixed (as in a telephone directory), or varied (as in different pieces of text) and whether the target we are searching for is a fixed string or one of a set of possible strings.

Numerous algorithms exist for this type of search ranging from the brute force method wherein the target is slid along the source string one character at a time and tested for match at each position, to more refined methods. For the case where the target is a fixed string Knuth, Morris and Pratt [8] proposed a refinement, involving preprocessing the target string, allowing the target to be moved several positions along the source at once when a mismatch occurs. This is based on determining the minimum possible number of moves during which there can be no match, knowing how how much of target string matched the source at the current position and what the character that caused the mismatch was. This was further extended by Boyer and Moore [1]. Rabin and Karp proposed an algorithm based on hashing the target sequence as well as all possible substrings of the source of the same length as the target. For the case where the target is a set of strings represented by a regular expression Thompson [23] developed a method based on converting the regular expression to a deterministic finite automaton. Another algorithm, due to Morrison [17] is based on constructing a modified B-tree, known as a "patricia" tree and is useful in the case where the source text is fixed. This has been applied to create an index for the entire Oxford English Dictionary to search for words or word entries in the dictionary. Inverted indices [12] are also used in a variety of bibliographic systems such as *refer*.

Another type of sequence related problem is finding out how closely two sequences *resemble* one another, without insisting on syntactic equality. In this case we need to be able to define *closeness* of sequences in order to be able to make meaningful comparisons such as "which of these two sequences is a given sequence closer to". This is obviously more difficult than searching for identical (sub)sequences, wherein we are essentially dividing up the set of (sub)sequences into a set of equivalence classes. These types of problems are very important in the context of Molecular Biology and we shall examine them in greater detail in chapter 3.

A note on terminology

It must be noted that for the most part the terms *sequence* and *string* are used interchangeably. However, in places where the distinction is important, the term *substring* will always refer to characters that are contiguous in the original string, whereas the term *subsequence* refers to a sequence of characters that appear in the same order as in the original sequence, although not necessarily contiguously. Thus for example the text strings "*wert*" and "*quip*" are both subsequences of the string "*qwertyuiop*" whereas only the former is a substring. Also a substring that begins at the first position of the parent string shall be referred to as a *prefix* of the string. Thus "*q*", "*qwert*" and "*qwertyuiop*" would all be prefixes of the above string. A prefix that is shorter than the parent string is a *proper* prefix.

Organization of Thesis

The next chapter looks at some background information concerning the nature of genetic sequences and familiarizes the reader with the domain which we shall be considering. Chapter III looks at the existing concepts of sequence homology and various existing algorithms. Chapter IV presents a brief theoretical overview of the concept of sequence mutation and presents a mathematical model of the process which will be necessary to justify future computer simulation experiments. Chapter V introduces the vector distance metric of sequence similarity, establishes its validity, and demonstrates its applicability to speeding up the process of sequence comparison. Chapter VI presents extensions arising from the notion of sequence vectors that lead to fast indexing algorithms for genetic sequences. Chapter VII presents the conclusions of this thesis.

II. Genetic Sequences

The Nature of the Genetic Material

Genes are the units of inheritance by which a living organism passes on its characteristics to the next generation. While this genetic theory of inheritance has its origins in the experiments of Mendel in the 19th century, the physical and biochemical nature of the gene was not understood till this century, with the discovery of chromosomes in the nucleus of the cell and the realization that genes reside on these chromosomes. Chromosomes are composed of twin complementary strands of *Deoxyribo Nucleic Acid* or *DNA*. This DNA is the genetic material which is the physical carrier of information from generation to generation as well as from cell to cell.

DNA is a polymer consisting of a chain of *nucleotides* connected together by sugar-phosphate links which form a backbone. There are four possible nucleotides, *Adenine (***A***) and Guanine (***G***), together known as the purines*, and Cytosine (**C***) and Thiamine (***T***)*, together known as the *pyrimidines*. These nucleotides are also sometimes referred to as *bases*[†]. Adenine and Thiamine are *complementary* bases as are Cytosine and Guanine. Complementary refers to the fact that they are capable of bonding with each other by means of *hydrogen bonds*. In some cases the base *Uracil* occurs in place of Thiamine but the number of bases and the complementarity is preserved. The *duplex*, or *double helix* structure of DNA

[†]The terms nucleotide, base and nucleic acid will be understood to be identical in this work. Although there is a chemical

consists of two complementary strands, *ie.,* each base in one strand is attached to its complementary base in the other. This implies that the sequence of bases in one strand fully determines the sequence in the other. This is what allows DNA to self replicate, since during the cell division process, the two strands separate and a mating strand is created for each from a pool of loose bases, thus resulting in two copies of the original DNA duplex.

The genetic information contained in the DNA is encoded in the actual order or sequence of bases that comprise it. In other words, from the point of view of the information it carries, a gene is no more than a sequence or word in a four character alphabet. This hints at the possibility that at least certain aspects of genetics may be studied purely as a textual problem, divorced from the physical and chemical properties of the substances involved.

Gene expression, the process whereby the information carried in the DNA is translated into the appropriate physical or chemical effect, is accomplished by translating the DNA sequences into *proteins* or *polypeptides*. These macromolecules are essentially responsible for controlling all the chemical activity in the cell either as reagents or, more importantly, as catalysts. This translation, as well as certain other cellular processes are mediated by *Ribonucleic Acid* or RNA, which is a single stranded nucleic acid chain with almost the same base alphabet as DNA. Proteins are comprised of long chains of *amino acids*. There are twenty different amino acids and as with DNA the structure and activity of proteins is fully determined by the actual sequence of amino acids that comprise it. Also a

difference between them, the difference does not affect the computational problems which we shall be considering.

fixed DNA sequence gives rise to a fixed amino acid sequence, or, in other words, one gene - one protein. The translation between DNA and protein was found to be by means of a fixed triplet code. Each sequence of three adjacent bases (a *codon*) codes for a specific amino acid. This *genetic code* is not only fixed for the entire genetic material in an organism but is in fact common to all life on earth. Thus the process of *transcription* of genes to proteins consists of reading the sequence of bases three at a time and adding the appropriate amino acid to the protein molecule being constructed. All codons represent an amino acid (there are no nonsense triplets) except some known as *TERM* codons which indicate where the transcription is to start and end.

Thus, in the case of both DNA and protein their effect is determined completely by the sequence of their constituent bases or amino acids. In the case of proteins, as well as RNA, much of their function comes from the shape in which the molecule folds about itself, known as its *secondary structure* or *3-D conformation* but these higher order structures are themselves completely determined by the primary structure - the sequence itself. Similar sequences will therefore, in some sense, have similar effects or properties. The notion of similarity is yet to be defined, but it should be intuitive at this point that if there is very little difference between two gene sequences, then it may be expected that there will be very little difference in the proteins that they code for and in the activity of those proteins. This further shows the important role that sequence related problems play in the study of genetics.

Genetic Mutation

The ability of DNA to act as a carrier of genetic information arises from the ability of the cell to copy it exactly thus passing on the identical genetic information to successive generations. This process is known as *transcription*. However due to errors in this transcription process the copy is not always identical to the parent and this change is further passed on to future generations by transcription. This process, whereby a DNA sequence differs from its parent is known as *mutation*. Genetic mutation is a very important phenomenon since it is responsible for genetic diversity as well as evolution. Thus the similarity of genetic sequences between two species can offer a clue to their common ancestry on the evolutionary tree. This too motivates interest in finding similarities between genetic sequences.

The effect of mutations may also be varied. Some may cause no difference in the gene expression, while others may block the gene expression altogether. We will look at the process of mutation further in chapter IV in trying to establish a mathematical model for the process.

This thesis focuses on sequence similarity and mutation related problems. Essentially we are interested in efficient techniques for quantifying the similarity of sequences or finding similar sequences from among a large set. However many other types of problems, of interest from a computer science point of view, also arise in the context of molecular biology. Some of these are outlined below.

Computational Problems in Molecular Biology

Improving techniques for determining the actual sequence of bases comprising a DNA sequence have led to a rapid increase in the amount of sequence information available. This large and rapidly increasing volume of DNA and Protein sequence information leads to obvious problems involving their storage and retrieval and essentially constitutes a database problem.

Another problem arises in the context of "sequencing" or determining the actual nucleic acid sequence in a given strand of DNA. Due to the physical limitations of the process used to determine the base structure - a process known as gel electrophoresis - it becomes difficult to read extremely large fragments with any accuracy. Hence the methods commonly used involve "digesting" or breaking up the DNA with restriction enzymes and sequencing the smaller fragments. This is done with various restriction enzymes that break the DNA at different points thus resulting in various overlapping fragments of DNA. The problem then is to determine exactly where each fragment fits into the overall sequence (the *Restriction map*) by looking at the various fragments and determining exactly where the overlaps should lie. This is further complicated by having to account for errors in the sequencing process.

The RNA secondary structure problem deals with trying to predict the shape of a given segment of RNA given that it tends to fold back on itself and form bonds between bases at different positions along the RNA strand based on various thermodynamic considerations. It is important to know this *Secondary Structure* (as opposed to the primary structure which is the sequence itself) since that is what determines what portion of the

RNA sequence is exposed, which in turn determines its activity since many biochemical processes take place by means of a lock and key mechanism, *ie*., one macromolecule "fitting" into another.

III. Sequence Homology

What is understood by the statement that two sequences are similar depends largely on what meaning or interpretation is given to the sequences. In this they differ from numerical data in the sense that the set of integers or real numbers and the operations possible on them are well defined and are not subject to change, although, when representing any data as a number or set of numbers we are free to choose the manner of that transformation. For example we may chose to represent the heights of a set of people by a set of numbers using any of several different units but a person whose height is greatest will remain the same regardless.

This need not be necessarily be true in the case of sequence objects. Take for example the case where our objects are English words. If we are interested in identifying misspelled words we may wish to define the distance between two words as the number of single character changes (*ie.*, deleting or inserting a single character) which are required to change one into the other. We may even go to the extent of weighting the changes based on the distance between the characters on a standard typewriter keyboard. However this notion of similarity would be useless if we are interested in knowing how similar in meaning or ideas conveyed two words are. In the latter case we could perhaps define distance between two words as the number of "links" needed to find one starting from the other in a thesaurus. The important point to be noted here is that in cases where the data objects cannot be easily expressed as numbers by a representation that is physically simple (*ie.*, corresponds well with the real world

interpretation) any method used to study them must depend ultimately on interpretation by a knowledgeable human being. This will become important as we look at Genetic Sequences.

Similarity of Genetic Sequences

In the context of genetic sequences, similarity is referred to by the term *homology*. Two sequences which are similar or have similar regions would be described as having significant homology or homologous regions or simply as being homologies. As discussed in the previous section, the meaning we give to the term similarity depends on the context of the sequences or what they represent. In the case of genetic sequences, we are interested in their effects in biological systems. Thus the phrase "similar sequences" would, in the case of DNA, refer to sequences that code for similar proteins or polypeptides and, in the case of proteins, it would refer to proteins with similar function/structure.

Another reason for studying sequence similarity is to see how sequences evolve, especially since the evolution of the genetic material is inexorably linked with the evolution of the species. Looking at the mechanism of sequence mutation, outlined in the previous chapter, suggests a way of measuring sequence similarity based on quantifying the amount of mutation it would have taken to create one from the other. This leads to the concept of *edit distance* as a metric for the comparison of genetic sequences.

Edit Distance

One of the earliest general notions of "distance" as applied to character strings was the *edit distance*, proposed by Wagner and Fischer [25]. The idea of edit distance is based on three basic *sequence editing operations*, which are operations that can be performed on a given sequence to transform it into another. These operations are *insertions, deletions* and *substitutions*. Insertion refers to inserting a character at a given position in the string, increasing the length of the string by one, *ie.*,

$$\alpha\beta \rightarrow (\textit{insert } A) \rightarrow \alpha A \beta$$

Deletion refers to removing the character at a given position, thus decreasing the length of the string by one, *ie.*,

$$\alpha A \beta \rightarrow (\textit{delete } A) \rightarrow \alpha\beta$$

Substitution refers to replacing one character with another, not affecting the length of the string, *ie.*,

$$\alpha A \beta \rightarrow (\textit{substitute } A \textit{ by } B) \rightarrow \alpha B \beta$$

In the above A and B refer to arbitrary (but different) characters and α and β refer to arbitrary strings from the sequence alphabet. (If we so choose we may think of a substitution as an insertion followed by a deletion.) Further, a *weight* or *cost* is assigned to each operation. The *edit distance* between two strings then refers to the minimum weighted number of edit operations required to transform one string into the other or, alternately, to transform both strings into a third, arbitrary, string. The reason for the alternative definition is to ensure that the distance measure is *symmetric, ie.,* given two strings S_1 and S_2, the minimum weight sequence of edit operations which transform S_1 into S_2 and the minimum weight sequence of edit operations which transform S_2 into S_1, must have the same

weight. These edit operations are sufficient to transform any string to any other and thus the *edit distance* between two strings is always defined.

It should be noted that a sequence of edit operations which transform S_1 into S_2, can be converted into one that transforms S_2 into S_1 by changing inserts to deletes and *vice versa* and by reversing any substitutions *ie.*,

substitute A by B

will be replaced by

substitute B by A

Thus, since insertions in one sequence are mirrored by deletions in the other, both insertions and deletions must have the same weight, else the

---

$S_1[1 \cdots m]$ and $S_2[1 \cdots n]$, input strings.
d = weight of insert or delete.
$s \leq 2d$ = weight of substitution.

$D[i, 0] = d \cdot i$
$D[0, j] = d \cdot j$
$\forall\ i$ in $1 \cdots m$ and j in $1 \cdots n$
if $S_1[i] = S_2[j]$

$$D[i,j] = min \begin{cases} D[i,j-1]+d \\ D[i-1,j]+d \\ D[i-1,j-1] \end{cases}$$

else

$$D[i,j] = min \begin{cases} D[i,j-1]+d \\ D[i-1,j]+d \\ D[i-1,j-1]+s \end{cases}$$

edit distance = D[m,n]

Algorithm 3.1
Calculation of Edit Distance

---

costlier operation will never be used. Also, the weight of the substitution cannot be more than twice that of an insertion or deletion since we may choose to perform an insert followed by a delete instead of the substitution. In particular, we shall refer to the edit distance obtained by defining the weight of an insertion or a deletion as being 1 and the weight of a substitution as being 2, as the *Unit Weight Edit Distance*. This will become important later on.

The definition of edit distance leads almost immediately to the simple dynamic programming algorithm for its calculation as outlined in Algorithm 3.1 [25]. This algorithm is based on calculating the edit distance between all possible prefixes of the two sequences and the dynamic programming step is based on the observation that when calculating the edit distance between a prefix of length i of sequence $\boldsymbol{S}_1$ and a prefix of length j of sequence $\boldsymbol{S}_2$, we must perform one of following actions:

Delete the i^{th} character of $\boldsymbol{S}_1$: The edit distance is then that between the current prefix of $\boldsymbol{S}_2$ and the previous prefix (of length $i-1$) of $\boldsymbol{S}_1$ + 1 (or other cost)
to account for the last deletion.

Delete the j^{th} character of $\boldsymbol{S}_2$: Symmetrical to the above

Substitute the last character of one by the last character of the other: The edit distance is then the the distance between the previous prefixes of both sequences + 2 (or other substitution cost) to account for the substitution.

match the last two characters (if they are identical): The edit distance is then the same as that between the previous prefixes of both sequences.

The calculation of edit distance by this dynamic program requires three comparisons for every pair of prefixes, one from each sequence. Thus the running time of the algorithm is quadratic in the sequence length, taking O*(nm)* steps where *n* and *m* are the lengths of the sequences.

The unit weight edit distance is also related to the concept of *longest common subsequence* of two sequences, which is simply the longest subsequence (not substring) common to both sequences, which is another method of describing the amount of similarity between two sequences.

Variations on the Dynamic Program

Needleman and Wuncsh [18] first applied such a dynamic programming approach to determining homologies between protein sequences, and since then the notion of edit distance has been widely accepted as a measure of homology among genetic sequences by the molecular biology community. Their technique differs from that of Wagner and Fischer in that they calculate a measure of *closeness* rather than a measure of distance and that they use a non uniform weighting for substitutions based on the number of matching bases (nucleic acids) in the triplets coding for each amino acid pair. However the technique is essentially similar.

Other variations include techniques that incorporate different weighting schemes for gaps or consecutive inserts/deletes, for example the weight or cost for a string of consecutive deletes (a "gap") may be concave [26], *ie.,* the weight of each successive delete costs less than the previous one. The scoring scheme proposed by Wilbur and Lipman [27], for example, assigns a uniform constant score to each gap irrespective of their length. Further, their algorithm calculates the *alignment* between a pair of

sequences, rather than just a number that is indicative of their similarity. The alignment is a map showing identical bases of both sequences that correspond to each other, and is meant to indicate that in the optimum (least weight) series of edit operations transforming one sequence to another, these positions remain unaffected. This information is often of much more importance to biologists than just the distance alone, for it is the actual bases in alignment that determine whether the homology observed between two sequences has any real significance or not.

The essential limitation of this family of techniques is the quadratic time complexity of the dynamic program. This can be especially crucial in light of the fact that, as sequencing techniques improve, biologists are able to sequence longer and longer strings of genetic material. It therefore becomes imperative to search for faster comparison techniques, perhaps even at the expense of resolution. Also, often the problem is to be able to compare a set of sequences rather than a pair. In such a situation, the ability to do only pairwise comparisons may be too time consuming and may not even be meaningful.

Various heuristic approaches have been proposed to lower the quadratic time limitation of the dynamic program and many techniques originally developed for general string matching and string comparison have been applied to the comparison of genetic sequences.

Landau Vishkin and Nussinov [10] have described a heuristic algorithm that is linear in the sequence lengths provided that the sequences have no more than k differences between them for a fixed k. The running time does depend on k. This class of heuristic, that performs well for

close sequences but may not be accurate for very dissimilar sequences, is very important in the context of genetic sequences since the distance that is calculated is only meant to draw attention to close sequences and may not be significant in itself. Hence, for dissimilar sequences, it is sufficient if the algorithm pronounces them to be dissimilar without being accurate about the actual distance.

Perhaps the most widely used programs for biological sequence comparison are the FASTA and FASTP family of programs developed by Lipman and Pearson [14][19]. These use a modification of the algorithm of [27] and speed up the dynamic program by first looking for regions of exact matches of small lengths using a lookup table and then concentrating the dynamic program around these regions of similarity, thus reducing the number of comparison and minimization steps the dynamic program must perform. However this method can, in the worst case, have the same quadratic time complexity.

Various hardware approaches have also been proposed for speeding up the dynamic programming computation of sequence homology. The Princeton Nucleic Acid Comparator of Lopresti and Lipton [15] uses a systolic array implementation of dynamic programming to produce a linear-time implementation of the edit distance computation, using a linear number of processors. Lander and Mesirov describe a parallel algorithm for protein sequence comparison.

Concerning the problem of comparing or analyzing more than two sequences at a time Queen, Wegman and Korn [20] have developed an algorithm which can deal with multiple sequences by searching for partial homologies common to all or some minimum sized subset of the

sequences. This is achieved by finding all possible substrings of a given length in each sequence, as well as noting their position and then comparing the substring composition of all the sequences to see which ones share common substrings or similar substrings, where "similar" is defined as substrings not differing in more than a certain number of positions.

IV. Abstractions for Mutations in Genetic Sequences

This chapter will introduce certain mathematical models for the sequence mutation process. This is important in terms of developing a framework within which to pose and answer questions about sequence mutation related problems, as well as to make any kind of analysis of the algorithms we encounter.

In biological terms, a *mutation* is the chemical process that causes a DNA sequence to be different from its parent, of which it is normally expected to be an identical copy. It can also refer to the result of such a process. This can happen in a variety of ways: as the result of errors during transcription, causing single bases to be changed into others, or blocks of one or more bases to omitted altogether from the copy. This process is naturally occurring and for the most part can occur anywhere in the sequence but mutations have been known to concentrate sometimes at specific locations in a sequence known as *hot spots*. Certain chemicals and radiation may also increase the incidence of these errors.

For our purposes however, we may think of mutation as simply the process by which two genetic sequences differ. These sequences may be parent and child in which case this would correspond to the biological notion above, or they may be two arbitrary sequences between which we are trying to establish a possible ancestor-descendent relationship or a possible common ancestor, or even a sequence and its "image" *ie.*, what the sequencing process tells us that the sequence is. In the latter case the difference is between the physical reality and our observation of it. This

difference can occur due to possible errors in the process of "reading" a sequence.

We present below two models for mutations. The idea behind any model should be to mirror what is known about the process it is modeling and yet be simple enough to work with. Thus, for example, if we were interested in comparing typewritten samples of text we may wish our model to incorporate the concept of transposition - two consecutive characters being exchanged - since that is a kind of error we would expect to happen during typing but we would not expect such an error mechanism in the context of DNA sequence mutation and hence our model would probably not incorporate such a mechanism.

The P-I-D Model

In this model we view a mutation as a black box or a machine that reads in a sequence one character at a time and writes out the other one character at a time.

original sequence $\rightarrow$ **[mutation]** $\rightarrow$ **mutated sequence**

At any given time the machine can perform one of three actions:

PASS copy a character from the input to the output.

INSERT put a character on the output without consuming any input.

DELETE consume an input character without producing any output.

This model is universal in the sense that for any two sequences we can find a sequence of moves that will take us from one to the other. Also, under the assumption that transcription or replication errors can only occur at that point where a sequence has been opened up for reading, this

linear view is quite appealing. These sorts of errors occurring at the point of transcription - so called *Point Mutations* - form an important class of mutation events and are known to be widely occurring in biological systems.

The machine itself can be viewed as a finite automaton or Markov process. This gives the advantage of being easy to simulate, a facility which can be useful when empirically trying to study the performance of any heuristic algorithms when theoretical predictions are difficult to make. Also the ability to change the probability of occurrence of mutation events, for example, to make certain types of substitution more common than others gives this some flexibility.

As can be seen, this model mirrors the dynamic program for the edit distance calculation. This then is one of the advantages of the edit distance as a measure of sequence similarity, since it lends itself to an elegant and simple mathematical representation which allows many theoretical observations to be made about the edit distance between two sequences as well as about the algorithm itself. Chvatal and Sankoff [3], for example, have proven bounds on the largest common subsequence of two random sequences. This is of importance since it provides a background against which to determine if any perceived similarity is significant or merely the result of chance.

The Mutation Pattern Model

We may look at a mutation as a sort of mask laid over the sequence with positions marked as "insert particular character here" or "delete here". This view is of use when we are interested in counting possible mutations

and seeing if different mutations can lead to the same result (*ie.*, can a particular mutation acting on a particular sequence be replaced by another that will have the same effect and is smaller (fewer edits)). These sorts of problems are important in the biological context because it is sometimes of interest to actually reconstruct the mutation events that occurred to cause one sequence to change into another and in such cases it may be important to be able to establish good bounds on the probability that an observed difference between two sequences may be caused by mutations different from the ones predicted by the edit distance algorithm or other means. Another related problem is that of establishing the probability that a region that appears identical in two sequences is in fact *conserved*, *ie.*, unaffected by any mutations, as opposed to the possibility that it was subjected to a series of mutations whose net result was to cause the regions to be identical in the two sequences. We shall address this question in the next section.

Define a *mutation* of size k to be a list of k commands, to be executed in order, of the form

INSERT (*character*,*position*):

INSERT *character* into the *position* place in the sequence. ($1 \leq$ *position* $\leq 1+|$ *length of sequence*$|$)

DELETE (*position*):

DELETE the character in the *position* place.

Obviously a mutation is only well defined for sufficiently long sequences (*i.e.*, the sequence is long enough for *position* to fall within the sequence). Therefore we will insist that the size of sequence on which the mutation is to act be explicitly stated. It is obvious that we can have the

same mutation act on longer sequences too but for clarity we shall call that a different mutation. It is also obvious that the size of the mutation is an upper bound on the edit distance.

Note that *position* in the above definition refers to position in the original sequence. We could have defined it as the position in the sequence as it looks after the last edit operation but this could lead to redundancies such as inserting a character and then deleting it. A consequence of this is that all the delete operations can be permuted among themselves as well as among the insert operations so we need only think of the combination of such operations rather than the actual order. Similarly insert operations can be permuted with the important exception of two or more inserts occurring at the same position, in which case their relative order must be maintained, since they represent a string of characters being inserted at that position.

It may be useful at times to look at a modified version of the above formalism in which we split the pattern into two parts: an *insert pattern* consisting only of insert operations and a *delete pattern* consisting of only deletes. We apply one, generating a new sequence and then apply the other to the resulting sequence. Thus if the parts are of size i and d respectively ($i + d = k$, the size of the whole mutation), and we apply the insert part first on a sequence of length N, then the delete part must be defined on a sequence of length $N+i$.

The Middle Third Lemma

One reason for studying the similarity or homology between genetic sequences is to trace the evolutionary history of the sequences - to be able to say that one evolved from the other or that two sequences share a

common ancestor. To do this it is important to be able to determine what were the mutations that actually occurred. The edit distance metric however, can only suggest what and how many the smallest number of mutations should be. Thus if we perform an extremely large number of mutations on a sequence and compare it with the result we may get an edit distance actually smaller than the number of mutations we performed and possibly a quite different set of mutations. Therefore we may be interested in knowing what the probability is that the mutations suggested by the edit distance algorithm are in fact the ones that occurred, or, alternately, what is the probability that a substring that seems to be identical between two sequences is in fact identical; that is, there were no mutations at all in that region. We address this problem briefly in this section for the case where the number of mutations are not too large. We need to assume a small number of mutations in order to be able to make some statement about whether the subsequence is conserved or not. For large numbers of mutations it becomes difficult to identify the conserved regions. We therefore assume that the number of mutations k is less than $N/logN$ which is not much worse than assuming the number of mutations to be a constant fraction of N.

Given a Sequence S of length N, which undergoes $k \leq N/logN$ mutations to become S', we wish to know if we can find of subsequence of length L which is *conserved* from S to S'. We may understand the idea of conservation as saying that if we remove this subsequence from both S and S' then the edit distance between the right halves of the two sequences and that between their left halves add up to at most the edit distance between the two sequences themselves. This does not guarantee that

the two subsequences are unmutated copies of each other but only that they have identical base sequences.

We further assume that the k mutations are uniform and *i.i.d.* over the entire sequence, *ie.* the *mutation rate* is $m=\frac{k}{N}$.

To assure a decent chance of finding such a sequence, let $L(N) = N/k(N) = \log N$. This assures us at least one and expected $O(k)$ such sequences. (From the coupon collector problem we can see that as $L \cdot k$ exceeds $N \log N$ the probabilty of our being able to find such a sequence dimnishes rapidly).

Now having found a subsequence common to both S and S' we must assure that it is indeed "valid" for the criteria above; that is, it shouldn't be a stochastic aberration caused because we have chosen an L so small that we find many subsequences of that length identical to each other.

Now obviously the (starting) positions of the subsequences S and S' can differ by no more than k, since we know that there were only k mutations and therefore that is the maximum distance they could have been moved apart.

Now we have to assure that any matching subsequences we find represent the same "position" in both S and S'; that is, we have to ensure that we don't get any false matches. We may get a false match if in S we already have two subsequences of length L which are identical (to within $k/N \cdot L \approx 1$ mutations) and are within a distance of k positions of each other.

Prob{ two sequences of length L are identical to within 1 mutation } $\approx \frac{5L}{4^L}$ Since a mutation could be either a delete or an insert of one of four bases and this could happen at any one of L locations, thus there are $5L$

possible DNA sequences within a distance of one mutation from a given sequence of length L and there are 4^L total possible sequences of that length. This then is the probability that an arbitrary substring of length L in S' is a "false match" with given substring of length L in S. The probabilty that the two do not constitute a false match is therefore 1 minus the above. We wish to ensure that for a given substring in S, no substring of S' which is within k positions is a false match. This would ensure that the match we have found is in fact a conserved region. For this to be true all L length substrings S' within k positions of the one in S must not be false matches. The probability of this is therefore:

$$\text{Prob } \{ \text{ No false matches } \} \approx \left[1 - \frac{5L}{4^L} \right]^k$$

We want this to increase to 1. Therefore, obviously L must grow faster than any constant; that is, $L > \Omega(1)$.

But this is not quite enough. we require

$$\frac{d}{dN} \left[1 - \frac{5L}{4^L} \right]^{N/L} > 0$$

(since $k \leq N/L$).

$L = \log N$ satisfies this.

Thus if two sequences of length N differ by fewer than $k = N/\log N$ mutations then, with high probability, identical substrings of length $\log N$ within k positions of each other represent conserved regions.

Now it is possible that even though we may have obtained such a "valid" match, there could have been some mutations within this subsequence itself, especially near the ends. How could this occur?

Now if there was only one mutation then it is obvious that from the point of deletion towards one of the two ends the sequence must consist entirely of one single character repeated over and over — since we are in effect sliding that portion over by one position and finding that it matches with itself! Therefore a prefix (or suffix) of length 1 is repeated. Similarly if there are two deletes one would have a two-base pair repeated and so on. (If there was an insert and a delete then there would be a single character repeated in the interval between the two.) Thus it is evident that we are unlikely to find any mutations toward the center of the sequence.

Now let $P[i \mid t]$ be the probability that the innermost mutation is at a distance at least i characters from either edge given that t mutations do occur. We can bound $P[i \mid t] \leq \frac{4^t}{4^i}$. *ie.*, a fragment of length t is repeated over and over from position i to the edge. We can use this to place an upper bound on the probability of a mutation affecting the central portion of a matching region.

Taking $L=logN$, $K = N/logN$ we get mutation rate $m = 1/logN$ so $P[i]$ = probability that innermost mutation is at least i characters from the edge

$$= \sum_{t=1}^{i} P[i \mid t] \cdot \text{Prob}[t \text{ mutations}]$$

$$\leq \sum_{t=1}^{i} \frac{4^t}{4^i} \cdot m^t \cdot L$$

$$= \sum_{t=1}^{i} \frac{4^t}{4^i} \cdot \left(\frac{1}{logN} \right)^t \cdot logN$$

$$= \sum_{t=1}^{i} \frac{4^t}{4^i} \cdot \left[\frac{1}{L}\right]^t \cdot L$$

For $i = L/3 = logN/3$ we get:

$$P[L/3] \leq \sum_{t=1}^{L/3} \frac{4^t}{4^{L/3}} \cdot \left[\frac{1}{L}\right]^t \cdot L$$

$$= \frac{L}{4^{L/3}} \cdot \sum_{t=1}^{L/3} \left[\frac{4}{L}\right]^t$$

$$= \frac{L}{4^{L/3}} \cdot \frac{4}{L} \cdot \frac{1-(4/L)^{L/3}}{1-4/L}$$

$$\leq \frac{4^{1-L/3}}{1-4/L}$$

$$= \frac{L \cdot 4^{1-L/3}}{L-4} \qquad (4.1)$$

But $P[L/3]$ is in fact the probability that the middle third of the sequence was affected by any mutations at all. Plugging in some actual values for L we get:

L	$P[L/3]$
10	0.0656
15	0.0053
20	0.0005

Thus we see that when we find even relatively small matching substrings between two sequences we may state with a good measure of certainity that at least the middle third of the matching area was unaffected by (point) mutations.

Hence,

Lemma 4.1 : If a substring of length L is observed to be identical between two DNA sequences, we may state that with probability $1 - P[L/3]$ — where $P[L/3]$ is given by equation 4.1 above — that the middle third of that substring was conserved from one to the other (or to both from a common ancestor) without being affected by any mutations.

V. The Euclidean Vector-Distance Metric

This chapter introduces a different measure of sequence similarity, which we shall refer to as the *Euclidean* or *Vector-Distance* metric. This metric is based on representing sequences as points by means of the *tuple frequency vector*. Its validity is established in the comparison of genetic sequences by showing the correspondence between this metric and the edit distance metric discussed previously. We shall also see how this method can be applied to speeding up sequence comparison.

The k-Tuple Frequency Vector

Consider sequences from a given alphabet Σ. The *k-tuple frequency vector* represents the relative frequency of various strings of length k (including overlaps) in a given sequence. We use the term *k-tuple* to refer to a string of length k, $a_1 a_2 \cdots a_k$. where each $a_i \in \Sigma$ and k is the *tuple size*. For a given alphabet Σ and tuple size k there exist exactly $d = |\Sigma|^k$ possible distinct k-tuples.

A sequence $\boldsymbol{S}$ of length N has $N-k+1$ substrings of length k. The relative frequency of a particular k-tuple is therefore the number of times it appears in $\boldsymbol{S}$ divided by $N-k+1$. By ordering all the d k-tuples in some fixed order (perhaps lexical order) and listing their relative frequencies in $\boldsymbol{S}$ in that same order we have a vector $\overline{V}$ of relative k-tuple frequencies in the sequence $\boldsymbol{S}$. It follows that $\boldsymbol{S}$ uniquely determines $\overline{V}$. We shall refer to $\overline{V}$ as the k-tuple frequency vector associated with $\boldsymbol{S}$ or simply say that $\overline{V}$ is

the *vector of* $\boldsymbol{S}$ and we may represent this by writing $\overline{V}(\boldsymbol{S})$. Example 5.1 shows this conversion for an alphabet of size $|\Sigma|=4$ and $k=2$.

We may think of *vector* as a function that takes a sequence $\boldsymbol{S}$ to a unique point $\overline{V}(\boldsymbol{S})$ in the unit cube in d dimensions. All the points fall in the unit cube because $\overline{V}$ is a vector of relative frequencies hence each component must be non-negative and all the d components are bounded by one. Further the sum of all the components must be one. It follows that one dimension is, in fact, redundant but we shall continue to consider $|\Sigma|^k$ dimensions for symmetry. It should be noted that this

---

Σ = { A, C, G, T } k = 2

S = ACTGATACGATTAGCAGTGACAGATAGACCAGTAACCGGTTACCCGATTTT

base pair	num. of occurrences	$\overline{V}(\boldsymbol{S})$
AA	1	0.02
AC	6	0.12
AG	5	0.10
AT	4	0.08
CA	3	0.06
CC	4	0.08
CG	3	0.06
CT	1	0.02
GA	6	0.12
GC	1	0.02
GG	1	0.02
GT	3	0.06
TA	5	0.10
TC	0	0.00
TG	2	0.04
TT	5	0.10
total	50	1.00

Example 5.1
Conversion of Sequence to Vector

---

transformation is undefined for sequences shorter than k (unless we were to add a 'blank' or 'padding' character to the alphabet which does no occur in any sequence and use it to pad out short sequences). However, as we shall see, we will not be dealing with very large k and hence all sequences of any interest at all will be long enough.

Given a set of sequences we may therefore convert them into what is essentially a set of points. The advantage of this is that the concepts of distances between points or groups of points are fairly widely studied and perhaps easier to understand and deal with than the corresponding concepts for sequences of text.

Given two sequences $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ we shall refer to the 2-norm of $\bar{V}_1 = \bar{V}(\boldsymbol{S}_1)$ and $\bar{V}_2 = \bar{V}(\boldsymbol{S}_2)$

$$|\bar{V}_1 - \bar{V}_2| = \left[\sum_{i=1}^{d} (V_1^i - V_2^i)^2 \right]^{1/2} \tag{5.1}$$

where V_j^i is the ith component of vector V_j, as the *euclidean distance* or *Vector distance* between the sequences $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$.

The calculation of $\bar{V}(\boldsymbol{S})$ takes time linear in the length of sequence $\boldsymbol{S}$ as can be seen from algorithm 5.1. Further, the calculation of vector distance is linear in d as is obvious from the definition in (5.1). Also the calculation of the vector need be done only once per sequence. The upshot of this is that we now have a metric whereby the distance between two sequences can be calculated in time essentially linear in sequence length, far faster than the quadratic time needed to calculate edit distance.

However the conversion from sequences to vectors is not one-to-one but rather many-to-one, since different sequences may have identical

$\Sigma = 0,1, \cdots ,s-1$: input alphabet of size s.
k : *tuple size*
$S[1 \cdots n]$: input string
$\overline{V} = (V^0,...,V^{s^k-1})$: output vector

initially : $\overline{V} = (0,0,...,0)$

$$b = \sum_{i=1}^{k-1} S[i] \cdot s^{k-1-i}$$

for i from k to n
    $b = (b \cdot s + S[i]) \bmod s^k$
    increment V^b by $1/(n-k+1)$

Algorithm 5.1
Calculation of $\overline{V}(S)$

vectors (Example 5.2), and it is therefore not possible to reconstruct sequences from their vectors. In essence we have given up some information about the sequences in the interests of speed. We must therefore ensure that we have retained enough information to make worthwhile observations.

The idea of looking at short strings of some fixed or maximum length is not a new one and has been used in an implicit manner in several heuristic algorithms for finding the edit distance of sequences, such as the multiple sequence alignment program of Queen *et al.* [20]. The "d^2" measure of Torney et. al. [24], also uses a similar technique to compare sequences. However, these techniques continue to treat the DNA or Protein sequence as the basic data object and are therefore bound by the limitations of that representation.

$\Sigma = \{ A, C, G, T \}$ $\quad k = 2$

S_1 = **TAGTACTTGTCCATTGTACAT**
S_2 = **TGTACATTAGTACTTGTCCATAGTACTTGTACATTGTCCAT**

base pair	num. of occurrences in S_1	S_2	$\bar{V}(S_1)$ $=\bar{V}(S_2)$
AA	0	0	0.00
AC	2	4	0.10
AG	1	2	0.05
AT	2	4	0.10
CA	2	4	0.10
CC	1	2	0.05
CG	0	0	0.00
CT	1	2	0.05
GA	0	0	0.00
GC	0	0	0.00
GG	0	0	0.00
GT	3	6	0.15
TA	3	6	0.15
TC	1	2	0.05
TG	2	4	0.10
TT	2	4	0.10
total	20	40	1.00

Example 5.2
Two Sequences Mapping to Identical Vectors

Correspondence Between Edit Distance and Vector Distance

We make a case for the usefulness of the vector distance metric as defined above by showing that it is related to the widely used edit distance measure, thus giving it some legitimacy. It is possible to understand intuitively how there could be a relationship between this euclidean distance and the edit distance by imagining how the vector of a sequence S is affected as we edit it to create another sequence S'. Let us assume that the size of the tuples k is small compared to the length of the sequences, that we do not perform too many edits and that the edits are spaced more or

less evenly. Now a single substring of length k will contribute to its position in the vector an amount inversely proportional to the length of the sequence. This is evident from the fact that the number of such substrings is approximately the length of the sequence (for small k), and the sum of all the vector components is one. Further, a single edit operation can only cause a small (inversely proportional to sequence length) change in at most a few ($2k$) positions in the vector. Also the number of such changes, which in turn affects the distance between the resulting vectors, generally increases with the number of edits, which is the edit distance! (This is in general not true for very large numbers of edits in which case the edit distance may in fact turn out to be smaller than the number of edits performed.)

Let us start with a source sequence $\boldsymbol{S}$ and perform edit operations on it one by one and try to estimate the effect on the resulting vector.

let

Σ = *sequence alphabet*

v = *vector distance* (to be estimated)

e = *edit distance* (number of edits performed)

N = *length of sequence* $\boldsymbol{S}$

k = *size of k-tuples*

d = $|\Sigma|^k$ (number of dimensions)

$a,\ a_1,\ b$, etc, $\cdots \in \Sigma$

$\alpha,\ \alpha_1,\ \beta$, etc, $\cdots \in \Sigma^*$

For each edit of the form :

$\alpha a_1 a_2 \cdots a_k a_{k+1} \cdots a_{2k-1} \beta \rightarrow$ delete $a_k \rightarrow \alpha a_1 a_2 \cdots a_{k-1} a_{k+1} \cdots a_{2k-1} \beta$

There is a change of approximately $-\frac{1}{N}$ in the vector positions corresponding to the k-tuples

$a_1 \cdots a_k$, $a_2 \cdots a_{k+1}$, ..., $a_k \cdots a_{2k-1}$

and approximately $+\frac{1}{N}$ in the vector positions corresponding to the k-tuples

$a_1 \cdots a_{k-1} a_{k+1}$, $a_2 \cdots a_{k-1} a_{k+1} a_{k+2}$, ..., $a_{k-1} a_{k+1} \cdots a_{2k-1}$.

The case of insertion is symmetrical. Thus a total of $2k - 1$ positions are affected by one edit. An example may serve to illustrate this point. Example 5.3 shows the effect of a single deletion in a DNA sequence on its corresponding vector. For the sake of example we assume a vector size of $k = 2$. In the case where $a_1 \cdots a_{2k-1}$ contains a substring of length greater than k which consists of a repeated single character, there are in fact fewer positions affected. Substitutions may be thought of as an insert followed by a delete but there are only $2k$ positions affected by the two operations put together. Further if two edits fall within a distance of k characters of each other, they will together affect perhaps fewer but not more positions than each one separately. Thus the number of changes is bounded by $2ke$.

Assuming that these changes are distributed evenly among the d components of the vector, we can estimate the resulting vector distance v :

if $2ke > d$:

we have d changes of approximately $\frac{2k \cdot e}{d \cdot N}$ each and hence :

Σ = { A, C, G, T } $k = 2$

S_1 = **ACTGATACGATTAGCAGTGACAGATA.*G*.ACCAGTAACCGGTTACCCGATTTT**
deleting **G** gives :
S_2 = **ACTGATACGATTAGCAGTGACAGATA.ACCAGTAACCGGTTACCCGATTTT**

base pair	num. of occurrences in S_1	S_2	$\bar{V}(fatS_{1})$	$\bar{V}(fatS_{2})$	change
AA	**1**	**2**	**1/50**	**2/49**	≈+1/50
AC	6	6	6/50	6/49	
AG	**5**	**4**	**5/50**	**4/49**	≈−1/50
AT	4	4	4/50	4/49	
CA	3	3	3/50	3/49	
CC	4	4	4/50	4/49	
CG	3	3	3/50	3/49	
CT	1	1	1/50	1/49	
GA	**6**	**5**	**6/50**	**5/49**	≈−1/50
GC	1	1	1/50	1/49	
GG	1	1	1/50	1/49	
GT	3	3	3/50	3/49	
TA	5	5	5/50	5/49	
TC	0	0	0/50	0/49	
TG	2	2	2/50	2/49	
TT	5	5	5/50	5/49	
total	50	49	1	1	

Hence

$$|\bar{V}(S_{1})-\bar{V}(S_{2})| \approx \left[3 \cdot \left[\frac{1}{50}\right]^{2}\right]^{1/2}$$

$$=\frac{\sqrt{3}}{50}$$

Example 5.3
Effect of Single Edit on Sequence Vector

$$v^2 \approx \left[\frac{4k^2 \cdot e^2}{d \cdot N^2}\right]$$

however, if $2k \cdot e < d$:

we have approximately $2k \cdot e$ changes of $\frac{1}{N}$ each and hence :

$$v^2 \approx \left[\frac{2k \cdot e}{N^2} \right]$$

Due to the nature of our assumptions, these can only be looked upon as upper bounds on the actual euclidean distance and these arguments may not be valid for large numbers of edits. However they do indicate that a relationship exists between the euclidean distance and the number of edit operations performed, that this relationship is quadratic, and that it is stronger for smaller numbers of edits. Hence,

For small numbers of edits on the order of $|\Sigma|^k$

$$v = O\left[\frac{\sqrt{e}}{N} \right] \tag{5.2}$$

We may also arrive at this conclusion by approximating the effect of each single mutation on the sequence vector as a step or rather a small number of steps in a random walk in d dimensions. Since each step is $O\left[\frac{1}{N} \right]$ in length and there are $O(e)$ such steps we may expect to arrive at the same conclusion as above.

Essentially, we may conclude that two sequences that are close in terms of edit distance will have close vectors. Thus comparison of sequences by means of this vector distance may be of some use after all and definitely merits further investigation, which is described in the next section.

The above discussion shows that, in general, a larger value of k is beneficial in that we get better resolution. In other words, a larger k,

which results in a larger (higher dimensional) vector, will result in a greater correlation between the edit distance and the vector distance. However the time complexity of the distance calculation is exponential in k. Therefore we must tradeoff between time and resolution and find a compromise that is not too time consuming, yet allows for meaningful comparisons.

Experimental Verification

The above discussion suggests the possibility that the concept of vector distance may be a useful tool in the comparison of genetic sequences. To further establish its usefulness we perform a series of computational experiments to compare it to the edit distance between actual sequences, the hope being that we will observe a strong correlation between edit distance and vector distance and that this in turn will provide a measure of confidence in the vector distance metrics. These experiments will also compare their relative running time performance in order to determine what computational advantages if any, may be obtained by using one metric instead of the other.

These tests were performed by generating pairs of sequences, in the manner described below and computing, for each such pair, both the unit weight edit distance, and the vector distance between the two sequences for a particular choice of tuple size k. The sequence pairs were generated as follows. In the case of DNA sequences, a sequence of the given length was chosen at random from a databank of known DNA sequences. This was then "mutated" by a simulation of the P-I-D machine described in the previous chapter to create the other sequence of the pair. Differing

mutation rates were used to provide a wide spectrum of edit distances. In the case of protein sequences, the pairs were generated by taking a random DNA sequence from the Genbank database, of thrice the required length, and translating it into a protein sequence by the known triplet coding of the genetic code. The other sequence of the pair was created by mutating the first one in the same manner as above.

The first series of experiments was run using a tuple size of 2. The results are presented visually in figures 5.1 - 5.10. The upper plot in each figure is a scatter plot showing the edit distance versus the vector distance. For the protein sequences, the lower plot shows the mean vector distance for a given edit distance (solid line) as well as the 1- and 2- standard deviation (dashed and dotted respectively) intervals of the vector distances for a given edit distance. The coefficient of the best fit quadratic curve relating the edit and vector distances is also indicated for the protein sequences. The obvious conclusion which can be drawn from observing these plots is that there is a very evident correlation between edit distance and vector distance in the case of protein sequences; a correlation that is markedly absent from our observations of DNA sequences. We may attribute this to our choice of tuple size k, since a tuple size of two in the case of DNA sequences will result in a vector of only $4^2 = 16$ dimensions, which may not be a large enough space to handle the sequence lengths being considered.

To test this hypothesis the experiments with DNA sequences were run again this time with $k = 4$ giving a vector space of $4^4 = 256$ dimensions. The results of these trials are displayed in figures 5.11 -5.14. This time the correlation between the vector and edit distances is well marked.

It may be observed that for larger edit distances the curves appear to flatten upwards. This may be due to the fact that as we perform a large number of edits, some may in effect, cancel each other out.

Figure 5.15 show the coefficient of correlation between the edit distance and (the square of) vector distance for protein sequences using tuple sizes of two and three. Similarly figure 5.16 shows the same information for DNA sequences. In both cases it may be observed that the coefficient of correlation does, in fact, increase with the tuple size, but not by much. The down side of increasing the tuple size is increased computation time.

The difference in computation time for the edit and vector distances using two different tuple sizes is shown graphically in figures 5.17 - 5.18. These clearly indicate the savings in time to be obtained by using the vector distance as a metric of sequence comparison as opposed to the edit distance. The results of increasing the tuple size are also immediately obvious. WE may therefore argue that the loss of correlation in using a vector of size two as opposed to size three (in the case of protein sequences) or of size four as opposed to size five (in the case of DNA sequences) is more than compensated for by the savings in time. Thus, for practical applications, it appears we may use a tuple size of two in dealing with protein sequences and four in dealing with DNA.

Conclusions

In this chapter we have introduced the concepts of the sequence vector and the euclidean vector distance as a measure of sequence homology and described their construction and calculation. We have also shown that this measure corresponds well with the more widely used edit distance measure

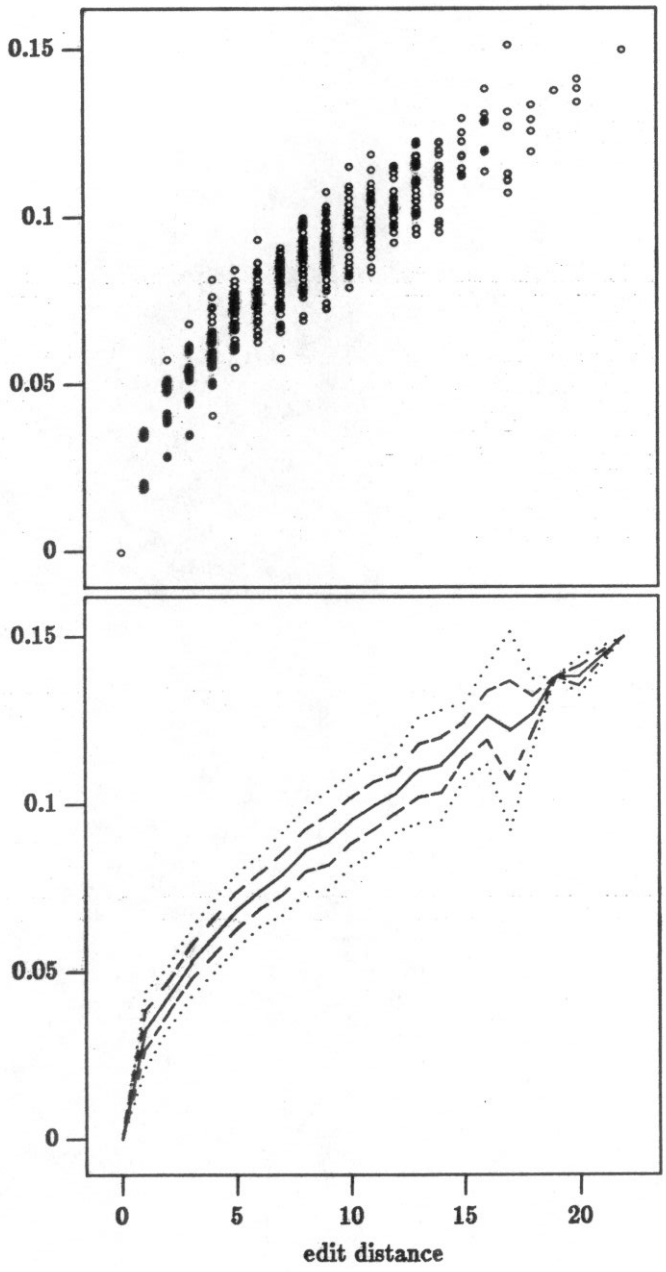
while offering distinct computational advantages in terms of time.

Some of the plots show a slight upward flattening as the edit distance increases. It is possible that this is due to the fact that as we perform more and more edits on the sequence the effect of some of them may tend to cancel out, in the sense that the edit distance between the original and mutated sequence may turn out to be noticeably smaller than the number of edits actually performed to transform one into the other.

In trying to establish the correspondence between the edit distance and the vector distance we have chosen to ignore the fact the edit distance metric is parametrizeable, by weights assigned to the different edit operations, and indeed this is one of the valued features of the edit distance. The correspondence between the edit and vector distance may not hold as strongly for all weighting schemes. However the vector distance is also parametrizeable in its own way. Thus, for example, we may choose to count the fact that two sequences match closely in some set of k-tuples to be more or less biologically significant than close counts in others. (For example, in the case of English language text, we may decide that knowing two texts have the same number of "qu" pairs conveys no further information than the fact that they have the same number of "q"'s and we may therefore choose to treat the "qu" dimension as less significant than the others). This sort of weighting can be easily accomplished in the vector distance case by stretching or compressing the space of the sequence vectors along those dimensions.

Another aspect to the vector distance metric, which is not developed in this thesis, is the fact that it very easily lends itself to implementation on parallel (vector) machines.

Protein Sequence Pairs

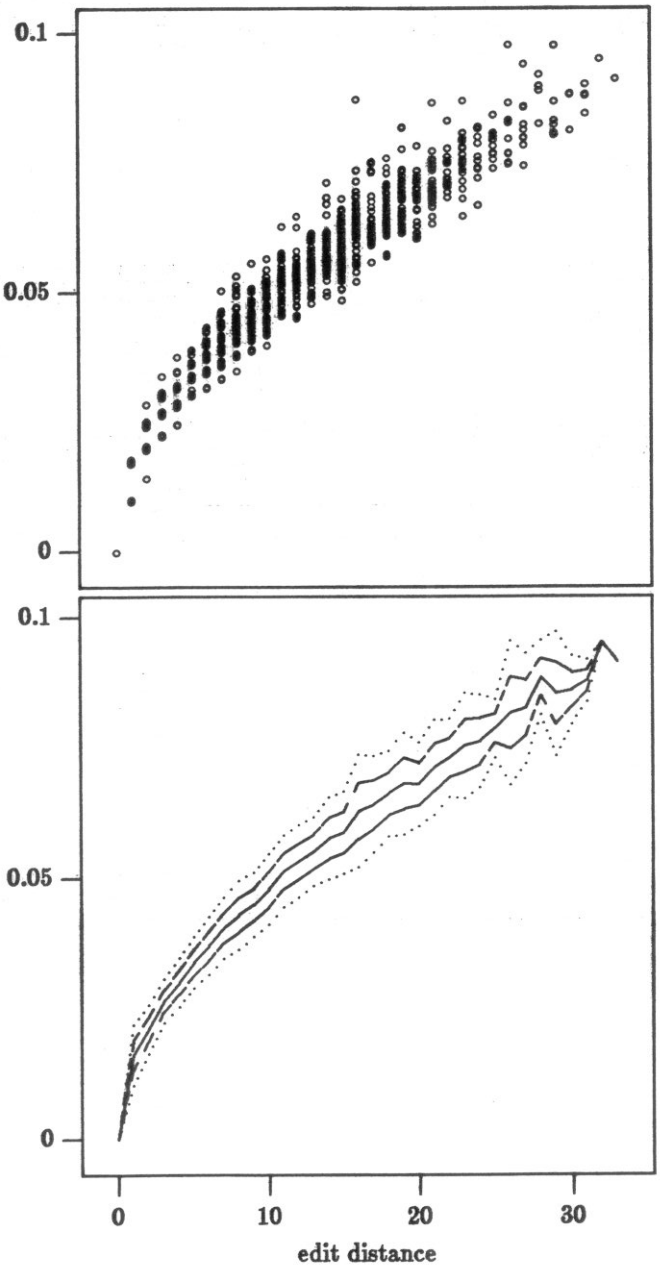


sequence length = 50

best fit $e = 1065 * v^2$

Figure 5.1

Protein Sequence Pairs

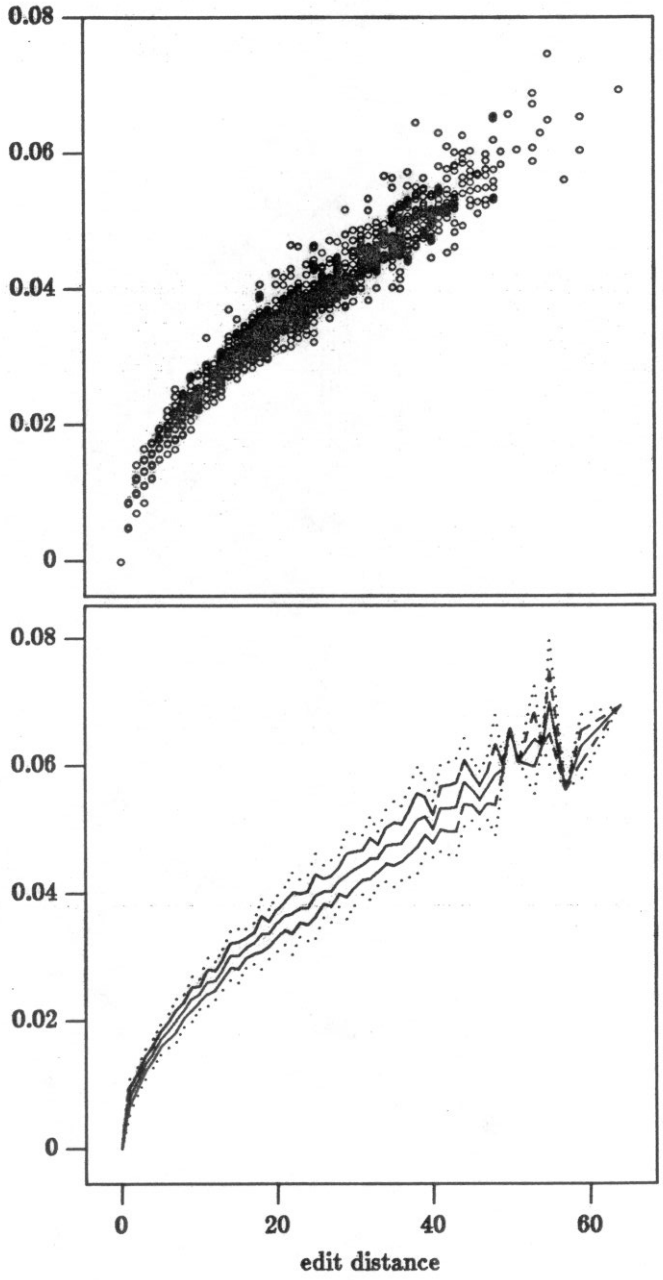


sequence length =100

best fit $e = 4051 * v^2$

Figure 5.2

Protein Sequence Pairs

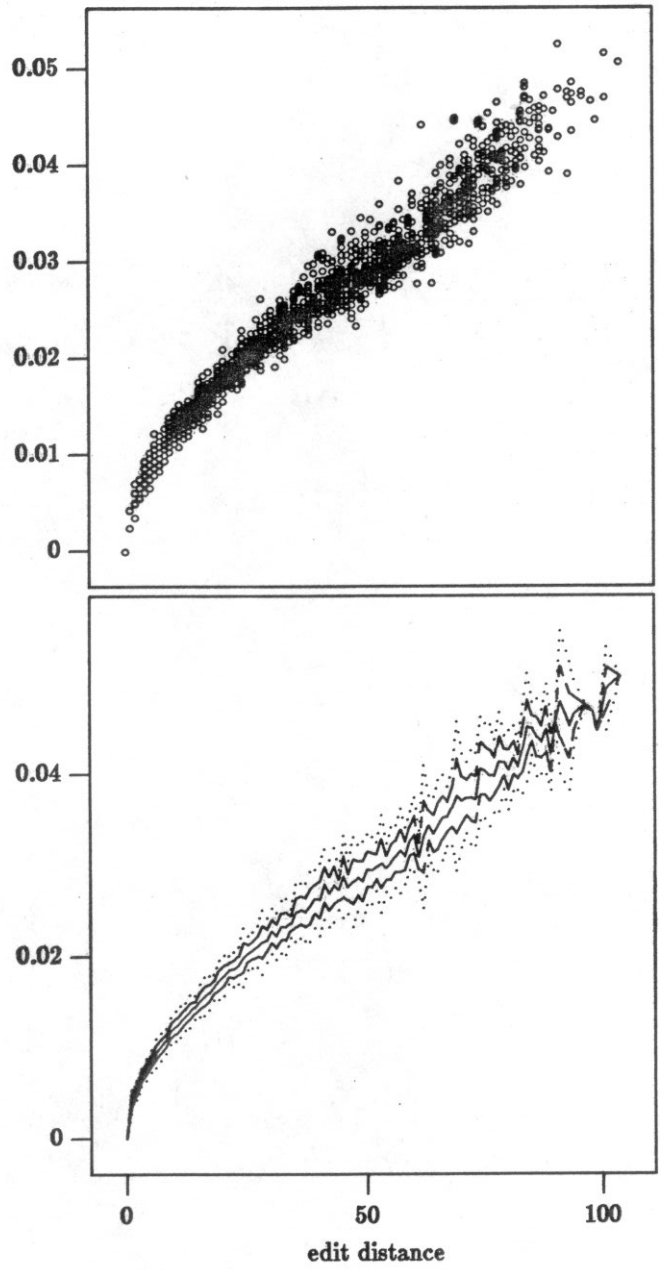


sequence length =200

best fit $e = 14966 * v^2$

Figure 5.3

Protein Sequence Pairs

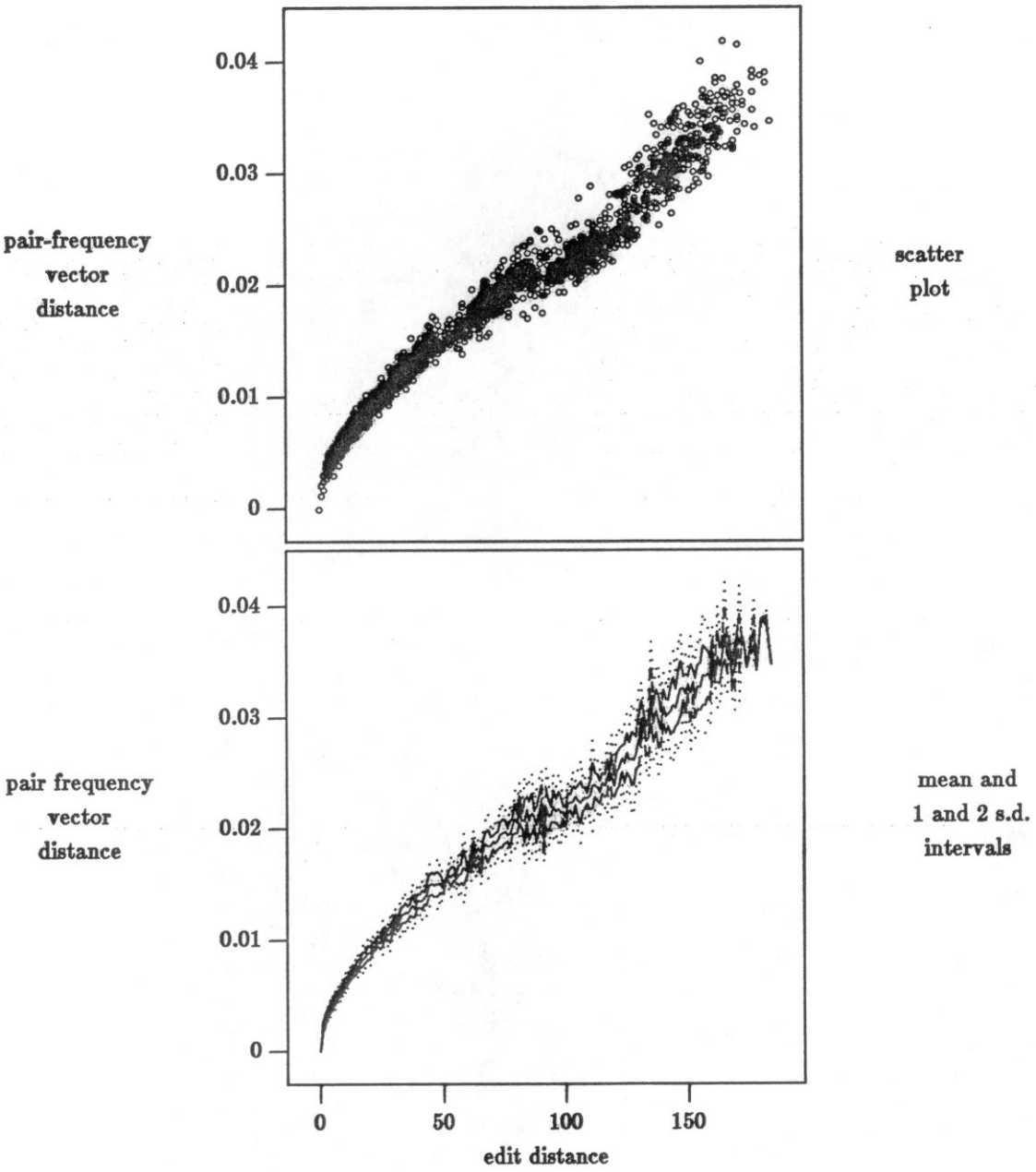


sequence length =400

best fit $e = 50586 * v^2$

Figure 5.4

Protein Sequence Pairs

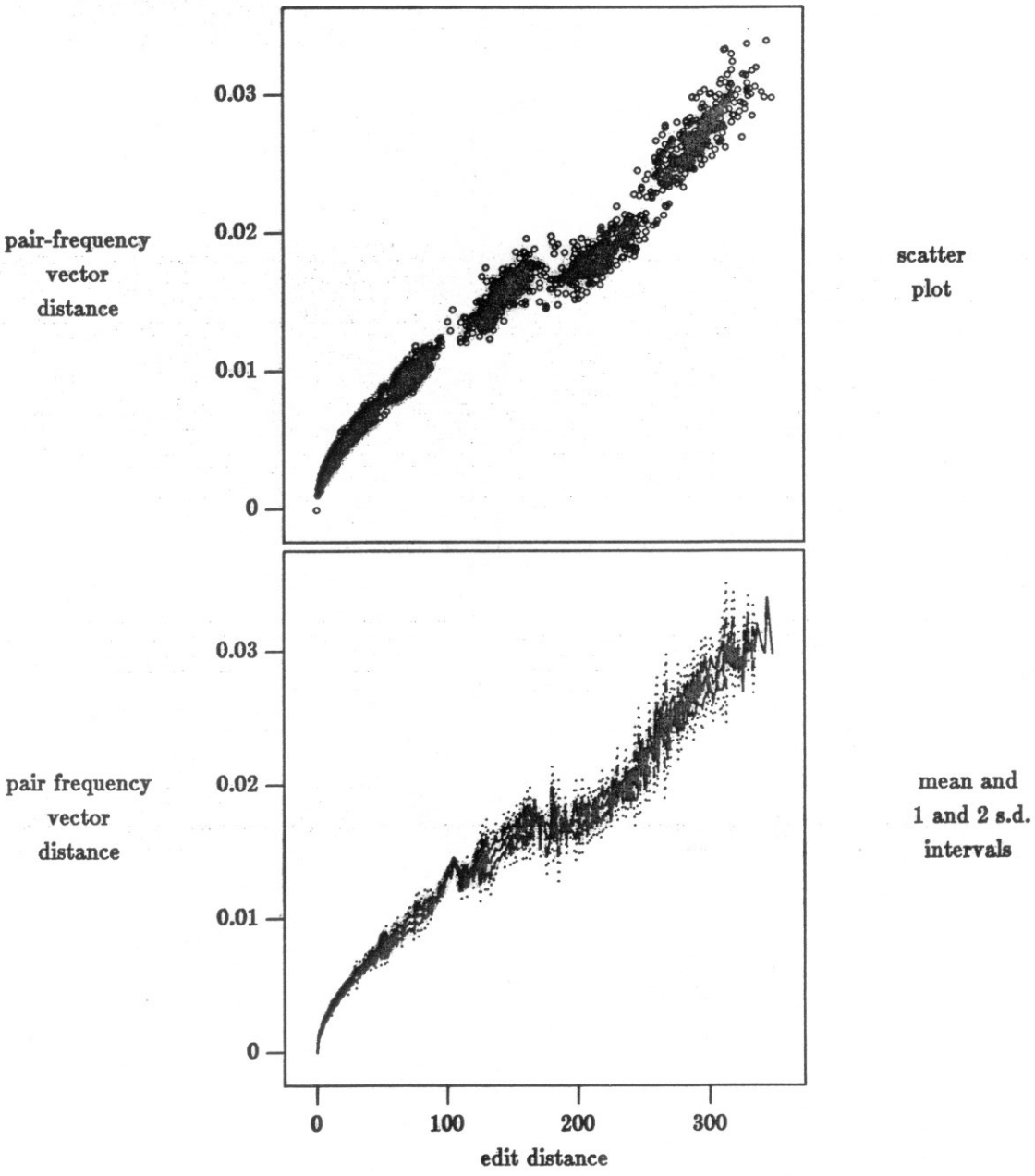


sequence length =800

best fit $e = 157957 * v^2$

Figure 5.5

Protein Sequence Pairs



sequence length =1600

best fit $e = .447711 * v^2$

Figure 5.6

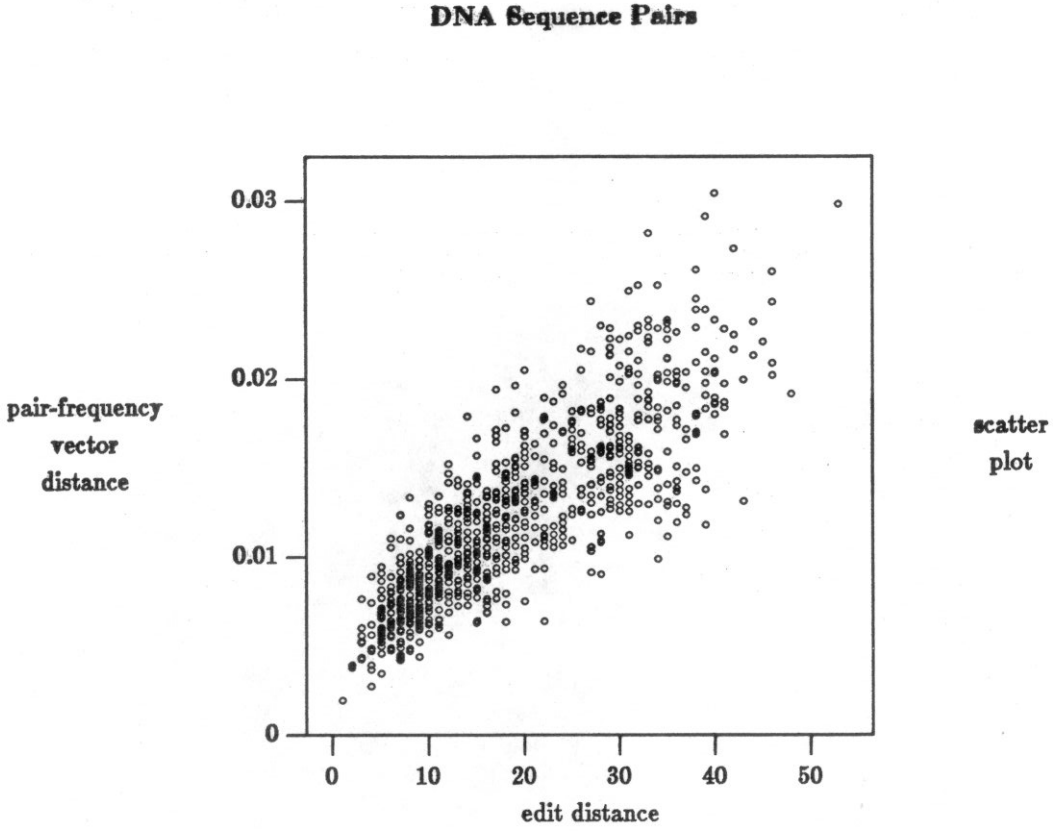

DNA Sequence Pairs
pair-frequency vector distance
0.03
0.02
0.01
0
0
10
20
30
40
50
edit distance
scatter plot
sequence length = 500

Figure 5.7

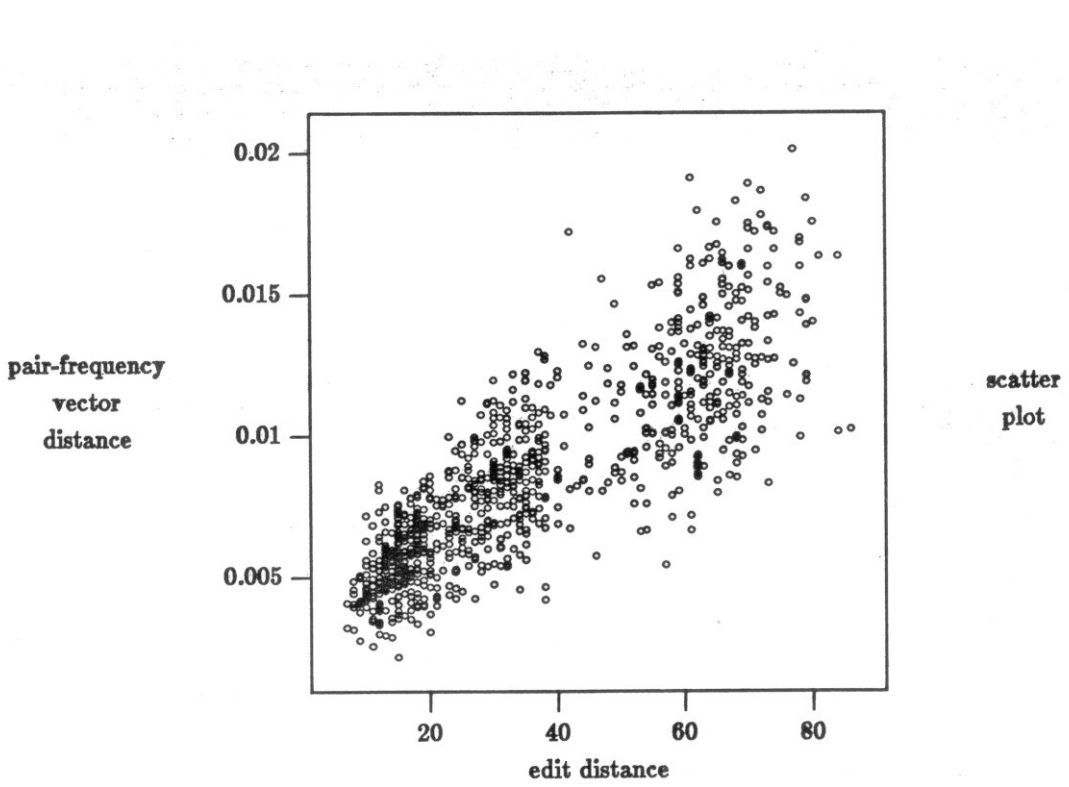
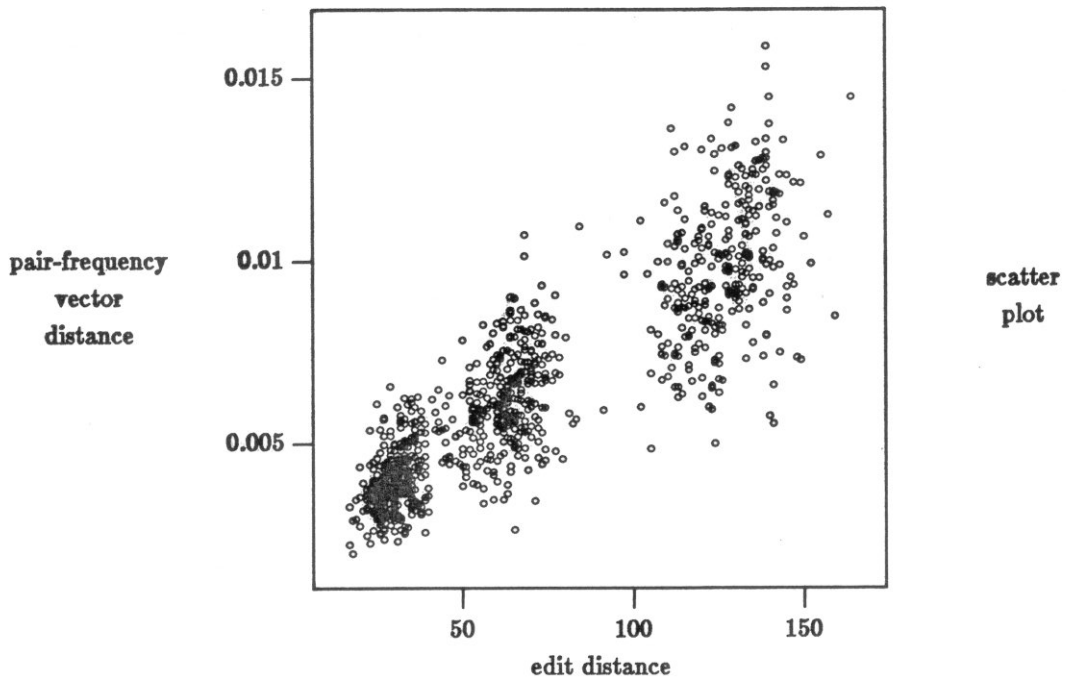

DNA Sequence Pairs
pair-frequency vector distance
0.02
0.015
0.01
0.005
20
40
60
80
edit distance
scatter plot
sequence length =1000

Figure 5.8

DNA Sequence Pairs



sequence length =2000

Figure 5.9

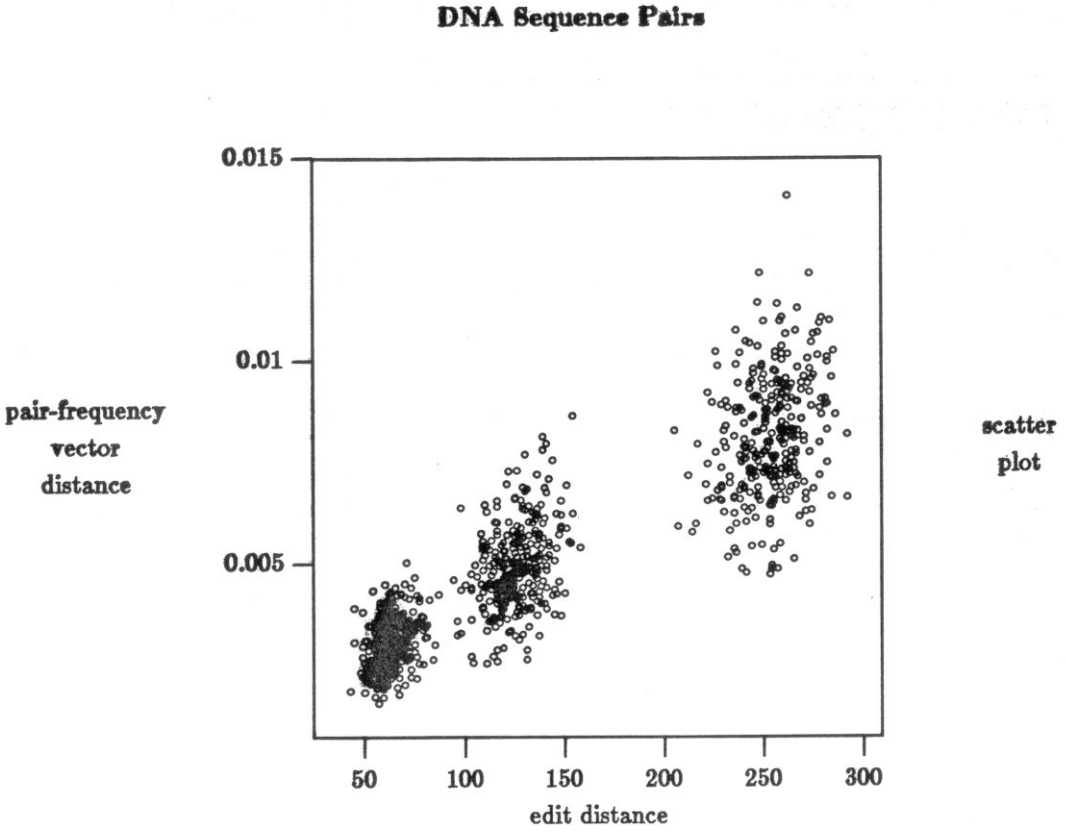
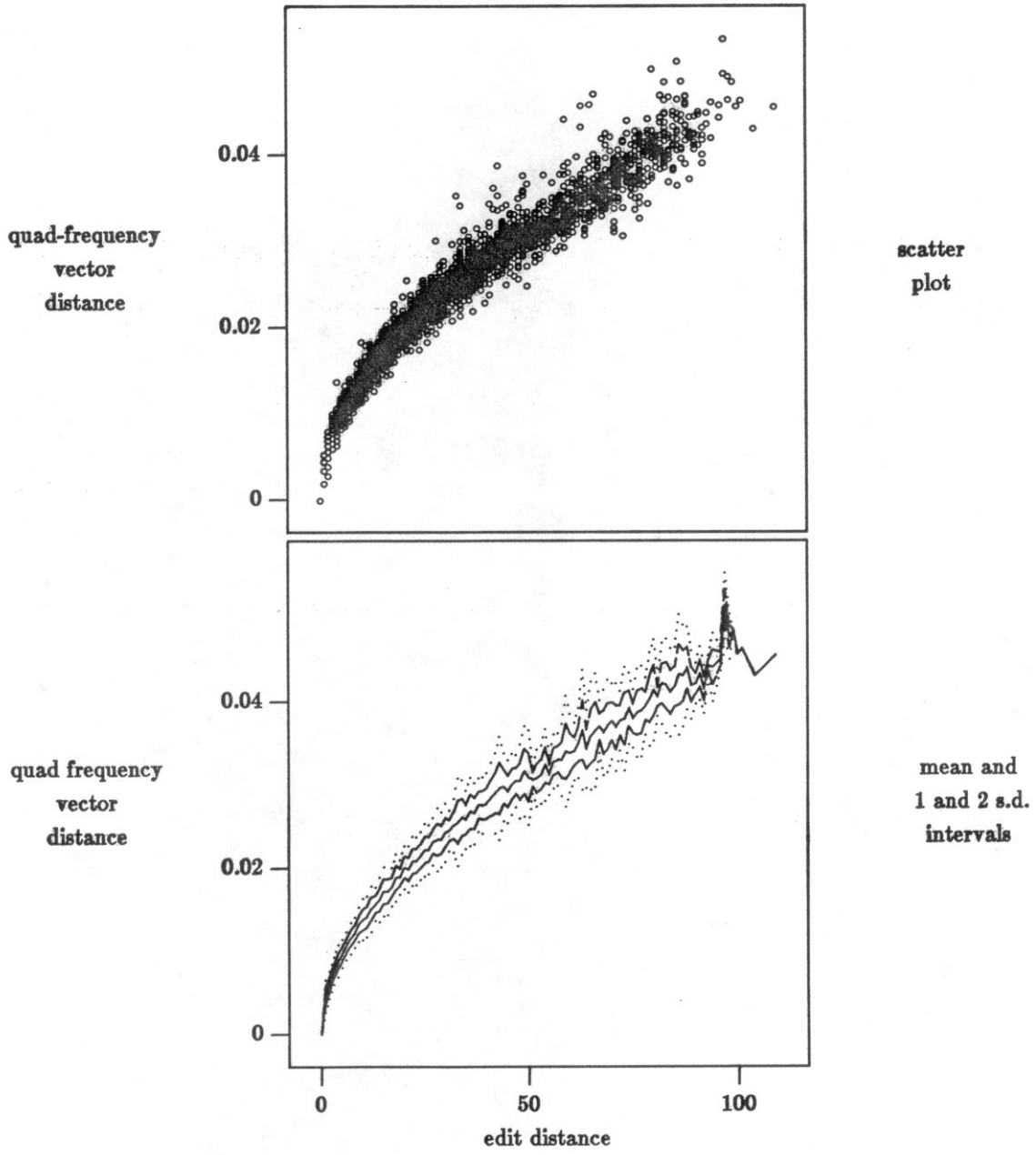

DNA Sequence Pairs
0.015
0.01
0.005
pair-frequency
vector
distance
50
100
150
200
250
300
edit distance
scatter
plot
sequence length =4000

Figure 5.10

DNA Sequence Pairs

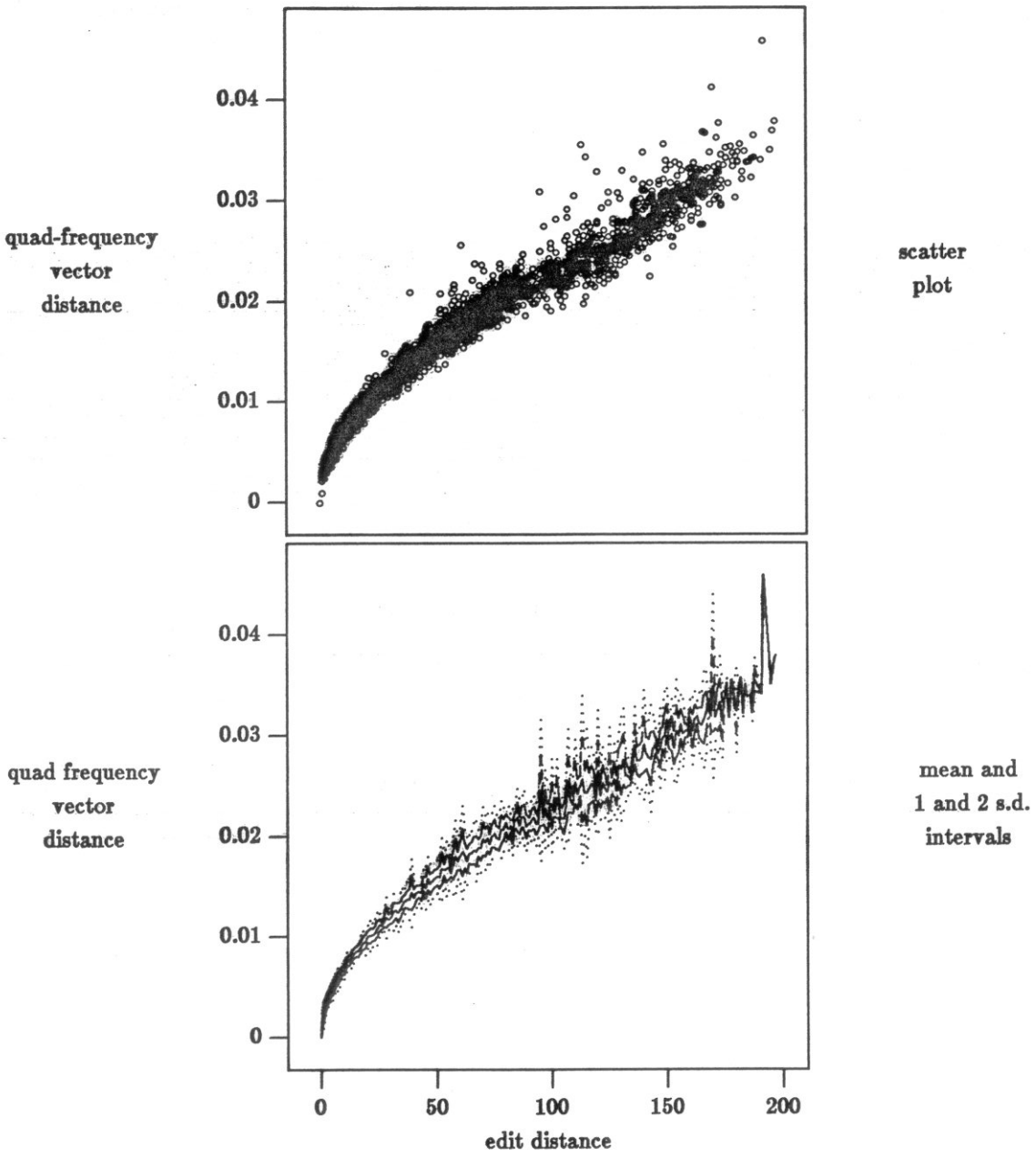


sequence length = 500

best fit $e = 48942 * v^2$

Figure 5.11

DNA Sequence Pairs

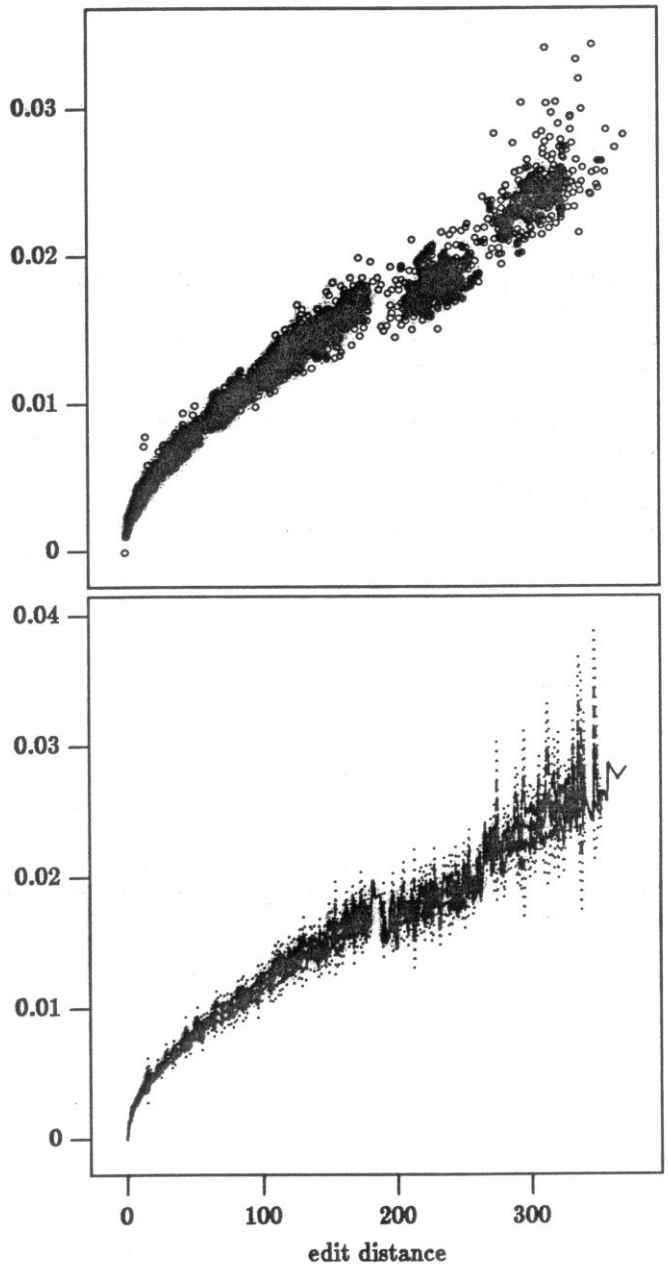


sequence length =1000

best fit $e = 173354 * v^2$

Figure 5.12

DNA Sequence Pairs

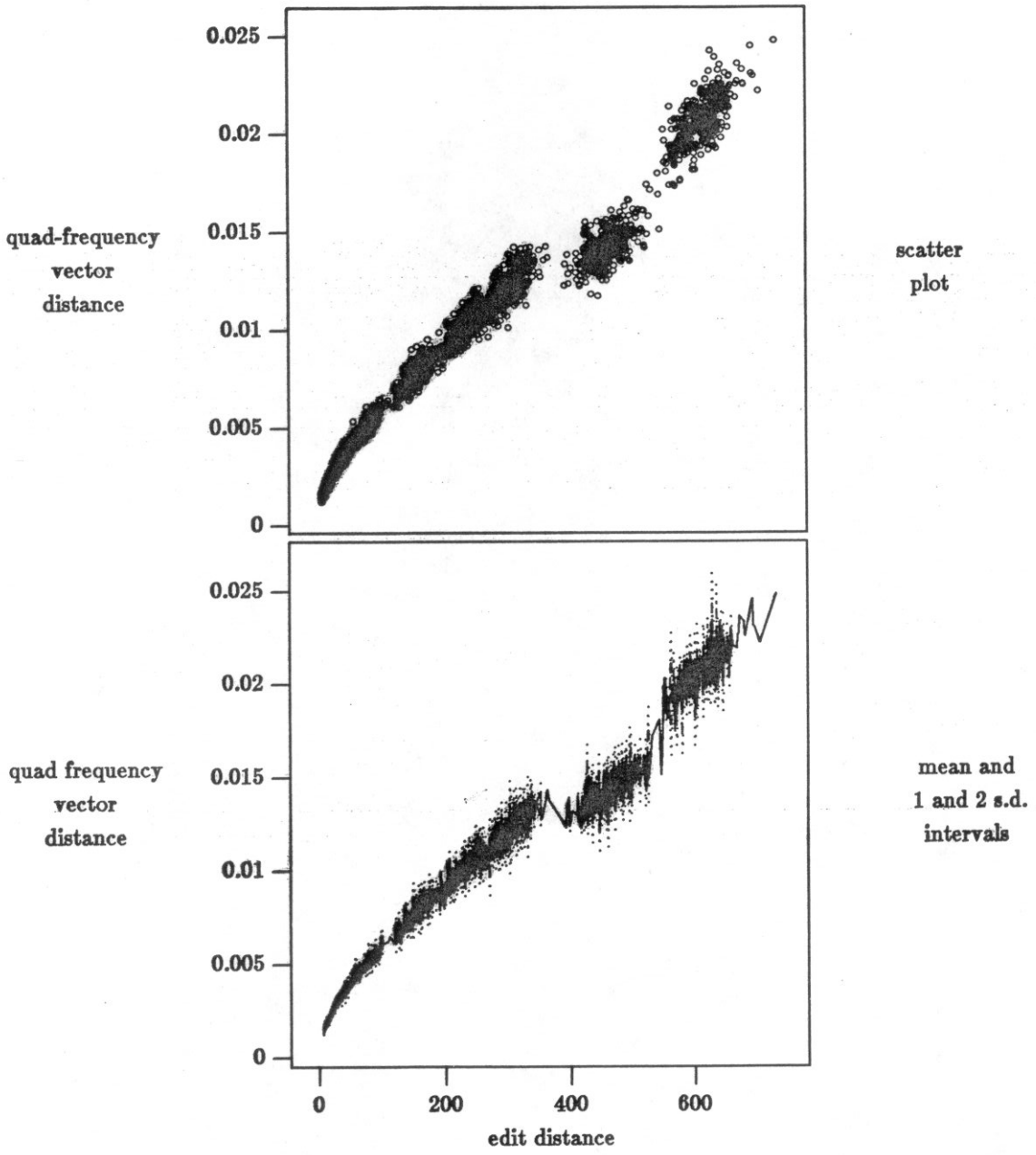


sequence length =2000

best fit $e = 55926 * v^2$

Figure 5.13

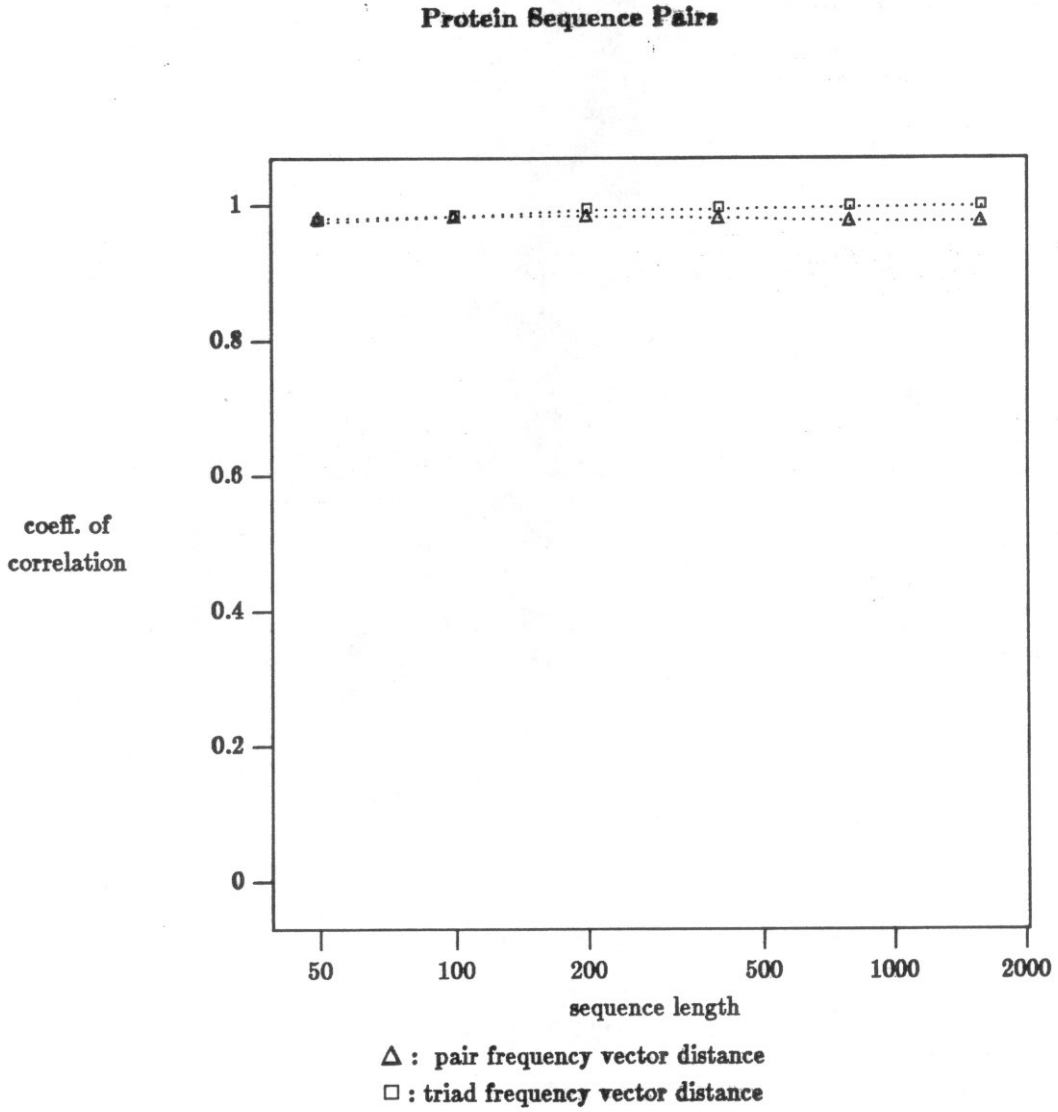
DNA Sequence Pairs



sequence length =4000

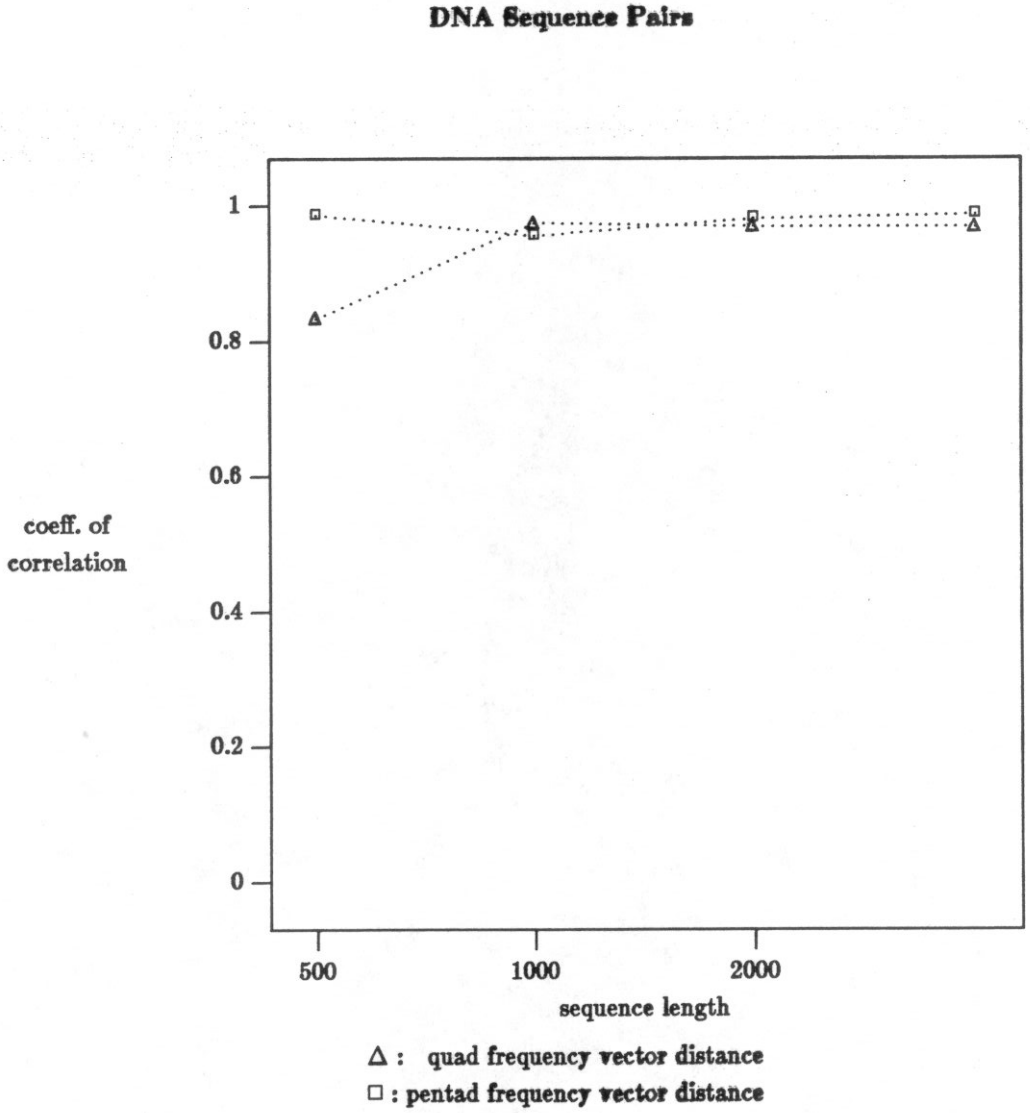
best fit $e = 1645557 * v^8$

Figure 5.14



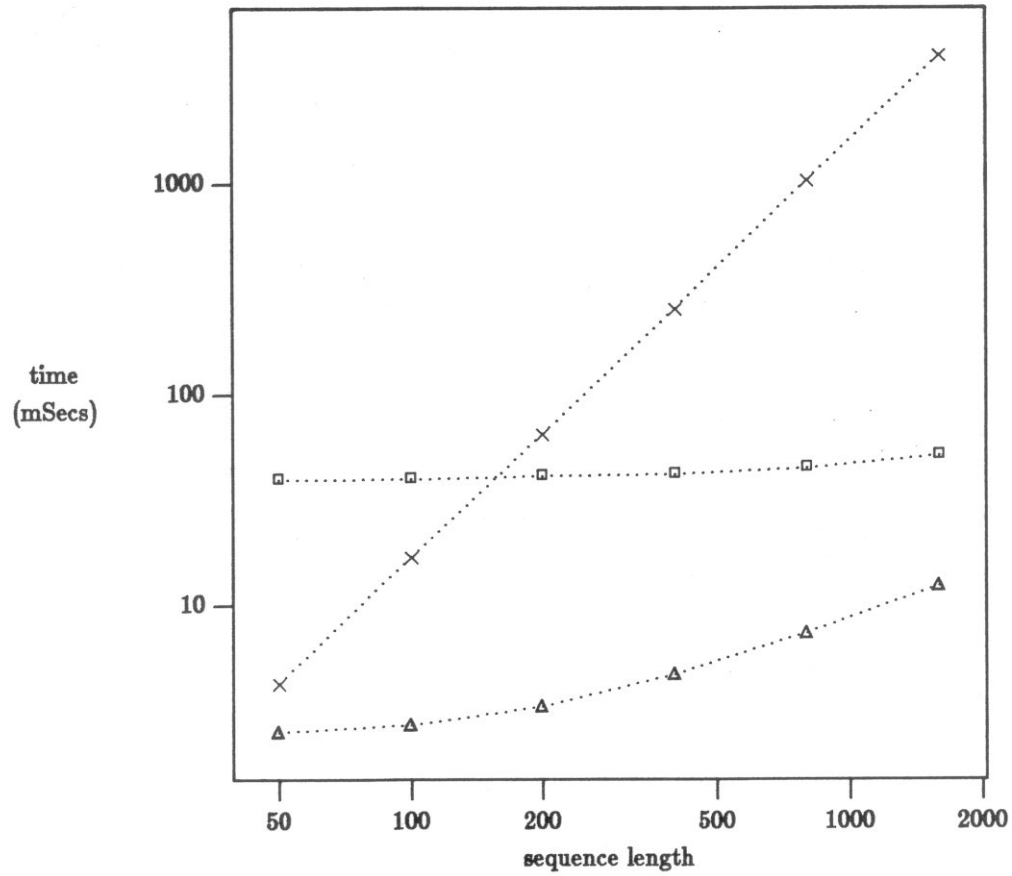
Coefficient of correlation between edit distance and vector distance
for pair- and triad- frequency vectors.

Figure 5.15



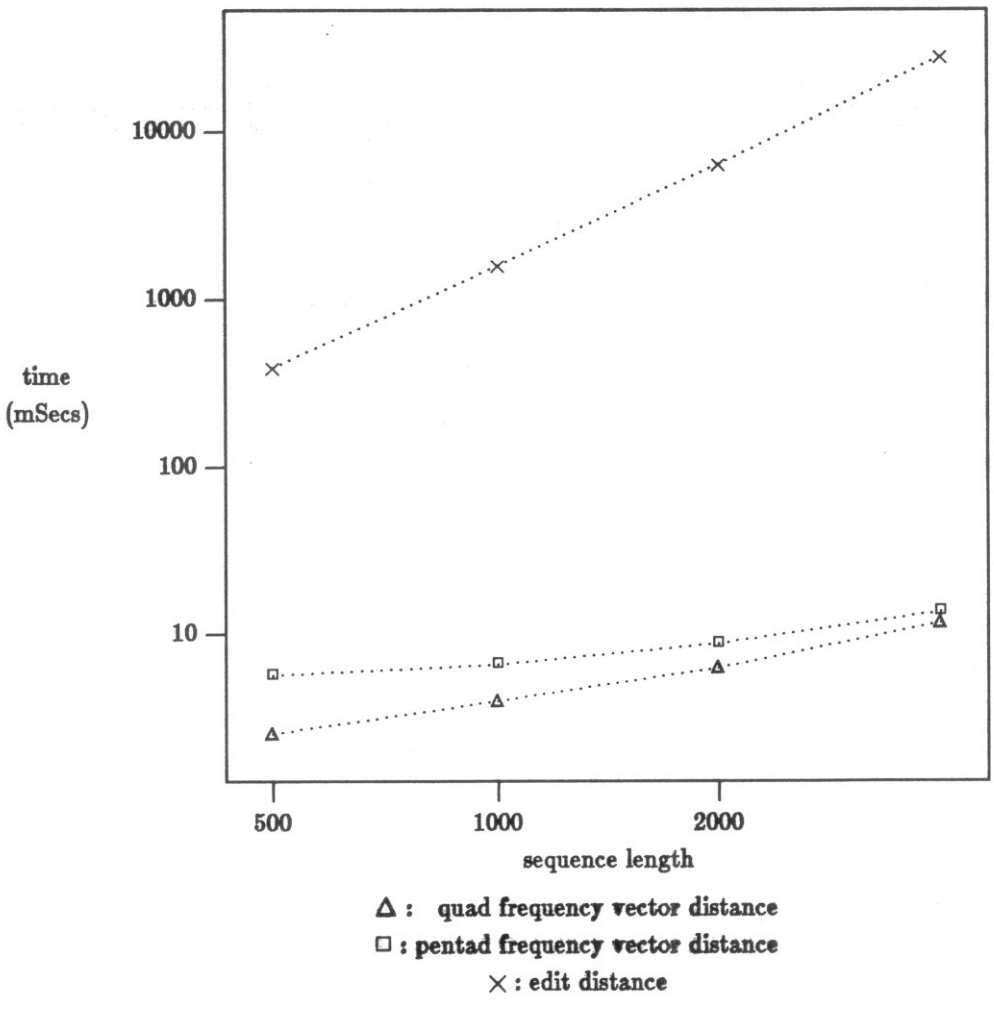
Coefficient of correlation between edit distance and vector distance for for quad- and pentad- frequency vectors.

Figure 5.16



Times to compute edit distance as well as
pair- and triad- frequency vector distances
for one pair of sequences of a given length.

Figure 5.17



Times to compute edit distance as well as
quad- and pentad- frequency vector distances
for one pair of sequences of a given length.

Figure 5.18

VI. Linear Projections of Sequence Vectors

In this chapter we shall examine the problem of dealing with more than two sequences For large groups of sequences the task of finding the edit distances between all pairs can prove to be infeasible, and even after converting the sequences to points by the pair or quad frequency vectors (for Protein and DNA sequences respectively), clustering them may still involve a quadratic number of distance calculations.

Projection preserves closeness

We observe that clustering a set of points on a line (a one-dimensional space) is fairly easy since it involves simply sorting the points along the line resulting in an effective ordering of the data with which we can answer many clustering-type questions by linearly scanning the sorted points. Queries such as finding all pairs of points that are within a given distance of each other, and so on, can thus be answered easily. Further, projecting a set of N points in d-space to a set of points on a line can be easily achieved by considering the projection to be a linear combination of the d vector components determining the point. Thus we have a technique which takes $O(N\log N+N\cdot d)$ time to convert a set of points into a simpler representation for handling clustering-type queries. We shall see that this leads to an effective technique for handling large sets of genetic sequences.

It is intuitively obvious that the closer two points are the closer their projections are likely to be. We may even think of a projection as a sort

of hashing done on the vectors - reducing them to a more easily handleable representation. Consider a series of linear combinations with randomly chosen coefficients. We may then contend that if two points are "close" their projections will almost always be "close" whereas if they are not, then their projections will be "close" only a small number of times. To develop this idea let us first look at two points.

Let $\overline{X}=[x_{1,}x_{2,}\ \cdots\ ,x_d]^T$

$\overline{Y}=[y_{1,}y_{2,}\ \cdots\ ,y_d]^T$

be two points in (the unit cube in) R^d.

and

Let $V=\left[\sum_{i=1}^{d}(x_i-y_i)^2\right]^{½}$

be the distance between them in the Euclidean norm.

Consider a random linear combination

$$\tilde{P}(\overline{X}) = \sum_{i=1}^{d} \tilde{\alpha}_i \cdot x_i$$

where the $\tilde{\alpha}_i$ are independent, identically distributed, normal random variables with mean $\mu = 0$ and variance σ^2

Let

$$\tilde{D} = \tilde{P}(\overline{X}) - \tilde{P}(\overline{Y})$$

ie.,

$$\tilde{D} = \sum_{i=1}^{d} \tilde{\alpha}_i (x_i - y_i)$$

Since the $\tilde{\alpha}_i$ are normal[†] and mutually independent, $\tilde{D}$ is also normal with

mean $\mu_D = 0$ and

variance $\sigma_D^2 = \sum_{i=1}^{d} \sigma^2 (x_i - y_i)^2 = \sigma^2 V^2$.

The density function of $\tilde{D}$ is :

$$f_D(x) = \frac{1}{\sigma V \sqrt{2\pi}} e^{-x^2/2\sigma^2 V^2}$$

The *distance* between the projected points is

$$|\tilde{D}| = |\tilde{P}(\bar{X}) - \tilde{P}(\bar{Y})|$$

The distribution of the projected lengths is therefore

$$f_{|D|}(x) = \frac{2}{\sigma V \sqrt{2\pi}} e^{-x^2/2\sigma^2 V^2} \qquad \text{for } x \geq 0$$

The mean projected length is then

$$\mathrm{E}[|\tilde{D}|] = \frac{2}{\sigma V \sqrt{2\pi}} \int_0^{\infty} x\, e^{-x^2/2\sigma^2 V^2}\, dx$$

[†]Note that even if the $\tilde{\alpha}_i$ were *not* normal, $\tilde{D}$ would still approach a normal random variable by the central limit theorem.

$$= \frac{2}{\sigma V\sqrt{2\pi}}\ \sigma^2 V^2 \int_0^\infty y\ e^{-y^2/2} dy \qquad \text{Substituting} \quad y = x/\sigma V$$

$$= -\frac{2\sigma V}{\sqrt{2\pi}}\ e^{-y^2/2}\Big|_0^\infty$$

$$= \sqrt{2/\pi}\ \sigma V \qquad (6.1)$$

Let us now consider, rather than the actual distance between the projected points $\tilde{P}(\bar{X})$ and $\tilde{P}(\bar{Y})$, the probability that they fall within some threshold distance T of each other. Naturally we expect that this probability will increase with T and decrease with V. Our aim is to use the value of T, which we are free to chose, and the observed number of times that the two points, $\bar{X}$ and $\bar{Y}$ project to within T of each other, as an estimate of their actual distance V.

Knowing that

$$f_{|D|}(x) = \frac{2}{\sigma V\sqrt{2\pi}}\ e^{-x^2/2\sigma^2V^2} \qquad \text{for} \quad x \geq 0$$

The probability that the projected distance is within the threshold is:

$$\text{Prob}\ [|\tilde{D}| < T] = \frac{2}{\sigma V\sqrt{2\pi}} \int_0^T e^{-x^2/2\sigma^2V^2}\, dx$$

$$= \frac{2}{\sqrt{2\pi}} \int_0^{T\ /\ \sigma V} e^{-y^2/2} dy \qquad \text{Substituting} \quad y = x/\sigma V$$

$$= \frac{1}{\sqrt{2\pi}} \int_{-T/\sigma V}^{+T/\sigma V} e^{-y^2/2} dy$$

$$= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+T/\sigma V} e^{-y^2/2} dy \quad - \quad \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-T/\sigma V} e^{-y^2/2} dy$$

$$= \Phi\,(T/\sigma V) - (\,1 - \Phi\,(T/\sigma V))$$

$$= 2\,\Phi\,(T/\sigma V) - 1 \qquad (6.2)$$

Where

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2/2} dy$$

is the cumulative distribution function of the standard, or unit, normal random variable.

This quantifies the intuitive idea that as we increase the threshold of closeness T, the probability of two sequence-vectors projecting together increases regardless of their distance. Also, as we consider increasingly distant pairs (increasing V) the probability of their projecting together within a given threshold decreases. Note however that in practice the $\tilde{\alpha}_i$ may not be normal but perhaps uniformly distributed in some given range. In such a case the above expression is merely an approximation to the actual probability based on the law of large numbers. In particular, for the case of uniform random variables, the actual probability will be higher than that indicated above as T approaches the range of the uniform distribution. Also V cannot exceed a specific value since all the sequence-

vectors are normalised and hence constrained to be within a unit cube in $\boldsymbol{R}^d$.

However, while this is of significance when considering just two sequences or even a small number, it is not immediately applicable to large sets of sequences. This is due to the fact that we may have to perform many linear combinations and score each pair by the number of times they project together and this scoring process itself can be quadratic in the size of the database. This may be unacceptable for large databases. We present a method to overcome this in the next section.

Clustering Based on Projections

Given a vector $\overline{X} = [x_1, x_2, \cdots, x_d]^T$, we defined a random linear combination $\tilde{P}$ acting on $\overline{X}$ as

$$\tilde{P}(\overline{X}) = \sum_{i=1}^{d} \tilde{\alpha}_i \cdot x_i$$

where the $\tilde{\alpha}_i$ are independent, identically distributed, random variables. Thus, an instance of $\tilde{P}$ is an instance of each of the $\tilde{\alpha}_i$.

Let us now perform k random linear combinations $\tilde{P}_1, \tilde{P}_2, \cdots, \tilde{P}_k$ on the entire set of sequence vectors, resulting in k sets of points on the line ($\boldsymbol{R}^1$)

Given a *window size*, λ, we can divide the line into adjacent non-overlapping widows of size λ:

$$\cdots [-\lambda, 0), [0, \lambda), [\lambda, 2\lambda), \cdots$$

Thus it is reasonable to talk about the *window into which vector* $\overline{X}$ *projects under linear combination* $\tilde{P}$, which we shall define as :

$$W(\tilde{P},\bar{X}) = \left[\left\lfloor \frac{\tilde{P}(\bar{X})}{\lambda} \right\rfloor, \left\lfloor \frac{\tilde{P}(\bar{X})}{\lambda} \right\rfloor + 1 \right)$$

where $[a,b) = \{ x \mid a \leq x < b \}$

Thus for each of the projections $\tilde{P}_i$ we may define an equivalence relation $\sim_i$ as :

$$\bar{X} \sim_i \bar{Y} \text{ iff } W(\tilde{P}_i,\bar{X}) = W(\tilde{P}_i,\bar{Y})$$

Thus we have k equivalence relations.

It is intuitively obvious that the closer two vectors $\bar{X}$ and $\bar{Y}$ are, the greater the number of relations in which they will be in the same equivalence class. It is also obvious that finding all the equivalence classes in a given relation can be done easily by simply sorting the projected points. Let us now attempt to quantify the probability that two vectors $\bar{X}$ and $\bar{Y}$ "match" (fall in the same equivalence class).

Let $v = |\bar{X} - \bar{Y}|$ be the euclidean distance between $\bar{X}$ and $\bar{Y}$.

Let p_i = Prob $\{ \bar{X} \sim_i \bar{Y} \}$

clearly $p_1 = p_2 = \cdots = p$

Let $q = 1-p =$ Prob{no match}

Let $\tilde{D} = |\tilde{P}(\bar{X}) - \tilde{P}(\bar{Y})|$ and let $P_D(x)$ be its probability density function.

$$\text{Prob}\{\text{no Match} \mid D = x\} = \begin{cases} \frac{x}{\lambda} \text{ if } x < \lambda \\ 1 \text{ if } x \geq \lambda \end{cases}$$

Therefore,

$$q = \int_0^{\lambda} \frac{x}{D} P_D(x)\, dx + \int_{\lambda}^{\infty} P_D(x)\, dx$$

$$\leq \int_0^{\infty} \frac{x}{D} P_D(x)\, dx$$

$$= \frac{E[D]}{\lambda}$$

$$\approx \frac{\sqrt{2/\pi}\,\sigma v}{\lambda}$$

where σ is the standard deviation of the coefficients $\tilde{\alpha}_i$.

Hence,

$$p \geq \left(1 - \frac{\sqrt{2/\pi}\,\sigma v}{\lambda} \right)$$

We may therefore look upon the number of projections in which $\bar{X}$ and $\bar{Y}$ match as a binomial random variable (series of Bernoulli trials) with parameter (k,p). For large k we will approximate this with a normal random variable with mean kp and variance kpq.

So, Prob{ $\geq i$ matches out of k}

$$\approx \frac{1}{\sqrt{2\pi kpq}} \int_i^{\infty} e^{-(x-kp)^2/2kpq}\, dx$$

$$= \frac{1}{\sqrt{2\pi}} \int_{\frac{i-kp}{\sqrt{kpq}}}^{\infty} e^{-y^2/2}\, dy \qquad \text{where } y = \frac{x-kp}{\sqrt{kpq}}$$

$$= 1 - \Phi\left[\frac{i-kp}{\sqrt{kpq}}\right] \tag{6.3}$$

Similarly Prob{ $\leq i$ matches out of k }

$$= \Phi\left[\frac{i-kp}{\sqrt{kpq}}\right]$$

Thus, for example, if we chose k=100 and p=0.8 then solving for i such that Prob{ $\geq i$ matches } $\geq$ 0.9 gives us $i \leq 75$. Then solving for p such that Prob{$\leq$ 75 matches } $\geq$ 0.9 gives $p \leq 0.65$. What this means is that if we define two "threshold" values δ_1 and δ_2 such that $p = \left[1 - \frac{\sqrt{2/\pi}\,\sigma\delta_1}{\lambda}\right] \geq 0.8$ and $\left[1 - \frac{\sqrt{2/\pi}\,\sigma\delta_2}{\lambda}\right] \leq 0.65$, then we may state with probability 90% that if two sequence vectors $\bar{X}$ and $\bar{Y}$ are such that $||\bar{X}-\bar{Y}|| < \delta_1$ then $\bar{X}$ and $\bar{Y}$ will match at least 75 times, and if $||\bar{X}-\bar{Y}|| > \delta_2$ then $\bar{X}$ and $\bar{Y}$ will match at most 75 times.

In other words, by counting only those pairs that match at least 75 times we can obtain almost all "close" or "good" (closer than δ_1) pairs while eliminating almost all "far" or "bad" pairs (farther than δ_2), although we are unsure of intermediate pairs.

Now while it is very easy to locate pairs that match in *all* projections, it is not so easy to locate those that match in some fraction. To find all the vectors that match in all the projections we create for each vector $\bar{X}$, an "identifier", $= \sum_{i=1}^{k} n_i \cdot w^{i-1}$. Where each n_i is the number of the window into which vector $\bar{X}$ falls under projection $\tilde{P}_i$, and w is the maximum

number of possible windows.[†] Then sorting the entire set of identifiers will bring together all vectors that match in all projections. Thus we can find vectors that match in all projections in $O(D\log D)$ time where D is the number of sequences.

However, it is reasonable to expect that if two vectors match in many, but not all, projections, then, if we take a small subset of the projections they are likely to match in all projections in that subset. This, coupled with our previous observation suggests a fast method for finding close pairs of vectors.

Let us consider c randomly chosen linear combinations. What is the probability of two vectors $\bar{X}$ and $\bar{Y}$ matching in all of them? (We shall refer to such an event as a "*c–match*"). Obviously

$$\text{Prob}\{\text{ c-match }\} = p^c$$

$$\approx \left(1 - \frac{\sqrt{2/\pi}\,\sigma v}{\lambda}\right)^c$$

Performing a c-match instead of just one projection has the advantage of reducing the number of pairs we must deal with, and also reduces the likelihood of matching vectors that are far apart. However, to ensure that we do not lose too many close pairs, it may be necessary to perform several c-matches and consider pairs that match in at least one or some fraction of them. This may not be as expensive as doing the same for several ordinary projections ("1–*matches*") since by our choice of c we can significantly reduce the number of false matches. Thus we may

[†]For example, we could call $[0, \lambda)$ window #1, $[-\lambda, 0)$ window #2 etc $\cdots$. If we chose the coefficients $\tilde{\alpha}_i$ to be in a given range then we are guaranteed to have a finite range for the projected points, and hence a fixed number of windows (depending only on λ). Thus the value of w is defined.

reasonably expect that most of the computation time is spent on pairs that will turn out to be interesting. It must also be noted that the risk of finding false matches is very real, since, of all the possible pairs of sequence vectors from a large database (approaching the millions) an overwhelming number of them will be far apart. Thus even if only a few of these distant pairs are maatched tghis can still be a large number of false matches. Our aim therefore is to reduce, as much as possible, the number of false matches, while at the same time, losing as few as possible of the good pairs. There is obviously a tradeoff involved, which is why this method can at best be a filter, reducing the number of pairs to be considered perhaps by hand or by slower but more exhaustive algorithm.

For example:

If we select p=0.8 as our upper threshold (pairs that have a greater than 80% chance of matching are considered definitely interesting); and a lower threshold of p=0.2 (sequences with a less than 20% chance of matching are considered definitely uninteresting), then, taking c=3 we will still retain $(0.8)^3 \approx 51\%$ of the "good" pairs while retaining only $(0.2)^3 \approx 1\%$ of the "bad" pairs.

If we now perform say N=3 such c-matches and retain pairs that match in at least one of them then we will obtain $1 - (1 - 0.8^3)^3 \approx 89\%$ of the good pairs while getting only $1 - (1 - 0.2^3)^3 \approx 2.5\%$ of the bad ones.

Thus, while decreasing c or increasing N will increase the number of good pairs located, it will also result in an increase in the number of bad matches, and vice versa, so we need to choose c and N appropriately to balance these two effects .

Figures 6.1-6.4 show, for N=1, how the number of pairs which we find varies with c as well as with λ, for DNA and for Protein sequences. Also, as an upper bound, we can assume that the number of pairs to be considered will increase at most linearly with N. Thus we see that by our choice of parameters we can, in almost linear time ($O(D\log D)$) reduce the number of pairs to be considered from $O(D^2)$ to a more manageable size. The figures also show the average size of the clusters (equivalence classes) found. We are able to talk of equivalence classes in this case because N=1 and $c-match$ is an equivalence relation, whereas one c-match out of N>1 is not an equivalence relation and it is therefore more difficult to define a concept of clusters on them — the best we may be able to do may be perhaps to find all the pairwise distances and apply weighted graph clustering techniques.

Figures 6.5-6.7 shows the relation between the probability of two vectors projecting together, p, which, as we have seen, depends on their Euclidean distance, and the expected percentage of pairs within that distance of each other that we actually locate (= probability of at least one c-match out of N for various c and N.

Indexing based on projections

The ease of handling large numbers of sequences by the linear projection technique can be applied to indexing-type queries too. Typically the

type of question asked would be of the form: given a database of known sequences, how many of them are related to a particular query sequence, which is perhaps not in the database. This sort of search is very important, for example, when a new piece of DNA is sequenced and one wishes to compare it with the known sequences. This section describes a technique, based on the process of projecting sequence vectors described above, to handle such queries. This technique has also been used to write a sequence indexing system. A description of this system appears in Appendix I.

As established in the previous section, if we were to take the set of sequence vectors and apply a linear combination with random coefficients to the components of each vector then we would expect that close sequence vectors would project closely together. In particular, if we were to look at a particular sequence and see where it projects, then we would expect that its neighbors along the projected line would be closely related sequences.

Given a non-negative real valued parameter w and two sequence vectors $\overline{X}$ and $\overline{Y}$ we say that $\overline{X}$ and $\overline{Y}$ (or their respective sequences) *w-match* under an arbitrary linear combination $\tilde{P}$, or that $\overline{Y}$ is in the *w-neighborhood* of $\overline{X}$ under $\tilde{P}$ if

$$|\tilde{P}(\overline{X}) - \tilde{P}(\overline{Y})| \leq W$$

where $\tilde{P}$, $\overline{X}$ and $\overline{Y}$ are defined as before.

Once again let us consider k random linear combinations $\tilde{P}_1, \tilde{P}_2, \cdots, \tilde{P}_k$ on the entire set of sequence vectors, plus another sequence vector $\overline{X}$ corresponding to the sequence we are interested in (the query sequence). Thus we have k subsets of the original set of sequences,

corresponding to the k w-neighborhoods of $\bar{X}$, one for each projection. Once again it should be evident that a sequence that appears in many of these sets is more likely to be closely related to the query sequence than ones that appear only in a few of these sets or not at all.

Let $v = |\bar{X} - \bar{Y}|$ be the Euclidean distance between $\bar{X}$ and $\bar{Y}$, a sequence vector from the original data set.

Let $p_i(w)$ = Prob { $\bar{Y}$ is in the w-neighborhood of $\bar{X}$ under projection i }. Clearly $p_1(w) = p_2(w) = \cdots = p(w)$. Now $p(w) = 2\,\Phi\,(w/\sigma v) - 1$ by equation 6.2

Thus if we were to look at all sequence vectors which project into at least m of the k w-neighborhoods above, the probability of $\bar{Y}$ being in this set is :

$$= 1 - \Phi\left[\frac{m - k \cdot p(w)}{\sqrt{k \cdot p(w) \cdot q(w)}}\right]$$

where $q(w) = 1 - p(w)$

This can be derived using much the same reasoning used to derive equation 6.3. This probability can be seen to increase with w and k and decrease with m and v

This suggests the idea that if we were to take a set of sequence vectors and another arbitrary sequence vector, we could project all of them together and count the number of sequences that fall within some distance of the query sequence in some minimum number of projections in order to determine a "cluster" around the query sequence. Further we note that if we are dealing with a fixed database of sequences, then we need not perform the projections over and over again. We need only choose the k random linear combinations in advance and do the projections of the entire

set of sequences in advances. Each projection is then sorted according to numeric (scalar) value of the projected point. Then when given a query sequence we can project it into each of these existing projections. To find its neighborhood set, first perform a binary search among the sorted projected points from the original set to find the closest one. Then, search the sorted list in both directions from this position to find all the points that lie within the neighborhood, and so on for each projection.

This observation leads to a simple implementation of a sequence indexing system, described in Appendix I. Some pre-processing time is necessary, proportional to the size of the database, D, and the number of projections k. But processing the query sequence is then very fast, since the number of operations performed is $O(k \log D)$ to perform the binary search in each projection plus an amount of time depending on the "window" size w, which affects the number of sequences ultimately found or the "cluster" size. Also the parameters w and m can be used to tune the search and discard as much or as little of the original sequence databank as desired.

Figures 6.8 - 6.15 show the results obtained by querying a database of known sequences (the GenBank Database) with a series of randomly generated query sequences, for various choices of window size w. In this implementation the number of projections k was chosen to be 100. In the case of DNA sequences (figures 6.8 -6.11) the database consisted of approximately 5000 sequences and in the case of protein sequences it consisted of approximately 13000 sequences. Each plot represents one choice of w ranging from 50 to 200. Each plot shows, for each of the query sequences, the number of sequences out of the total database of sequences

that were found to be in the neighborhood of the query sequence at least m times for m ranging from 0 to 100. As can be seen, in general raising the value of w causes a greater number of sequences to be found and raising the value of m causes this number to decrease. However the plots show a "waterfall" shape suggesting that for any query sequence there is a small range of values of m within which there is a great change in the proportion of database sequences found and thus manipulating the value of m in this range can give a method to tune the set of sequences found to whatever is considered a desirable size.

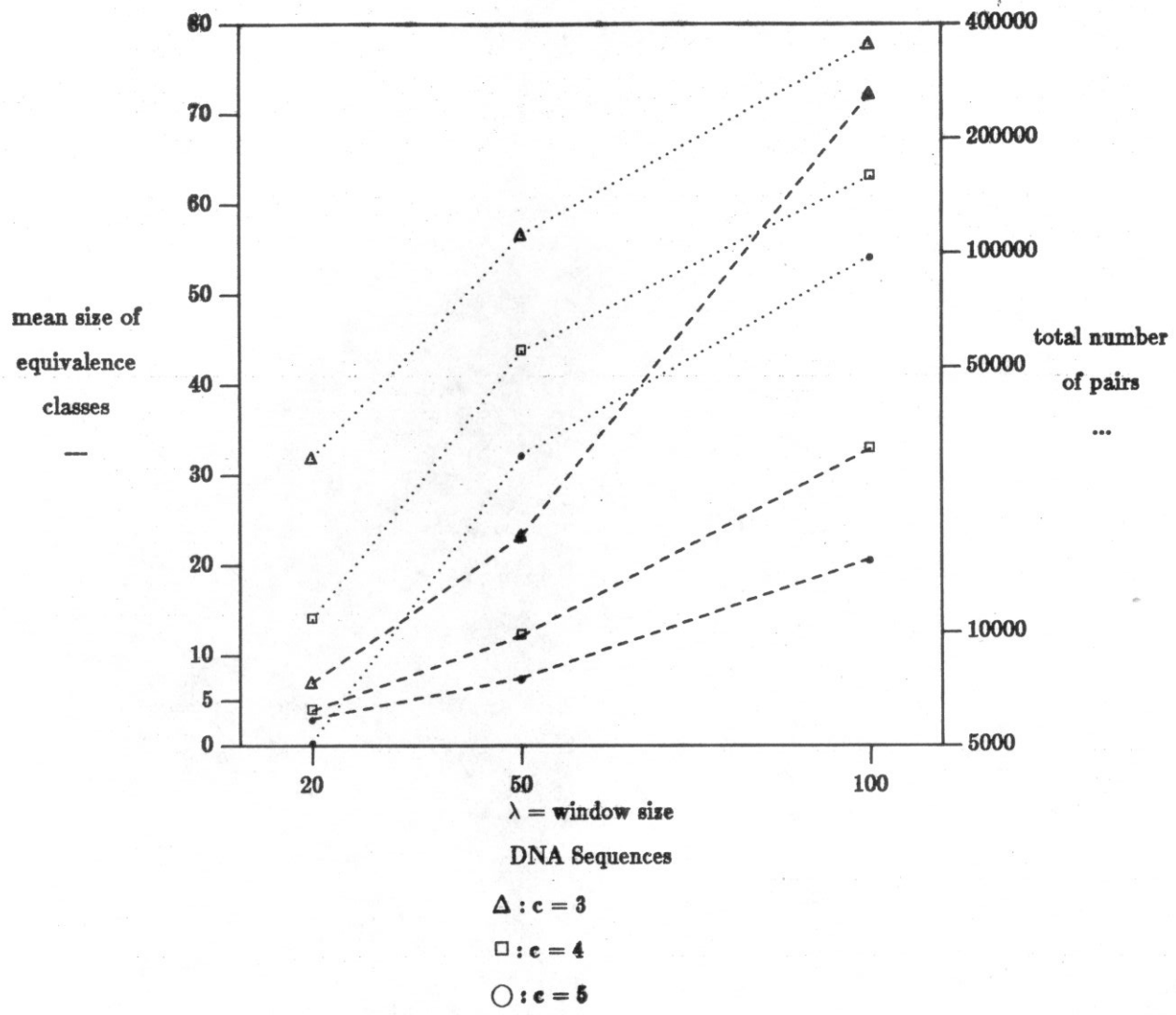

mean size of
equivalence
classes
—
80
70
60
50
40
30
20
10
5
0
400000
200000
100000
50000
10000
5000
total number
of pairs
...
20
50
100
λ = window size
DNA Sequences
Δ : c = 3
□ : c = 4
○ : c = 5

Fig. 6.1

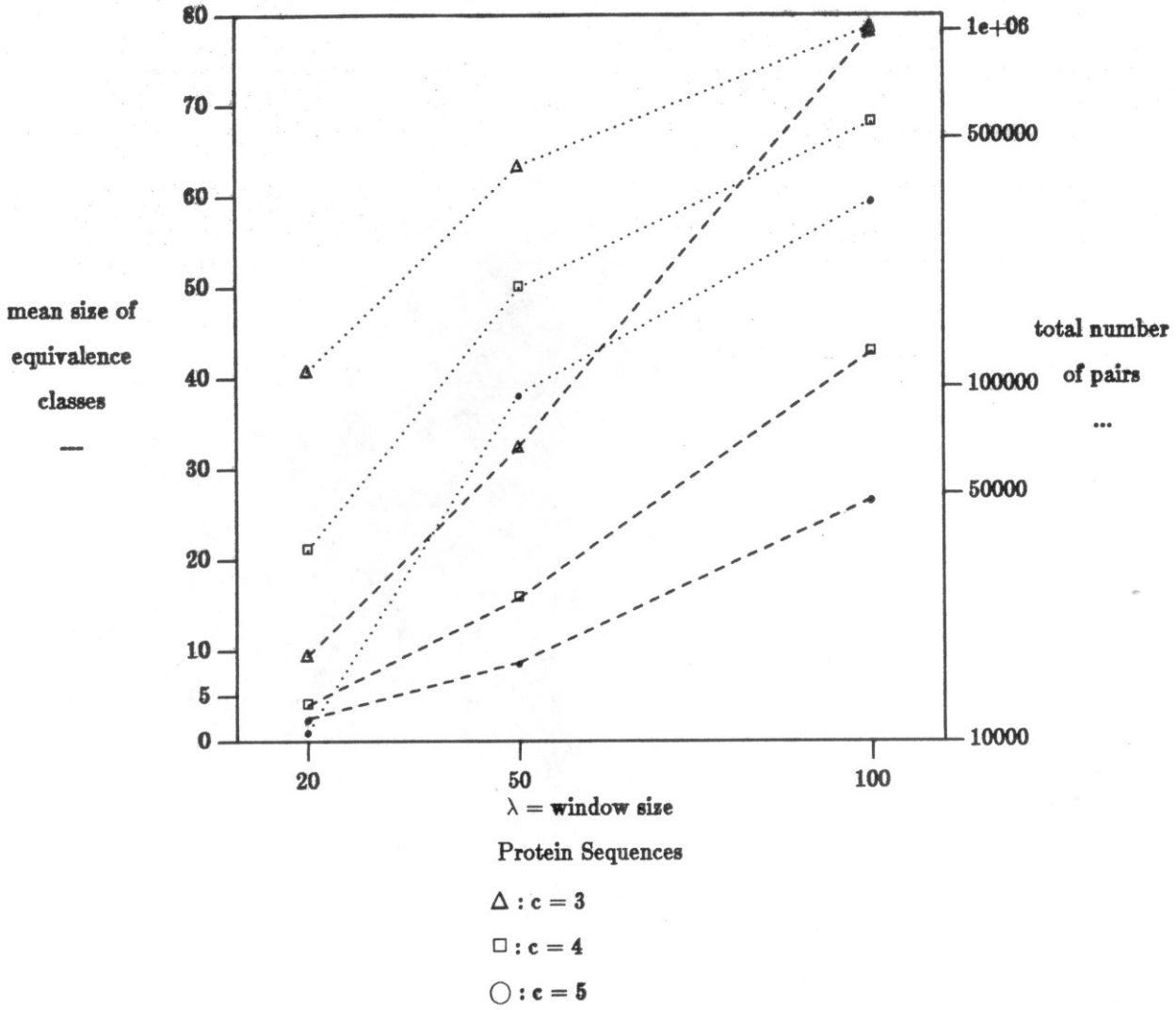

mean size of
equivalence
classes
—
total number
of pairs
...
80
70
60
50
40
30
20
10
5
0
1e+06
500000
100000
50000
10000
20
50
100
λ = window size
Protein Sequences
Δ : c = 3
□ : c = 4
◯ : c = 5

Fig. 6.2

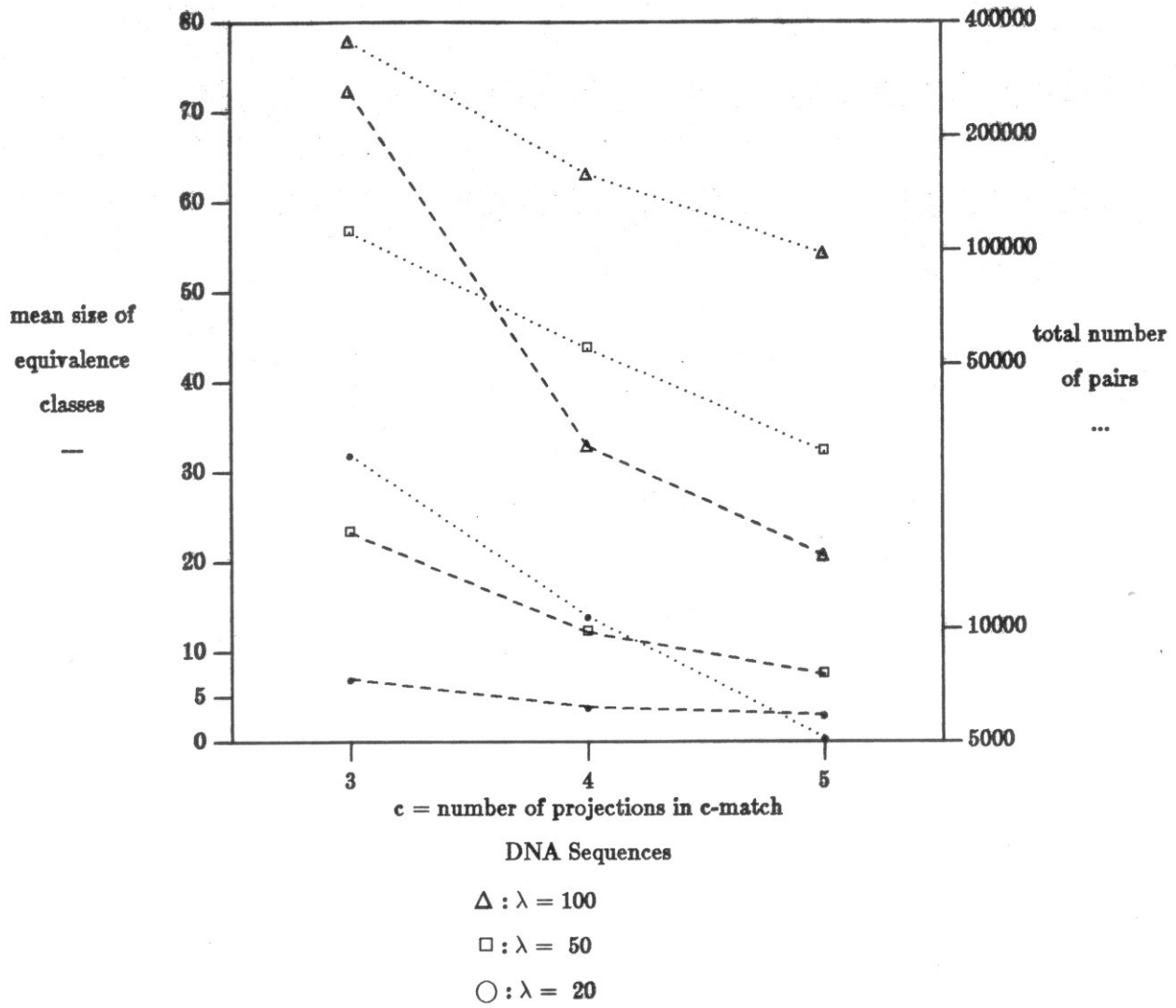

mean size of
equivalence
classes
—
80
70
60
50
40
30
20
10
5
0
400000
200000
100000
50000
10000
5000
total number
of pairs
...
3
4
5
c = number of projections in c-match
DNA Sequences
Δ : λ = 100
□ : λ = 50
◯ : λ = 20

Fig. 6.3

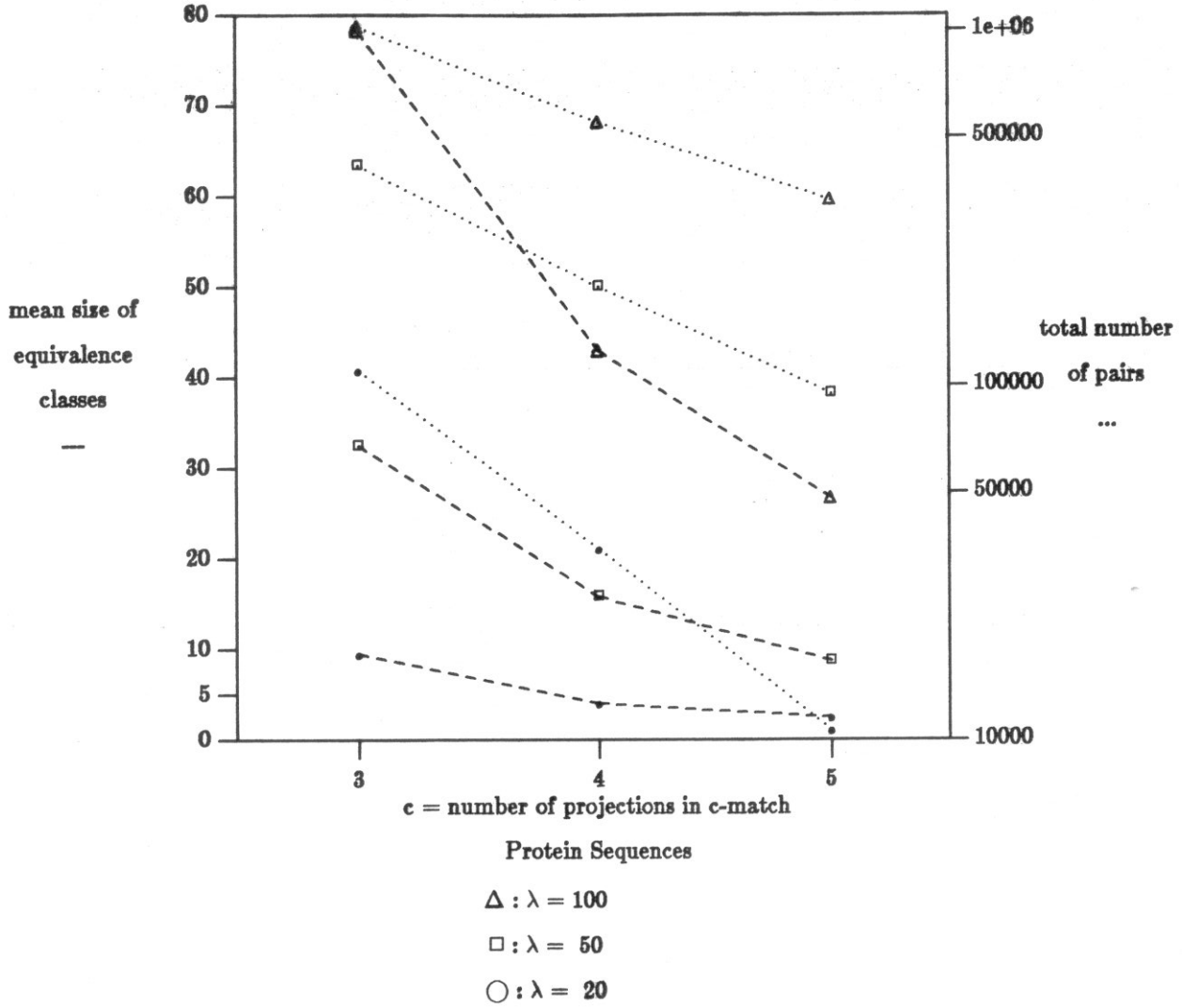
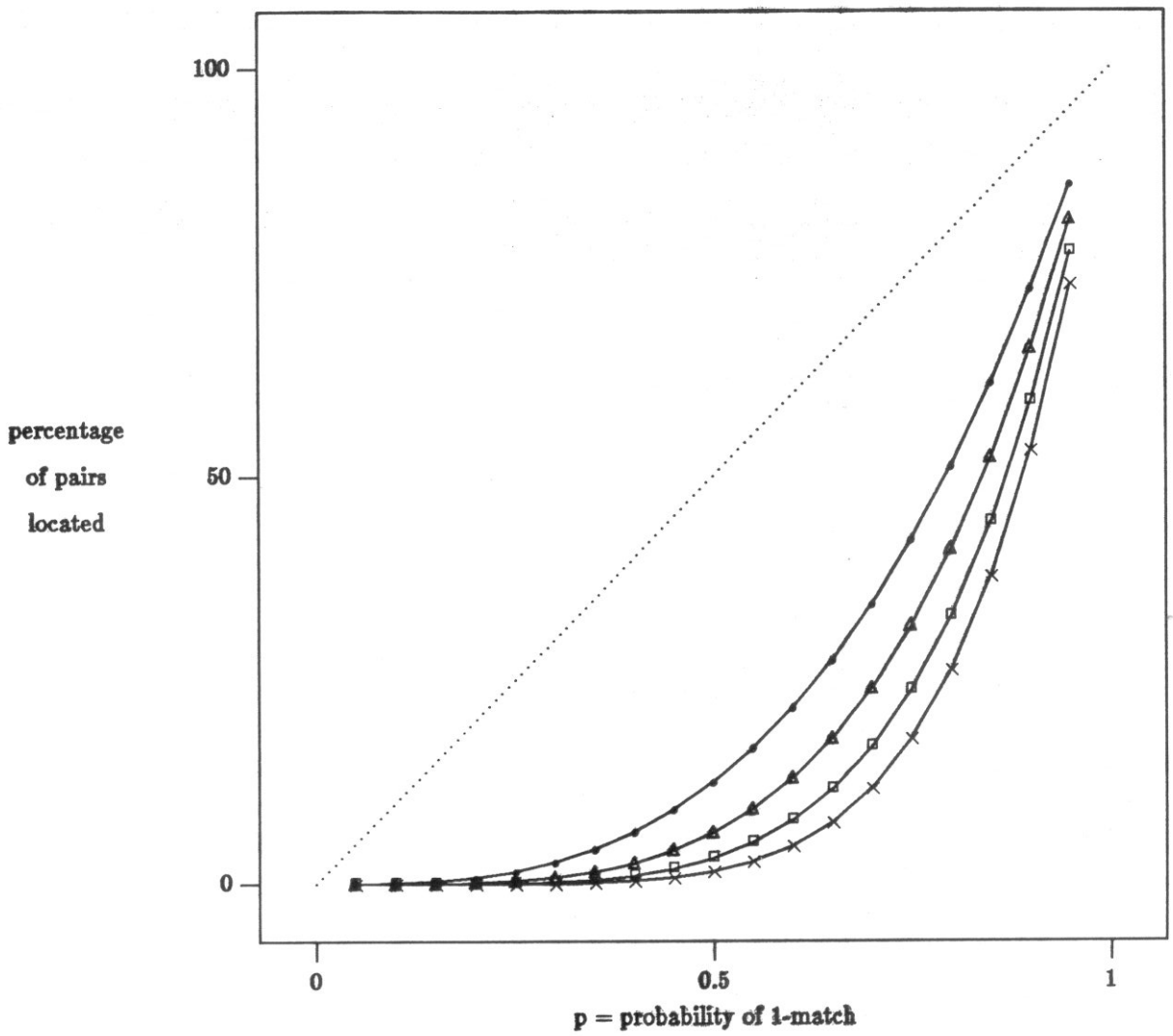

mean size of
equivalence
classes
—
80
70
60
50
40
30
20
10
5
0
1e+06
500000
100000
50000
10000
total number
of pairs
...
3
4
5
c = number of projections in c-match
Protein Sequences
Δ : λ = 100
□ : λ = 50
○ : λ = 20

Fig. 6.4



N = 1

○ : c = 3

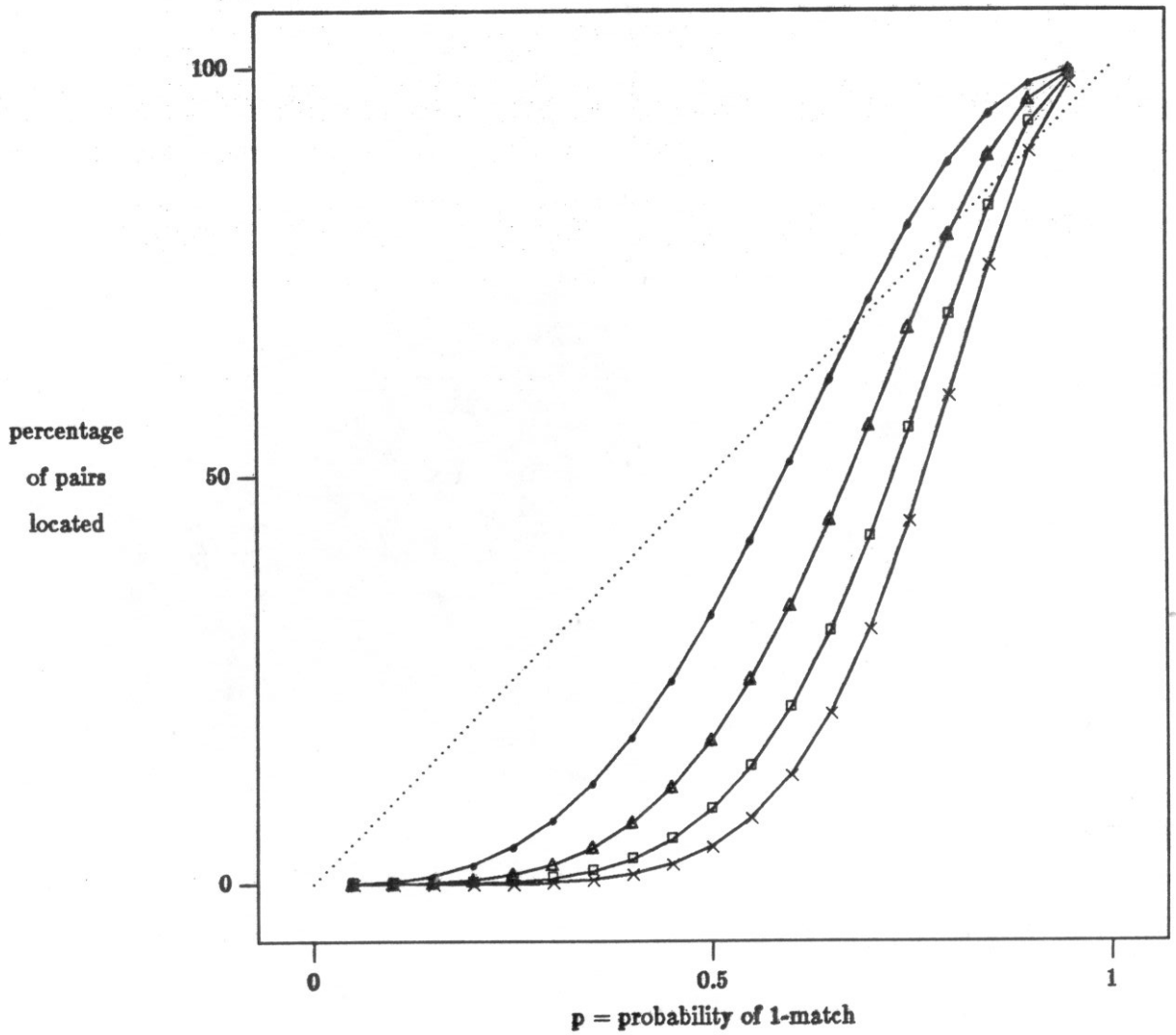
Δ : c = 4

□ : c = 5

× : c = 6

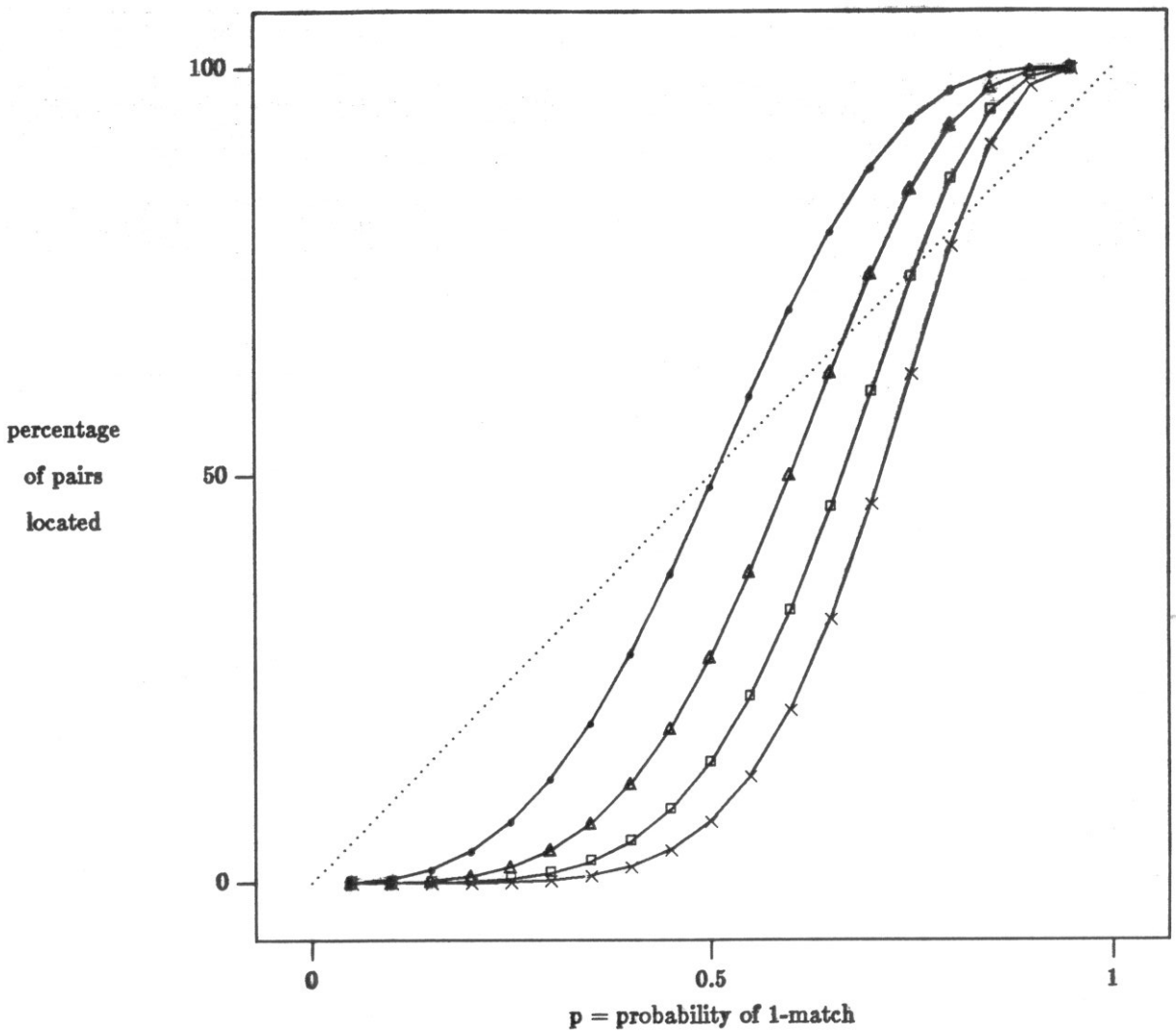
(dotted line represents c = 1, N =1)

Fig. 6.5



(dotted line represents c = 1, N =1)

Fig. 6.6



N = 5

○ : c = 3

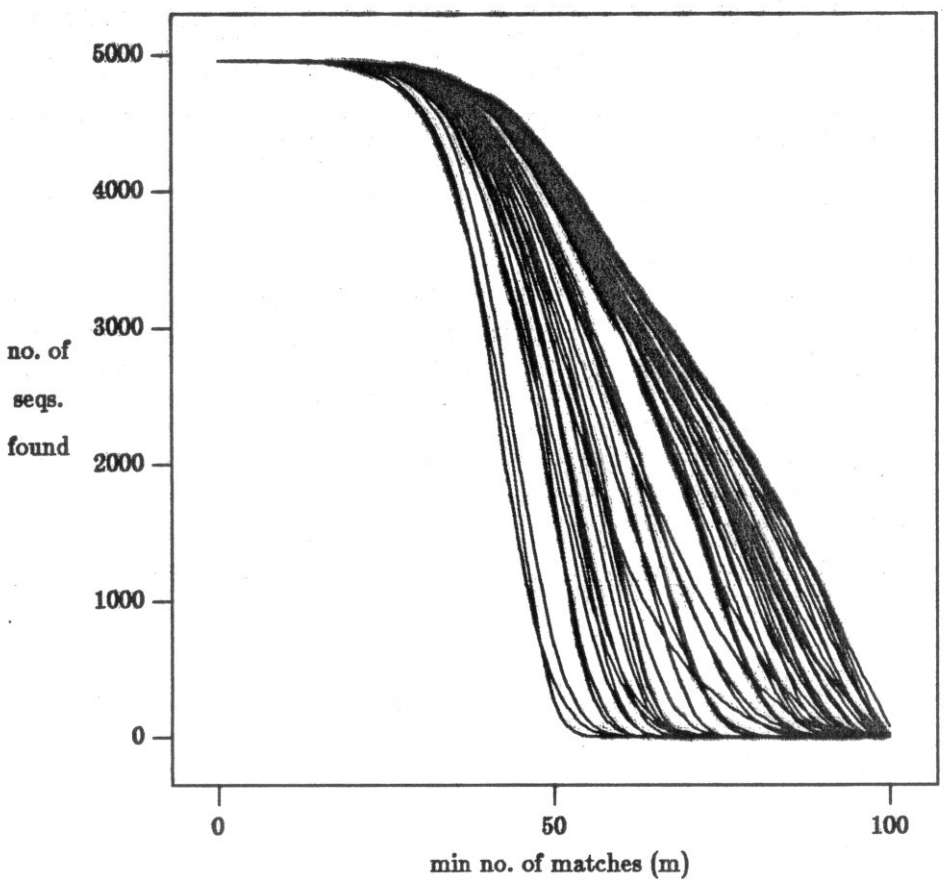
△ : c = 4

□ : c = 5

× : c = 6

(dotted line represents c = 1, N =1)

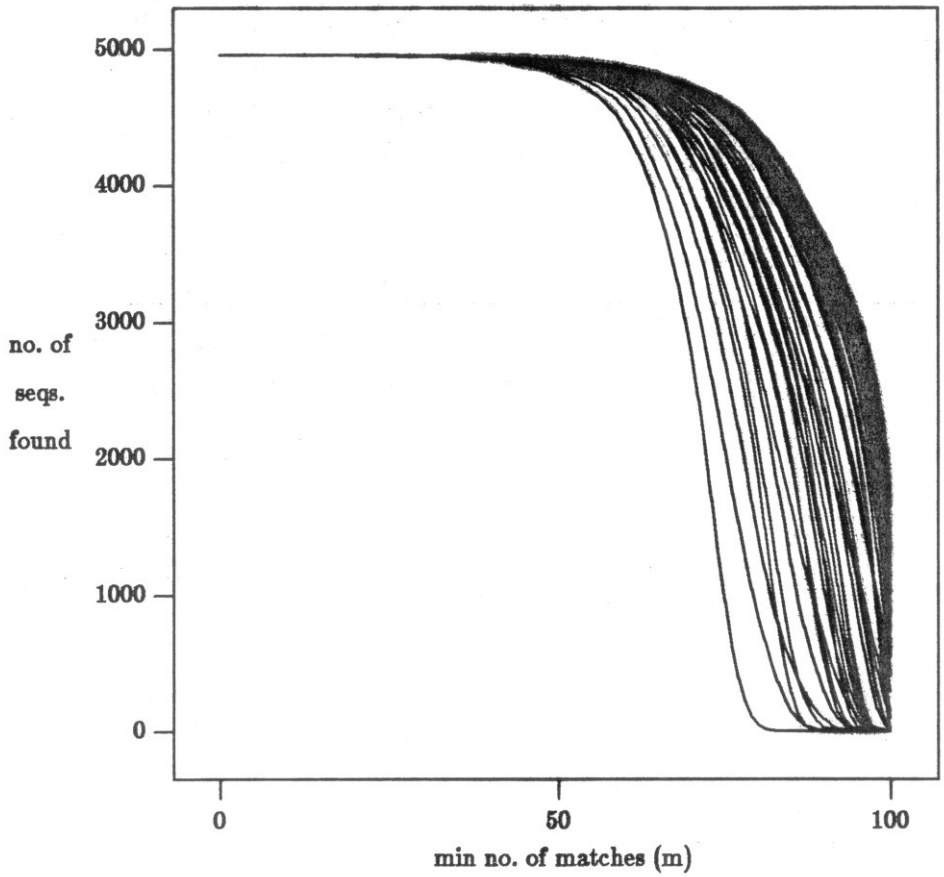
Fig. 6.7



window size w = 50

Fig 6.8

60 query sequences against DNA databank



window size w = 100

Fig 6.9

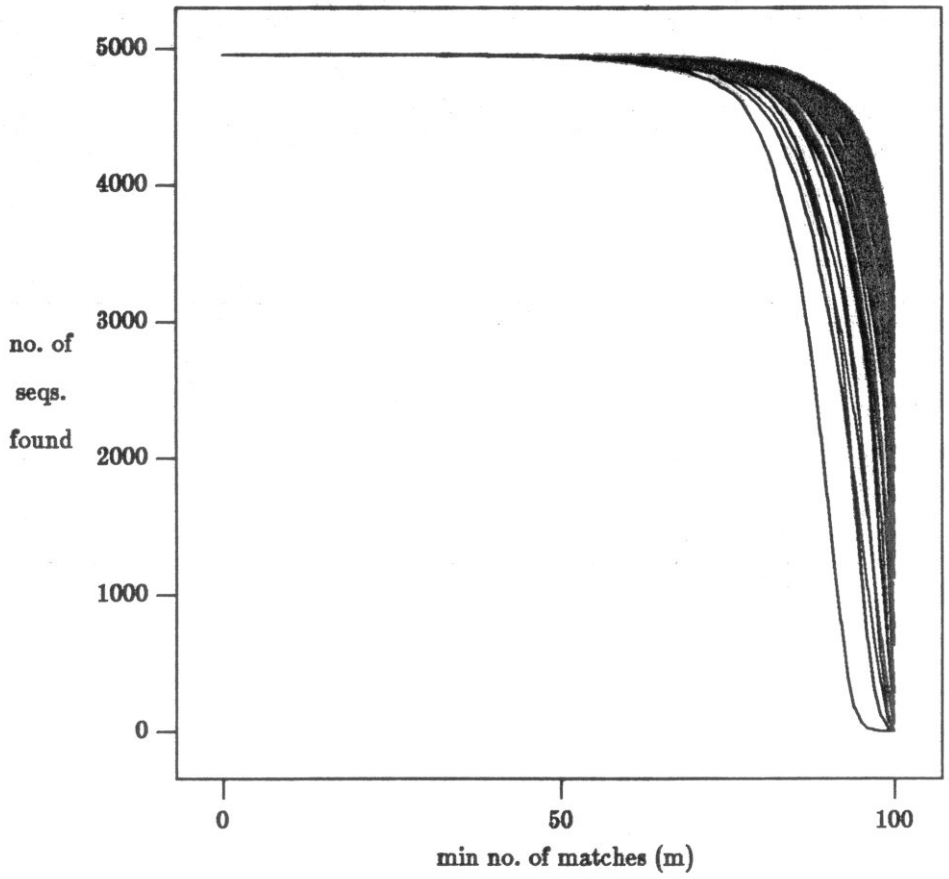
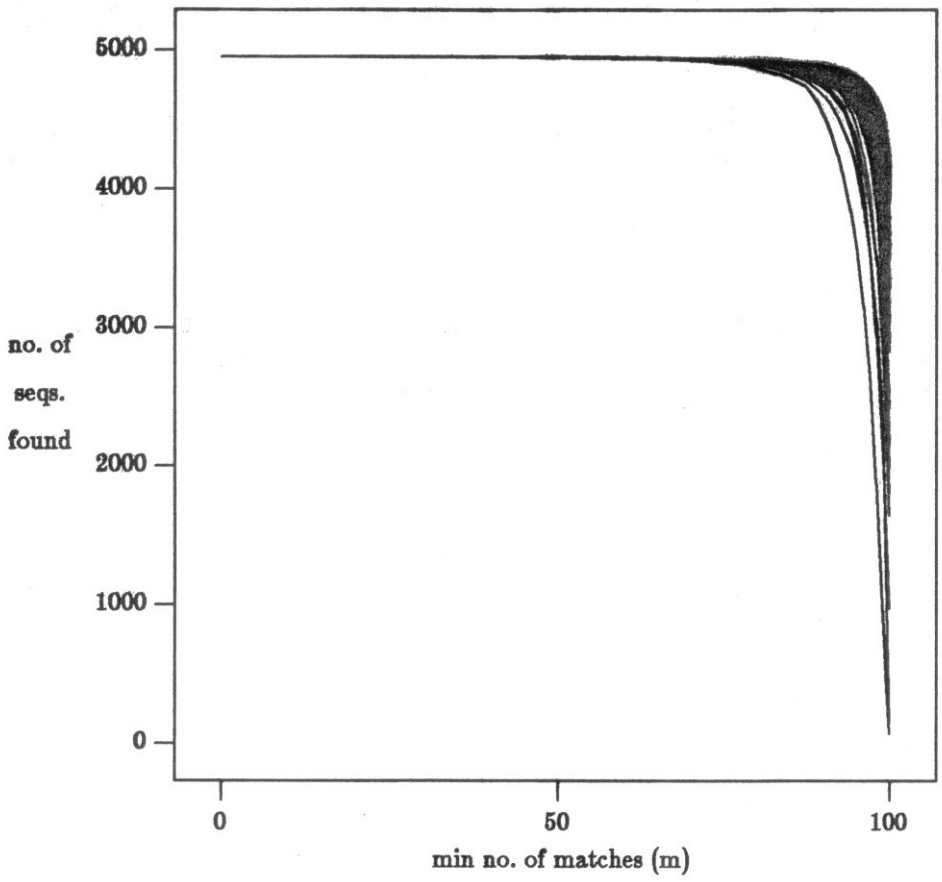

50 query sequences against DNA databank
5000
4000
3000
2000
1000
0
no. of
seqs.
found
0
50
100
min no. of matches (m)
window size w = 150

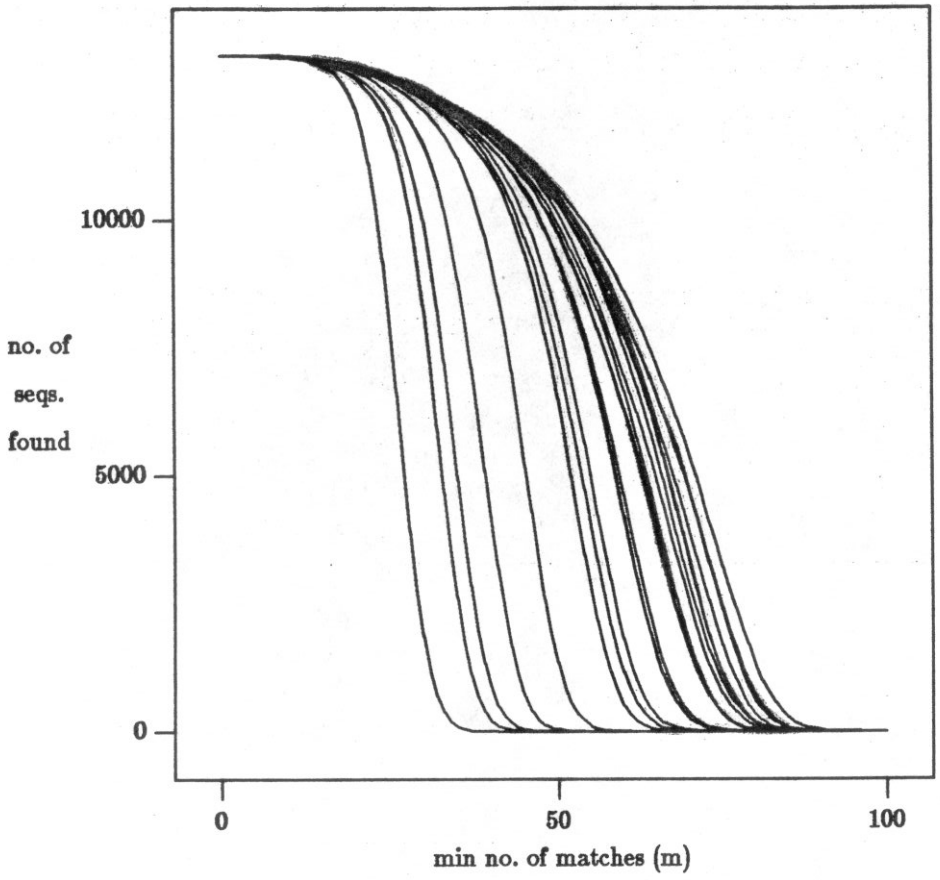
Fig 6.10



window size w = 200

Fig 6.11

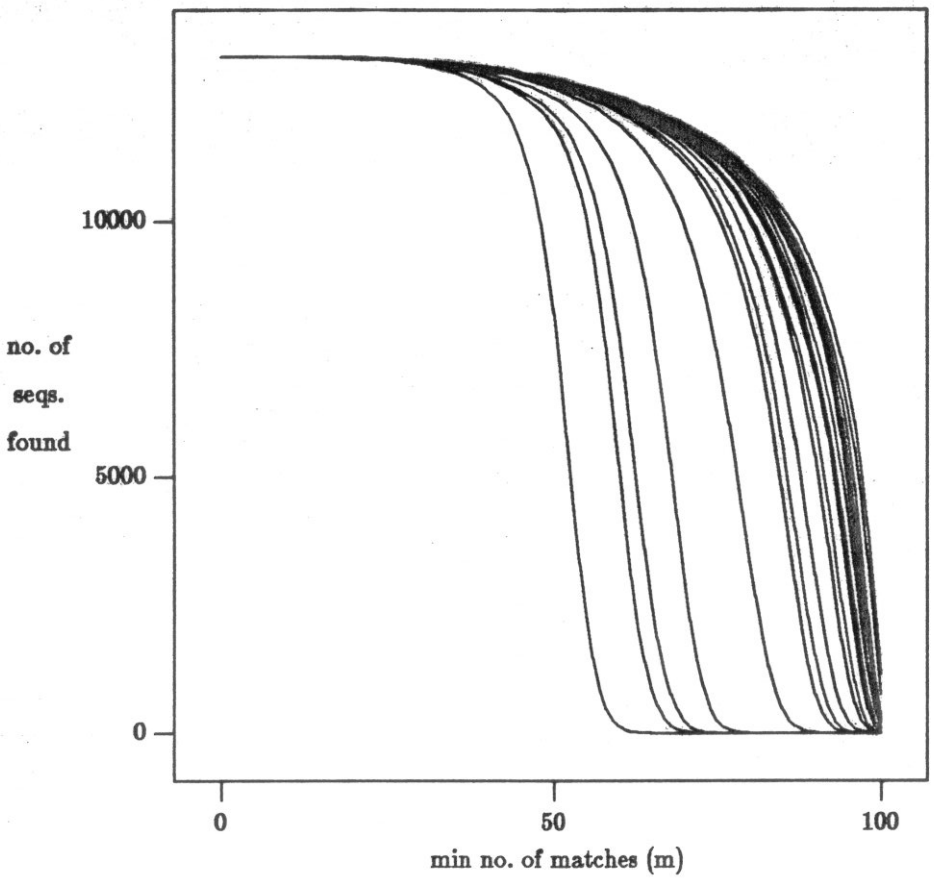
25 query sequences against Protein databank



window size w = 50

Fig 6.12

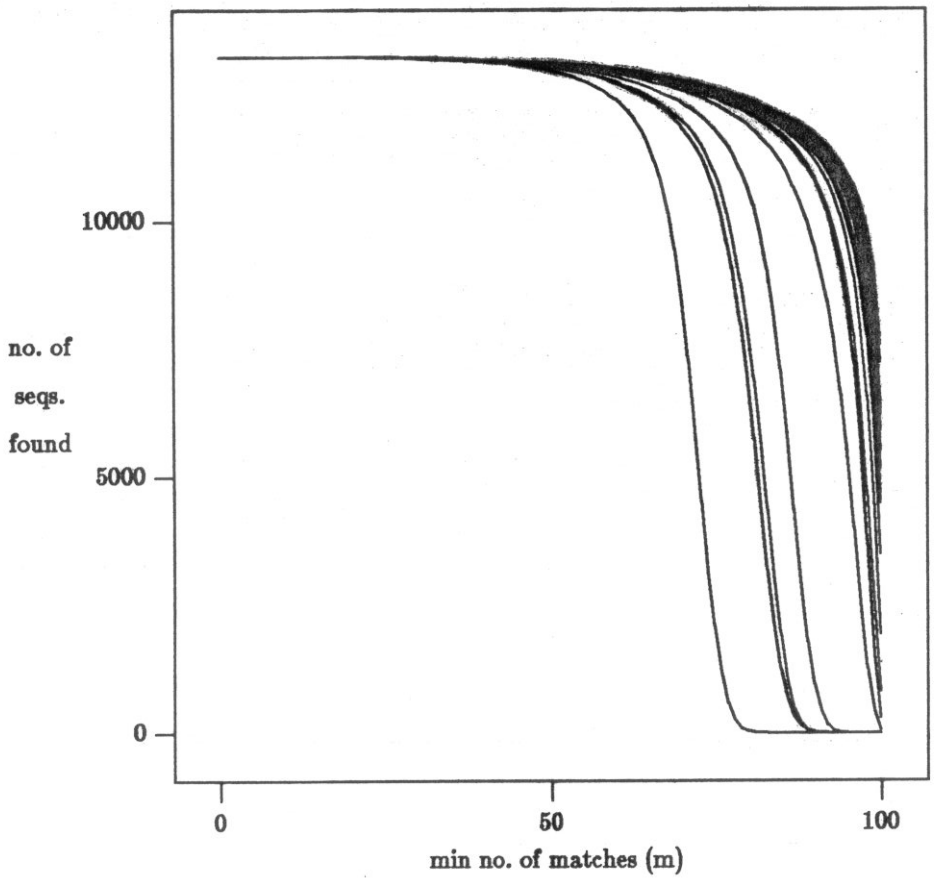
25 query sequences against Protein databank



window size w = 100

Fig 6.13

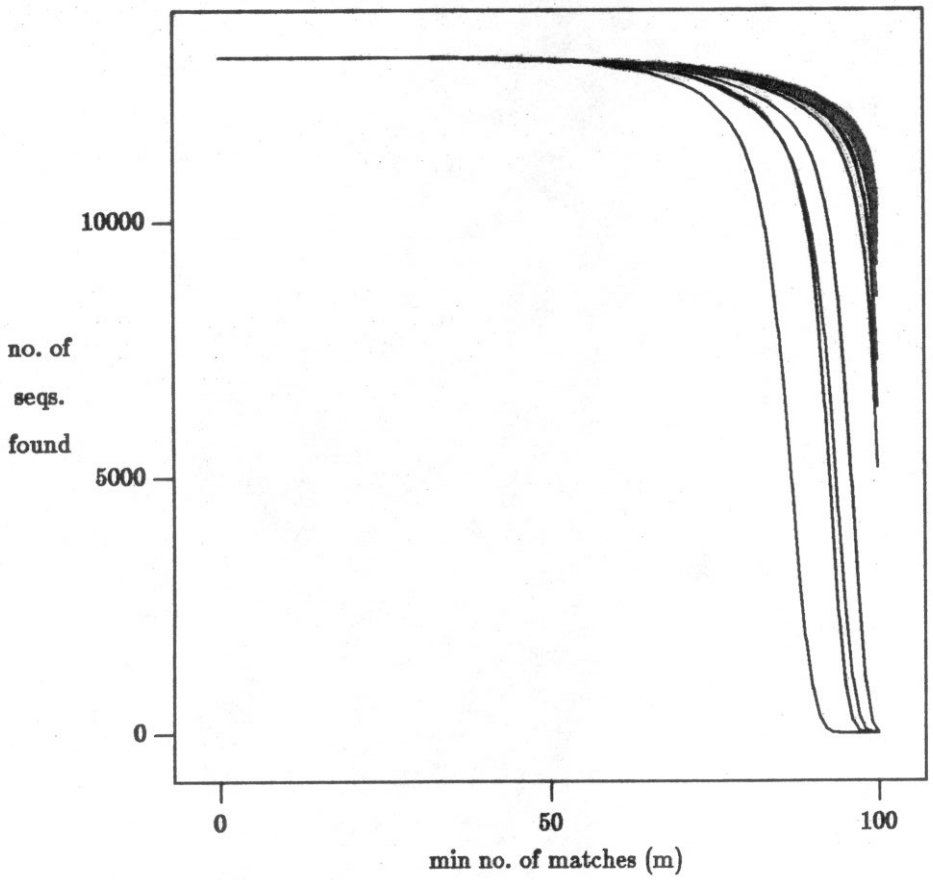
25 query sequences against Protein databank



window size w = 150

Fig 6.14

25 query sequences against Protein databank



window size w = 200

Fig 6.15

VII. Conclusions

This thesis has presented a new representation for genetic sequences as geometric points or vectors and shown that the concepts of closeness and distance as applied to ordinary points are also meaningful when applied to these "sequence points". This representation offers significant reduction in the time taken to compute the distance between any pair of sequences.

The conversion of the basic data objects from sequences to points also offers additional advantages in terms of allowing geometric techniques to be applied to the study of genetic sequences, including or perhaps especially, techniques that were not specifically designed to handle sequence objects. This is of importance since it opens up the study of computational problems in molecular biology to a wider variety of computing techniques, a result that cannot but be advantageous to the researcher in biology confronted by a problem that is computational in nature. This effect is borne out by the techniques described in the previous chapter, which are a direct consequence of the ability to think of genetic material as geometric objects rather than string objects.

Popular sequence comparison methods currently in use, most notably the FASTA and FASTP family of programs [14][19][28], rely on heuristic speedup of the dynamic program approach [27], by using local exact matches to narrow the region within which the innermost optimization step of the dynamic program must be performed. However they retain some of the disadvantages associated with the dynamic program in that the entire

array or a large fraction must be scanned in the worst case to determine the significant regions. Our approach obviates this problem since the only step which is dependent on sequence length is the initial generation of the vectors themselves, which takes time linear in the sequence lengths. Subsequent stages are independent of sequence length. Further the existing methods are applicable to only one pair of sequences at a time. Thus comparing a database of N sequences against itself would require on the order of N^2 steps, each of which is dependent on the sequence lengths. Our representation, on the other hand, leads to a much faster method of comparing a whole database against itself requiring only on the order of $N\log N$ pairwise comparisons each of which is a constant time comparison of two vectors.

This work also raises some questions and opens up several avenues to be explored. In particular, one important aspect which could be fruitfully exploited is the inherent parallelism in most vector computations. Thus most of the techniques described here can be very easily adapted to run on vector machines, which are becoming more and more widely available to the biology community, relatively easily.

We have also seen that while closely related sequences will have close sequence points, it is also conceivable that two unrelated sequences may by some chance have identical tuple compositions, an event which would lead to spurious indications of closeness between sequences which may then have to be weeded out by other means. It would be of great interest to see how often this sort of event does in fact occur. Another factor that deserves further consideration is the idea of weighting different tuples differently or, in other words, changing the scale on some of the co-

ordinate axes. This could perhaps be used to highlight the prescence of certain important or rare substrings or downplay the prescence of more common ones.

In conclusion, perhaps the most important understanding resulting from this thesis is the need for computer scientists to gain a greater understanding of the biological aspects of the problems involved as well as the computational aspects - a feeling that is becoming widespread in the among computer scientists working in this area - for only so can the field of computational biology continue to effectively the needs of biological research.

Appendix I : Implementation of a Sequence Indexing System

The techniques described in chapter 6 have been used to implement a sequence indexing system, written in C and currently running under the MACH operating system version 2.5 on a VAX 11/785 with 98MB available memory.

This system processes a database of genetic sequences, either DNA or Protein. After some initial processing of the sequences the system can handle queries in the form of being given a query sequence, all sequences from the database that are close to this query sequence within certain parameters. It has been designed to run in three phases, the first two constituting the preprocessing stages and the last being the actual query processor. A phase by phase description follows.

Phase 1 - Conversion to Sequence vectors

This phase handles the conversion of the database sequences to their vector representation. The input to this phase consists of a single file containing only the sequences themselves with no other information. It will then generate the vector representation as an array of floating point numbers (of size $4^4 = 256$ corresponding to a tuple size of 4 for DNA sequences and of size $20^2 = 400$ corresponding to a tuple size of 20 for Protein sequences) and store them in the same order in which they were generated. Hereafter any reference to the ordinal number or just the number of a sequence from the original database will refer only to the

order in which the sequences were input to this phase.

Phase 2 - Generation of projections

In this phase the system will generate a series of random linear combinations, the number of which (k) is settable by the user. The coefficients of these are generated as uniform random variables in a range, of size definable by the user, symmetric around zero. It will then apply each linear combination to the entire set of sequence vectors generated in the previous phase. Each projection of a sequence vector is represented as a structure with two fields, one for the projected value and the other for the ordinal number of the sequence. Prior to this phase the size of the entire database (number of sequences) and the number of projections desired must be set in the global parameters file. each set of projections is then sorted according to the projected value and store in this sorted order. The coefficients of the each linear combination are also saved.

phase 3 - Searching

This is the repeatedly executable search program itself. When executed it will first initialize the system by reading in the file of sorted projections as well as the file of linear combinations. It then executes a command interpreter which allows the user to give a query sequence to the system, control output of information regarding the sequences found or change the parameters of the search.

Input to the system is achieved by supplying the name of a file containing the sequence(s) to be search for. Input files can be changed.

Processing of one sequence in the input file terminates and processing the next one begins only under explicit command.

The sequence information output consists of short one or two line descriptions of the database sequences that were found to match the query sequence within the existing parameters. These descriptions are taken from a file, supplied by the user, which contains this information in the same order as the sequences themselves. Since listing this information can be time consuming it is only done under explicit command and thus allows the user to adjust the parameters to obtain a reasonable number of matching sequences before requesting a listing.

The searching parameters are the size of the "window" w and the minimum number of times m that a database sequence must fall in the search window before it can be considered. Initially these are set to be 0 and k respectively. They can be adjusted at any time.

When a new query sequence is encountered it is first converted to a vector. Then each of the linear combinations are applied to it in turn generating its projected value in each of the k projections. Then by performing a binary search on each of the projections we find that entry which is closest to but not greater than the projected value of the new sequence. From this central position we then scan linearly in both directions till we find all the sequences that are within the window size w of the projected value of the query sequence. Also a match count table is maintained of the number of times each sequence matches the query sequence and since this number must be between 0 and k we also maintain, for each number from 0 to k the number of sequences that match exactly that many times, in a frequency count table. Raising or lowering the the w value is easily

accomplished by saving the end points of the search window in each projection, thus the expansion or contraction of the window begins from known points and involves no searching. When the window is being expanded (or contracted), as new sequences fall in (or out) of the windows the match count and frequency count tables are updated accordingly. Finding the number of sequences from the database that match the query sequence under the given parameters is then accomplished by summing the values in the frequency count table from m to k.

Any time a new query sequence is encountered or one of the two parameters are changed the system will compute and display the total number of sequences from the database which have been found. The user can then decide whether this is a manageable size, too little or too much and adjust the parameters accordingly, before requiring the sequence information itself to be output/displayed.

When actual sequence information is required the system will scan the entire match count table to determine all those sequences that have matched the query sequence at least m times. The sequence information file entries corresponding these sequence numbers is then written. This is the only step that requires scanning the entire database, but as this is expected to occur less frequently than requests for changing parameters, it was felt that it would be more worthwhile to design the table structures to speed up the latter at the expense of time consumed in output.

References

[1] R. S. Boyer, J. S. Moore, "A fast string searching algorithm" *Comm ACM*, Vol 20, No 10, Oct 1977. pp 262-272

[2] Clift, B. et al, "Sequence landscapes" *The applications of computers to research on nucleic acids III* D. Soll & R. J. Roberts eds., IRL Press 1986. pp 141- 158

[3] V. Chvatal, D. Sankoff, "Longest common subsequences of two random sequences" *J. Appl. Prob.* Vol 12, 1975. pp 306-315

[4] D. Dobkin, R. J. Lipton, "Multidimensional Searching Problems" *SIAM J. Comput* Vol 5, No 2, June 1976. pp 181-186

[5] J. A. Hartigan, "Clustering algorithms" John Wiley and Sons, 1975

[6] X. Huang, "A lower bound for the edit distance problem under an arbitrary cost function" *Information Processing Letters*, vol 27, no 6, May 1988. pp 319-322

[7] D. Knuth, "The art of Computer Programming, vol. 3, Sorting and Searching" Addison-Wesley, 1973.

[8] D. Knuth, J. Morris, V. Pratt, "Fast Pattern Matching in Strings" *SIAM J. Comput.*, vol 6, no 2, June 1977. pp 323-350

[9] G. M. Landau, U. Vishkin, "Introducing Efficient Parallelism into Approximate String Matching and a new Serial Algorithm" *Proc. 18th ACM Symposium on Theory of Computing.*, 1986. pp 220-230

[10] G. M. Landau, U. Vishkin, R. Nussinov, "An efficient string matching algorithms with k differences for nucleotide and amino acid sequences" *The applications of computers to research on nucleic acids III* D. Soll & R. J. Roberts eds., IRL Press 1986. pp 31-46

[11] A. Lempel, J. Ziv, "On the complexity of finite sequences" *IEEE Trans. on Information Theory*, vol IT-22, Jan 1976. pp 75-81

[12] M. E. Lesk "Some Applications of Inverted Indices on the UNIX System" Bell Laboratories, Murray Hill, NJ.

[13] B. Lewin, "Genes III" John Wiley & Sons, 1987.

[14] D. J. Lipman, W.R. Pearson, "Rapid and Sensitive Protein Similarity Searches" *Science*, vol 227 (1985) pp. 1435-1441

[15] D. Lopresti, "Discounts for Dynamic Programming with Applications in VLSI Processor Arrays" Ph.D. Thesis, Princeton University, 1985.

[16] J. V. Maziel Jr., "Supercomputing in Biomedical Research" *Cray Channels*, Fall 1988. pp 2-5

[17] D. R. Morrison, "PATRICIA - Practical algorithm to retrieve information coded in alphanumeric" *JACM*, vol 15, no 4, 1968. pp 514-534

[18] S. B. Needleman, C. D. Wuncsh, "A general Method Applicable to the search for Similaities in the Amino Acid Sequence of Two Proteins" *Journal of Molecular Biology* vol 48, 1970. pp 443-453

[19] W.R. Pearson, D. J. Lipman, "Improved Tools for Biological Sequence Comparison" *Proc. Natl. Acad. Sci. USA,* vol 85 (1988) pp. 2444-2448

[20] C. Queen, M. N. Wegman, L. J. Korn, "Improvements to a program for DNA analysis: a procedure to find homologies among many

sequences" *Nucleic Acids Research*, vol 10, no 1, 1982. pp 449-456

[21] R. Schabak, "On the expected Sublinearity of the Boyer-Moore Algoprithm" *SIAM J. Comput.*, vol 17, no 4 Aug 1988. pp 648-658

[22] R. Sedgewick, "Algorithms" Addison-Wesley, 1984

[23] K. Thompson, "Regular Expression Search Algorithm" *Comm. ACM*, vol 11, no 6, June 1968.

[24] D. C. Torney et al., "Computation of d-squared : a measure of Sequence Dissimilarity." *Computers and DNA, SFI studies in the Sciences of Complexity, vol VII*, G. Bell, T. Marr Eds. Addison Wesley 1990. pp 109-125

[25] R. A. Wagner, M. J. Fischer, "The string to String Correction Problem" *J. ACM*, vol 21, no 1, Jan 1974. pp 168-173

[26] R. Wilber, "The concave least-weight sub-sequence problem revisited" *Jornal of Algorithms*, vol 9, no 3, Sept. 1988. pp 418-425

[27] W. J. Wilbur, D. J. Lipman, "Rapid Similaity Searches of Nucleic Acids and Protein Data Banks" *Proc. of the National Academy of Sciences*, vol 80, Feb. 1983. pp 726-730

[28] W. J. Wilbur, D. J. Lipman, "The Context Dependent Comparison of Biological Sequences" *SIAM J. Appl. Math.* vol 44, no. 3 (1984) pp 557-567