Fast Search Methods for Biological Sequence Databases

Sumit Ganguly
Jerry Leichter
Michiel Noordewier

ganguly,leichter,noordewi@cs.rutgers.edu

Department of Computer Science
Hill Center, Busch Campus
Rutgers University
New Brunswick, NJ 08903
October 26, 1993

LCSR-TR-217

Abstract

Biology researchers have a pressing need for data management technologies which will make the storage and retrieval of **DNA and protein sequence data** accurate and efficient. The volume of data generated by DNA sequencing is already massive and will continue to grow rapidly. Even if the current sequence databases are adequate today, they most assuredly will become inadequate in the future when far more sequence data has been determined. The direction of future research in sequence databases needs to be in the organization of information. This is so that the volume of data needing to be searched does not grow linearly with the volume of sequence data being discovered.

We propose to develop an **index structure** and retrieval system called PROXIMAL for **biological sequence databases** which promises to be efficient and general. This organization of the databases will complement other current efforts at sequence comparison and analysis, by providing an infrastructure in which other methods can be used to efficiently locate desired sequences. Our method relies on the use of reference strings to partition the database of sequences. It is *efficient* since the use of multiple reference strings for any given distance measure greatly reduces the number of sequences that must be examined, allowing us to quickly locate sequences based on a precomputed metric. It is *general* since multiple distance measures can be used. These include at least differing gap and mismatch weights for the basic edit distance calculation, or entirely different models of mutation. The only requirement is that there is a metric structure — mainly, that the calculations satisfy the triangle inequality. This is a weak requirement that is satisfied by many interesting measures, including those currently in wide use for sequence comparison.

Sequence Databases in Molecular Biology

Introduction

The efficient and accurate retrieval of sequence information from molecular biological databases will play a role in almost all future research in molecular genetics. This follows from two landmark discoveries:

- In 1953, Watson and Crick discovered the molecular structure of DNA, which explained the fine structure of genetic material [26]. Knowledge of this structure allowed representation of genes as strings of characters corresponding to the arrangement of individual nucleotide bases.

- In 1977, two separate laboratories developed methods for rapidly determining the sequence of DNA molecules [14, 18]. Modern study of heritable traits is inevitably associated with determination of the DNA sequence by variants of these methods. Sequencing technology is rapidly improving, and by 1994 it is estimated that 1.6 billion base-pairs will be sequenced each year [11].

The existence of libraries of DNA sequences allows researchers to attempt to find genes that may be similar to a newly determined sequence. An example of such a relationship was discovered in 1983, when Doolittle and colleagues reported that a newly discovered oncogene was similar to a normal gene for development and growth [3].

The idea of using databases to store biological information is not entirely new; they have been used for over 10 years [11]. The development of database concepts has been under way in the computer science community for much longer than a decade. However, biological problems have not influenced database research until recently. Today, sequence data are collected by several organizations which compile the data and distribute it publicly. GenBank, EMBL and SwissProt are examples of current, widely distributed databases containing DNA and protein sequence information. These databases are used by biology researchers in at least three different ways:

- Given a newly discovered sequence, search the database to determine if others have previously studied the same sequence.

- Search for known sequences *similar to* the new sequence. Similar sequences are of interest because they often give clues about the biological function of the sequence.

- Compare sequences in the database against the entire database, looking for similarities in groups of sequences.

The sequence databases are currently distributed in a flat file format. A sequence header describes the sequence; the nucleotides follow; and a special terminator symbol marks the end of the data for the sequence. The header for the next sequence follows immediately after the terminator. While some computer scientists may pause before calling a flat file a database, a term which often conjures up images of fast indexed access, such files are often referred to as databases in the biological literature. The original intent of this simple format was to distribute the data in a platform-independent fashion, with the assumption that target systems would structure it accordingly [7]. The data collections are therefore more aptly termed data libraries, and are not well-suited for fast retrieval. For example, given a newly discovered sequence, the biologist must scan the entire file to determine whether the sequence string has been found previously. At times this search has been performed using simple UNIX utilities such as `grep`. When one considers that in 1991 GenBank contained more than 44,000 separate sequence entries comprising a total of more than 66 million characters [11], this method appears woefully inadequate. The problems with this simple approach are compounded when the biologist wants to extract not only exact matches, but also similar sequences.

There are several measures of similarity in current use for sequence comparison. One of the simplest is the Hamming distance, which is a count of the number of positions where two sequences differ [8]. The most rigorously investigated measure of similarity between two sequences, however, describes similarity in terms of a *string edit* problem: Find the minimum cost of editing one string into another using a specified set of operations, each of which is assigned an individual cost. The editing operations are chosen to correspond to events that occur naturally in gene sequences. For example, the operations might include substitutions, deletions, insertions, duplications, transpositions, and inversions. The minimum edit cost with respect to such a set of operations provides a measure of the evolutionary and functional

relationship of pairs of sequences. In actuality, one must be careful in relating string-edit costs to real, biological evolutionary distance. The various string-edit events have different probabilities in the world of the laboratory, and many constraints (such as lethality) are unrelated to string-edit cost. Nonetheless, string-edit costs and various approximations are currently the most widely used measures for relating biological sequences.

Calculating the string edit cost can quite be difficult. The problem has received significant attention, and has inspired the development of several algorithms [16, 19, 20, 25]. The existence of these algorithms would seem to satisfy the need for a method to compare and retrieve gene sequences from sequence databases. Unfortunately, the algorithms are based on dynamic programming techniques which require a time for execution proportional to the product of the length of the query sequence and the total length of the sequences in the data libraries. Since the current size of the libraries precludes exhaustive pairwise comparison with a normal query sequence, approximate methods have been developed [24]. Two of the most widely used are FASTP [13], which utilizes a search based on identities but not gaps; and BLAST [1], which employs a more realistic mutation model, but whose implementation requires a direct tradeoff between accuracy and speed. The use of such methods is less desirable than string-edit calculations, which corresponds to known biological mechanisms for the evolution of nucleic acid sequences.

Current Use of Database Technology

There are currently no database techniques which are directly applicable to the problem. Historical development of databases for business and financial applications resulted in the "classic" hierarchical, relational and network models. The data structures used to implement these models, particularly B-trees and related structures, rely on the existence of a one-dimensional ordering of the data, and support queries for exact or range matches. For example, such a database can be used to very quickly retrieve motor vehicle records corresponding to a particular license plate number, or even records with license plate numbers falling within a given numerical range. However, they are of little or no help in finding records with license plate numbers that might have been misread to produce the invalid number reported by a witness to a crime. Such "plausible misreadings" are analogous to the bio-

logical relationship "close in function or evolution"; they are based on some notion of "closeness" that is expensive or difficult to characterize, and does not produce the required one-dimensional ordering in any case.

Database research in recent years has focused on such areas as:

- Support for complex data objects (Object-Oriented Data Bases);
- Use of logic and recursion, and the support of more complex deduction (Deductive Data Bases);
- Storage of geometric data accessible using geometric constraints (Graphic and Geographic Information Systems).

None of the techniques developed in these areas directly solves the problem of retrieving sequences which are similar to a genetic query sequence, although some of the techniques developed for graphic and geographic information systems may be applicable.

Several research efforts have attempted to improve the performance of large-scale biological sequence database searches. These efforts have been roughly divided into two categories:

- applying more computing power, and
- structuring the database to reduce the required number of sequence comparisons.

Below, we briefly review significant recent efforts in each of these directions.

More Computing Power

Research in this area is most commonly focused on developing methods for running conventional sequence comparison algorithms in parallel. For example, Miller describes an implementation of the FASTA algorithm using the machine-independent parallel programming language, Linda [15]. In this parallelized version of FASTA, there are many worker processors, each of which is given a copy of the query sequence and which then compares this with assigned sequences from the database. A master processor oversees the assignment of sequences to processors. This implementation has been run on

several parallel computers, including the Encore Multimax and Sequent Symmetry. The results of this work are reported to indicate that the performance of entire database searches can be significantly improved using a parallel implementation. However, this improvement is not linear with the number of processors due to a bottleneck: the master process is unable to perform I/O fast enough to satisfy all worker processors' requests. Several possible approaches to resolve the I/O bottleneck are discussed, such as preloading the database in memory. A similar example of applying more computing power is in work reported by Vogt [23]. Here, a personal computer served as a master and I/O server and was used to drive multiple transputers[1] which did the sequence comparison. Although no I/O bottleneck was reported in this configuration, it should be readily apparent that the division of work is fundamentally the same as that reported by Miller, and the same bottleneck would be expected here also with a larger number of processors.

The BLAST algorithm has also been implemented in a massively parallel fashion in two separate efforts; BLAZE (due to the IntelliGenetics Corporation) and BLITZ (developed at the European Molecular Biology Laboratory, EMBL). BLITZ is now publicly available through the EMBL mail server, and is a parallel implementation running on EMBL's MasPar developed by Shane Sturrock at Edinburgh.

It has been reported that the rate of discovery of sequence data is increasing 10-fold every 5 years (from 1.5 million nucleotides per year in 1989 to an expected 1.6 billion nucleotides per year in 1999) [11]. With this rapid increase in data volume, it is questionable whether advances in computer hardware and parallelization techniques will be able to keep pace. The fundamental problem with the application of more computing power is that nothing is being done to limit the amount of data to be searched. As currently stated, these approaches blindly search the entire database. The use of a supercomputer simply turns the problem from CPU-bound to I/O-bound. Substantially increasing I/O rates, given current and forseeable technology, is a more difficult and considerably more expensive undertaking than increasing CPU speed or parallelism.

[1]Transputers are microprocessors designed to act as parallel function units in easily extensible computers of any size [23].

Structuring the Database

The basic idea of our approach is to structure sequence databases so that the data within the database should be stored in a manner which obviates the need to search the entire database. This is a traditional approach used in indexing schemes to limit the search required for the desired data. With the addition of data structures also comes additional administration. With the previously described parallel schemes, the data could be used in the same format that it was distributed. Structuring the database requires additional work, since newly distributed data must be converted before it can be used, but it has the potential to greatly increase search speeds.

An example of adding structure to a sequence database has been reported by Gonnet [6]. Here, patricia trees [5] were used to organize a protein sequence database. Patricia trees group data lexicographically, so similar sequences lie near each other in the tree. This allows branches of the tree to be pruned while searching; a search on a branch is abandoned after the distance between the query sequence and the sequence in the tree exceeds a chosen limit. By using this method of database structuring, Gonnet was able to align[2] a database of 8,344,353 nucleotides. Without structuring the data, this had previously been thought impossible; it had been estimated to require more than 106 years of computational time [22]. Gonnet, et al were able to perform the computation using only 405 days of computational time without losing any rigor. It should be noted from Gonnet's efforts that structuring the database consumes enormous amounts of time. Therefore, it is very expensive to update the database if changes cannot be made incrementally.

Although the problem of sequence alignment is slightly different than the problem of sequence database searching, they are similar enough for us to conclude that there is a significant benefit to be gained from structuring sequence databases. The problem then becomes one of determining the method of data structuring.

[2] Alignment refers to the process of taking a set of sequences and minimizing the total distance between the sequences. This is accomplished by allowing insertions, deletions or mismatches of corresponding symbols between sequences.

Overview of Our Proposed Database

We propose an index scheme for the rapid retrieval of genetic sequences based on edit distance. Such an index would obviate the need for exhaustive search, yet retain the ability to retrieve sequences within a specified edit distance of a query sequence.

Since the string edit method of sequence comparison has a formulation which corresponds to distance, we can impose the structure of a metric space on strings (sequences). We can then precompute the distance from one or more reference strings to all strings in the database. Subsequently, knowing the distance between a query string and the reference string, we can immediately locate a region of the database in which *all* strings within a specified distance of the query string *must* be found. Figure 1 illustrates this, using Euclidean distance in the plane to represent some appropriate string edit distance. The goal is to locate all strings within a specified distance (radius) of a query string — those strings within the circle in the figure. Within the database, the distances from a reference string to every other string have been pre-computed and stored. We begin by computing the single distance from the query to the reference string. Only those strings that fall within the illustrated "strip" of the database can possibly be within the circle. This strip consists of only those strings in the database whose distance from the reference string is within a range of the distance from the reference string to the query string. This strip is actually a portion of a ring whose width corresponds to the radius of the circle.

There is no reason to stop with a single reference string. Figure 2 shows what happens when two reference strings are used. The desired strings must be within the region defined by the intersection of the two "strips." The same idea can be applied with any number of reference strings.

If the strings are broadly distributed over the database with respect to the distance calculation, this method will greatly reduce the amount of subsequent comparison. The query string must ultimately be compared to strings within the "strip"; the more closely the intersecting strips approximate the target circle, the fewer comparisons will be needed.

This scheme is valid for any distance measure between sequences which defines a metric. There are four required properties:

1. Distances are non-negative numbers;

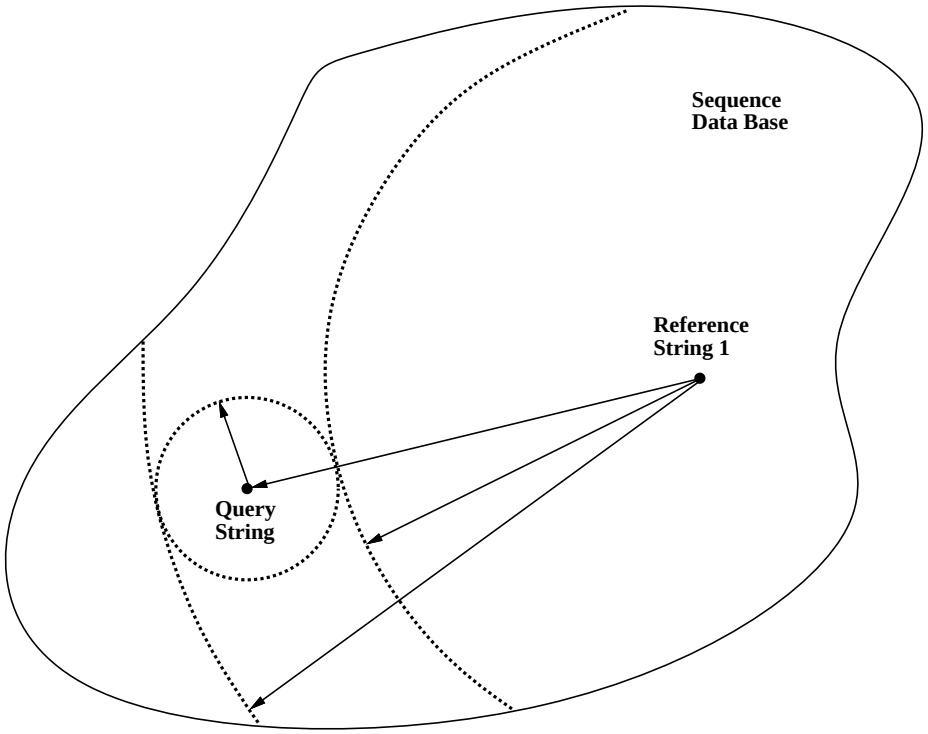

Figure 1: Reference String Radius

2. A sequence is at distance 0 only from itself.[3]

3. The distance between two sequences A and B is the same as the distance between B and A.

4. The distance satisfies the *triangle inequality*. The distance from A to B is no larger than the sum of the distances from A to C and from C to B.

For reasonable notions of distance, only the last property is commonly an issue. However, commonly used measures of sequence distance can be readily

[3]Actually, the technique also works if some distinct sequences are at distance 0 from each other.

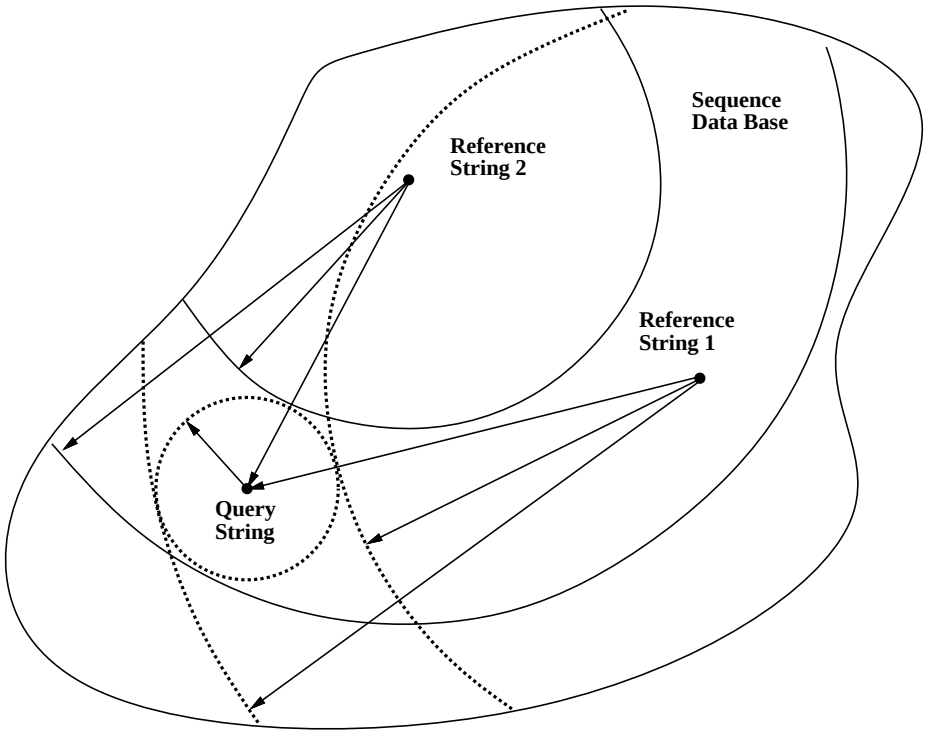

Figure 2: Multiple Reference String Radii

modified to be metrics. This will allow us to index a database using a number of distinct metrics, including:

- String edit distance with different choices of gap and mismatch costs;
- String edit distance with different operations, for example with a distinct cost for multi-character deletions;
- Motif and consensus searching (reduced alphabets, perhaps compiled into disjunctive queries);
- Other mutation models (e.g., MSP segments).

As an example, the availability of separate indices for differing edit costs for basic string edit is particularly attractive. Waterman [24] cites the example of a search which misses the desired sequence because of an unfortunate

selection of gap and replacement weights in the basic alignment algorithm. Multiple indices for different selections of these parameters would reduce the chance of missing such biologically related examples. The use of multiple indices is expected to produce the same benefits for other mutation models, such as MSP segments. Given the very high cost of searches without an appropriate database structure, it is unlikely that an experimenter would be able to explore more than a few possiblities.

In addition, indices may be constructed which facilitate the search for genes in uncharacterized sequence data now being produced by the various genome projects. In particular, indices could be constructed which correspond to the use of the "regions" method of comparing sequences [24]. This would allow conjunctive queries which quickly scanned a sequence database for the presence of multiple small regions. Such regions could be those which correspond to control regions required for gene expression.

Schematically, we envision the final index scheme as depicted in Figure 3. The triangular pointers represent differing indices into the underlying se-

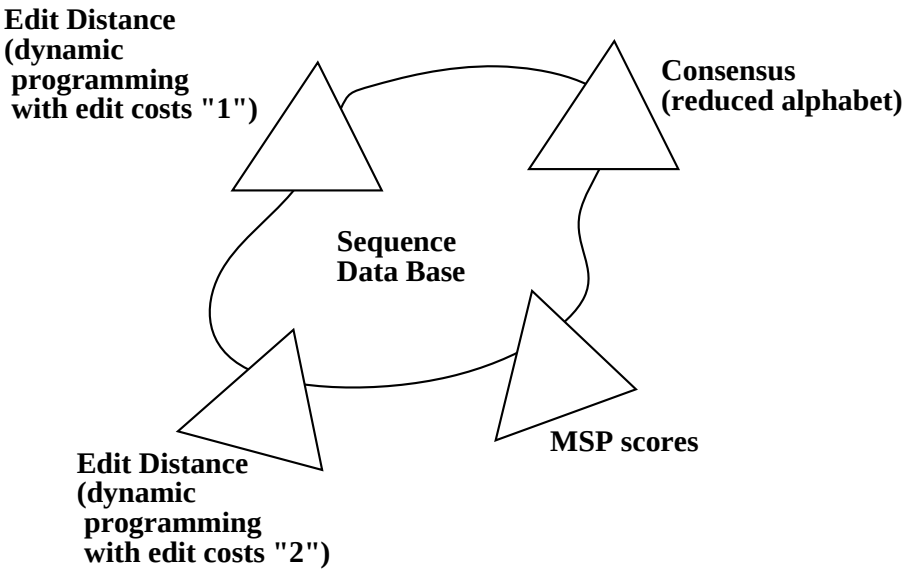

Figure 3: Conceptual Database Indices

quence database. Each of these indices is represented as a set of distances to each sequence from one or more reference strings, providing a variety of

perspectives on the library of sequences. The indices represent a one-time calculation, which can be carried out off-line, and updated incrementally as sequences are added to the database. Multiple queries are then carried out at a very low computational cost. Indices are expected to be small, so there will not much storage overhead. For example, if each pointer is four bytes, and a string is 1000 bytes, then storage for the index is $1/250^{th}$ of the room needed to store the underlying sequences.

Biology researchers have a valid and pressing need for data management technologies which will make the storage and retrieval of DNA sequence data reasonable. Even if the current sequence databases are adequate today, they most assuredly will become inadequate in the future when far more sequence data has been determined.

As has been shown here, the direction of future research in sequence databases needs to be in *structuring the data* so that the volume of data needing to be searched does not grow linearly with the volume of sequence data being discovered. Unfortunately, the application of existing algorithms and data structures to sequence databases may not always be appropriate. We propose to develop an index structure for the rapidly growing sequence databases, which promises to be efficient and general. It is efficient because the use of multiple reference strings for any given distance measure greatly reduces the number of sequences that must be examined. This allows us to quickly locate sequences based on a precomputed metric. It is general since multiple distance measures can be used. These include at least differing gap and mismatch weights for the basic edit distance calculation, or entirely different models of mutation. This organization of the databases will complement other current efforts at sequence comparison and analysis, by providing an infrastructure in which other methods can be used to efficiently locate desired sequences. The only requirement is that there is a metric structure — mainly, that the calculations satisfy the triangle inequality. This is a weak requirement that is satisfied by many interesting measures.

Theory and Preliminary Studies

We restate the fundamental problem: Given a database of biological sequences, a query sequence, and some pairwise numerical measure of similarity, find the sequences in the database that are "most similar" to the query. The brute force approach commonly used is simple to understand: Compute the similarity between the query and each sequence in the database, and then choose those sequences that gave the best results (greatest similarity). Typically, one asks for those sequences whose similarity to the query exceeds some cutoff value.

The problem with the brute force approach is two-fold:

- The number of sequences in existing databases is large and growing rapidly, and the number of similarity computations in this approach is linear in the size of the database;
- Many similarity measures of scientific interest require significant effort to compute, often effort proportional to the product of the lengths of the sequences being compared.

As a result, the cost of a brute force database search is typically proportional to the product of the length of the query and the total of the lengths of all sequences already in the database. This is a very large number, and one that is growing rapidly.

The alternative we propose is to pre-compute and store information that will allow us to inexpensively reject a large fraction of sequences in the database. The rejected sequences are guaranteed to be less similar to the query than the cutoff value. Only the remaining sequences need actually be compared to the query.

Central to our approach is the observation that the most widely used comparison methods for genetic sequences, while often viewed as defining a similarity measure, can equivalently be used to defined distance measures, or metrics. (Waterman [24] discusses the equivalence of similarity measures and metrics.) Metrics satisfy the triangle inequality, which allows us to use reference strings[4] to partition the database: The triangle inequality says, in effect, that "closeness is preserved"; for example, if the query is "close" to

[4]We use "string" when referring to any series of symbols, and "sequence" when referring to those strings arising in some biological context. Thus, both the contents of the database

the reference string, then only those database sequences that are "close" to the reference string need be examined.

In this section, we describe the formal basis of our approach and report the results of some initial experiments to demonstrate its feasibility.

General Scheme

Abstractly, we are given a finite alphabet Σ, and a metric (distance measure) d over Σ^* (finite strings of characters in Σ). Also given is a *database* $\mathcal{D} \subseteq \Sigma^*$. A *query* takes the form: Given a string $q \in \Sigma^*$ and a number ϵ, find all strings in the database whose distance from q is no more than ϵ. That is, determine the set

$$\{s \in D \mid d(q,s) \leq \epsilon\}$$

The two alphabets initially of interest are the four character nucleotide alphabet, conventionally written ($\Sigma = \{G, A, T, C\}$), and the 20 character protein alphabet. However, a variety of other alphabets are important. These include, but are not limited to subsets and power sets of the basic nucleic acid or protein symbols. Examples include purine/pyrimidine alphabets for nucleic acids, and reduced protein alphabets which group amino acid residues based on physical and chemical properties.

A metric is a real-valued function on pairs of points in a space satisfying the following four properties:

1. $d(a,b) \geq 0$.
2. $d(a,b) = 0$ if and only if $a = b$.[5]
3. $d(a,b) = d(b,a)$ — commutativity.
4. $d(a,b) + d(b,c) \geq d(a,c)$ — triangle inequality.

Suppose we choose a reference string r and calculate the distance $d(r,s)$ between the reference string and each string in the database. This (relatively expensive) calculation is done once, and the results are stored.

---

and the query are typically sequences, while the reference strings, which are artifacts of the partitioning algorithm, may or may not be.

[5]For our purposes, it is actually sufficient to replace "if and only if" in this condition with "if" — that is, some distinct strings may be at distance zero from each other. Such a distance measure is sometimes called a pseudometric.

Now, consider a query string q. A database string s which is within ϵ of q will be within $d(r,q) \pm \epsilon$ of r. This is an immediate consequence of the triangle inequality:

$$d(q,s) + d(s,r) \geq d(q,r)$$

so

$$d(q,r) - d(s,r) \leq d(q,s) \leq \epsilon$$

and thus

$$d(q,r) - \epsilon \leq d(s,r)$$

Finally, by commutativity $d(q,r) = d(r,q)$. The derivation in the opposite direction is similar.

Hence, we can restrict our search to those s satisfying the constraint:

$$d(r,q) - \epsilon \leq d(s,r) \leq d(r,q) + \epsilon$$

We will have gained something if the following two conditions hold:

- We can quickly determine those strings s in the database satisfying the constraint.
- "Not too many" strings in $\mathcal{D}$ satisfy the constraint.

There are a number of well understood techniques developed in the computer science community that can be used to satisfy the first condition. Some may be directly applicable; others will require adjustment to the particular details of this problem.

The second condition is more challenging, and much of our research will center on satisfying it. We refer to it as the "good partitioning" condition.

An idealized picture of the technique is shown in Figure 4. The curve is a histogram of database strings against distance. A particular query string q's position is shown, determined by its distance from the reference string r. The shaded band has a width of 2ϵ; only those strings whose distances from r fall within the shaded band can possibly be within ϵ of q. In this particular case, about 25% of the database strings are in the shaded band. If we imagine that the database has been stored sorted by distance from the reference string, the left edge of the band can be found very quickly using a binary search; the comparisons in the search are of numbers (pre-computed distances from the reference string are compared with $d(q,r)$). Since it takes about as long

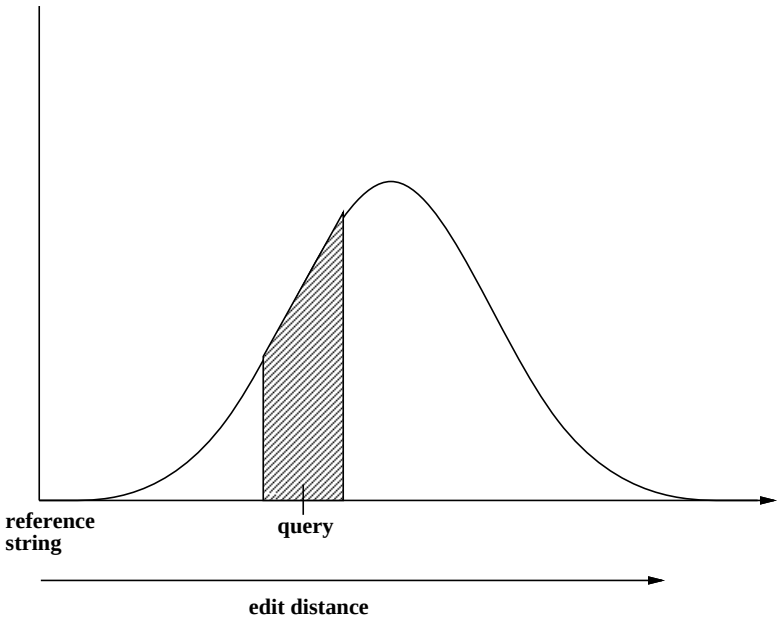

Figure 4: Distribution of Edit Distances

to pre-compute, sort, and store the reference distances as it does to do one brute force search of the database — the distance computations dominate — a net gain will be achieved even after only a few searches.

String Edit Distance

A *string edit measure* $d_{\mathcal{O}}$ is defined over a set of operations $\mathcal{O}$. $\mathcal{O}$ is a set of functions from Σ^* to itself; associated with each member $f \in \mathcal{O}$ is a non-negative *cost* $c(f)$. If $a, b \in \Sigma^*$ and $f_1, f_2, \ldots, f_n$ are operations, and

$$f_n(f_{n-1}(\cdots f_1(a))) = b$$

then we say that the tuple $F = \langle f_1, f_2, \ldots, f_n \rangle$ is a *transformation* of a into b. The cost of a transformation is defined by

$$c(F) = \Sigma\, c(f_i)$$

Finally, $d_{\mathcal{O}}(a, b)$ is defined as the minimum cost of any transformation of a into b.[6]

It is trivial from the definition that any $d_{\mathcal{O}}$ satisfies the first condition for a metric (non-negative value). If all costs are non-zero, it also satisfies the second condition, that a string be at distance zero only from itself.[7] It is easy to see that it also satisfies the triangle inequality: A transformation from a to c can be formed by first transforming a to b, then transforming b to c. The cost of the combined transformation is the sum of the costs, and must be at least as large as $d_{\mathcal{O}}(a, c)$, as by definition it is the minimum over all possible transformations.

In general, $d_{\mathcal{O}}$ need not be commutative. For cases of interest to us, however, it is always the case that:

- If $f \in \mathcal{O}$ then $f^{-1} \in \mathcal{O}$;
- $c(f) = c(f^{-1})$.

It is clear that these are sufficient conditions to ensure that $d_{\mathcal{O}}$ is commutative, and hence a metric.

The most commonly used string edit measure is defined by the class

$$\mathcal{O}_{\lceil\rangle\nabla} = \{del_{ic}, ins_{ic}, rep_{ic}\}$$

where del_{ic} deletes the character $c \in \Sigma$ at position i, ins_{ic} inserts c before position i, and rep_{ic} replaces the character at position i with c. The resulting distance measure is a metric if the same cost is assigned to insertion as to deletion.

A *rep* operation can, of course, be replaced by a deletion followed by an insertion, but in general with a different cost. In fact, insertion and deletion can clearly transform any string into any other string; additional operations affect the metric only if they have different (smaller) costs than the equivalent series of insertions and deletions.

For nucleotide sequences, all replacements are typically given the same cost. Even here, however, the relative cost of replacements as compared to

[6] In the cases of interest, at least one such transformation always exists. In the general case, where none exists we can assign some very large value to the distance, or even extend the notion of a metric to allow infinite distances.

[7] In any case, it is always a pseudometric.

insertions or deletions has been subject to debate with different researchers using different values. For protein sequences, different replacements typically are assigned very different costs. In particular an index could be constructed for each of a number of different PAM matrices (Point Accepted Mutation), which define the substitution costs of amino acid residues. These matrices are constructed by analyzing frequencies of actual replacements in different sets of proteins. Multiple indices would allow efficient queries against a database with respect to a variety of possible replacement costs.

In the case of $\mathcal{O}_{\lceil\rangle\nabla}$, an algorithm based on dynamic programming can calculate the associated distance between two strings in time proportional to the product of their lengths. While high, this cost is not so high as to render brute force searches of current databases impossible. As the set of operations is expanded, however, the distance calculation may become much more difficult. Many variations along these lines have been proposed in the literature. For example, the deletion of a number of adjacent characters may be assumed to be "cheaper" than the simple sum of the individual deletion costs. This is easily modeled by adding a "delete n characters" operation.[8] For nucleotide sequences, it is possible to consider the addition of an operation modeling transposition, and the results of subsequence "inversion" (reversal and complementation of the subsequence). These last two operations are not typically included in the string edit definition, and algorithms to compute the costs are expected to be quite expensive. Nonetheless, the index scheme we propose ensures that the calculations would only be done once, with efficient subsequent queries.

Hamming Distance

The Hamming distance, defined only between strings of equal length, is the number of positions in which the two strings differ. Hamming distance is an example of a string edit distance over a database of strings of fixed length N, with a single replacement operation which, at unit cost, replaces any single character with any other. We consider only queries of length N.

The Hamming distance is of less biological significance than other measures of distance. However, it is easy to analyze mathematically, thus pro-

[8] Of course, a corresponding "insert n characters" operation, with an equivalent reduced cost, must be added to retain commutativity.

viding a simple test case for our approach. We also note that it is related to some current schemes for database searches. For example, FASTP compares sequences using mismatches but not gaps, and other approaches utilize exact matches in hashing algorithms. Let us take as our alphabet the four nucleotides. Suppose that sequences in the database are random: Each position is independently chosen as one of the four bases.[9] If we choose any reference string and compare it to a database sequence, we see that at each position, independently, we have a one in four chance of a match (cost 0), and a three in four chance of a non-match (cost 1). The Hamming distance thus follows a binomial distribution with $p = .75, q = .25$.

How well does Hamming distance satisfy the "good partitioning" condition? For a binomial distribution, the standard deviation is $\sqrt{Npq}$. If we take $N = 100$, this evaluates to 4.33. The distribution is extremely "peaky." The chances are very high that the distance from a query string to the reference string will be very close to the center of the narrow peak, so that even if we chose ϵ as small as 3, almost the entire database would remain to be searched. It's important to keep this example in mind to avoid being led astray by the geometrical diagrams: The metrics we work with do not have the same properties as the familiar Euclidean metric.

These results are valid for random strings. As an experiment, we calculated the distribution of distances to different reference strings from a large extract from the genome database. 10000 sequences, each of length 100, were randomly sampled from the primate collection of GenBank release 74.0 (`gbpri.seq`). The Hamming distance was calculated from a reference string, and a histogram plotted with the frequencies. When a random reference string was used, the results closely match the predictions (Figure 5). However, reference strings consisting entirely of any single base produce much broader distributions (Figure 6). The exact reason for this effect remains to be elucidated; we believe it results from local biases in base distribution in actual sequences, which are known to occur.

[9] This assumption is certainly false for actual genetic sequences, but will serve for our example.

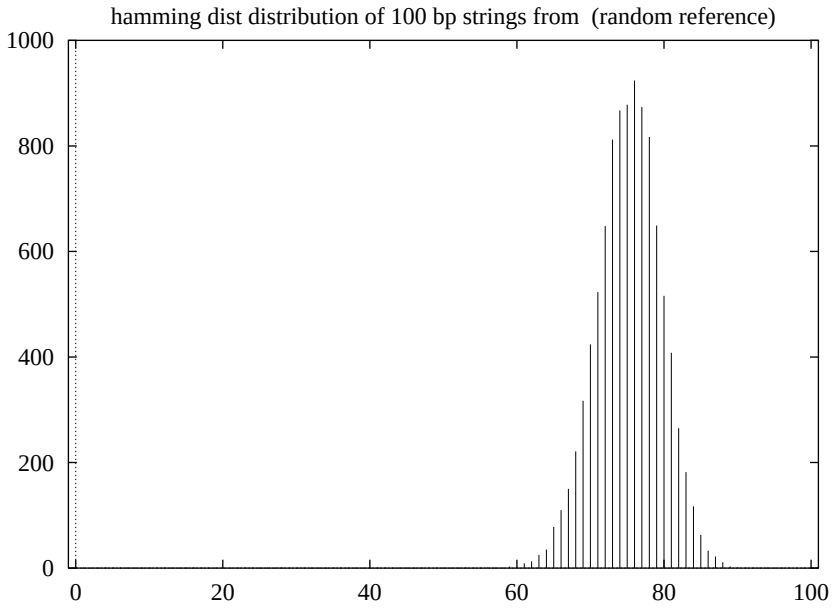

Figure 5: Distribution of Hamming Distances (random reference string)

Edit Distances In the Genome Database

The metric defined by the insertion, deletion, and replacement operations ($\mathcal{O}_{\lceil\rangle\nabla}$) is much less tractable mathematically. We have implemented a version of the basic edit distance calculation using dynamic programming in order to test the distribution. 5253 nucleotide sequences of length 300 were chosen randomly from the same primate collection used in the Hamming calculation reported above. A 300 character sequence was randomly generated, and the edit distance to each of the 5253 randomly selected sequences was calculated, using a mismatch penalty of 2.0 and an insertion/deletion penalty of 2.5. The results are shown in Figure 7. It can be seen from the figure that the distribution is fairly broad. Let us take the worst case: Suppose a reference string falls at the peak, near distance 52. Take $\epsilon = 10$. The sequences to be searched fall at distances from 42 to 62 from the reference string. There are roughly 2000 sequences in this range, under half the database.

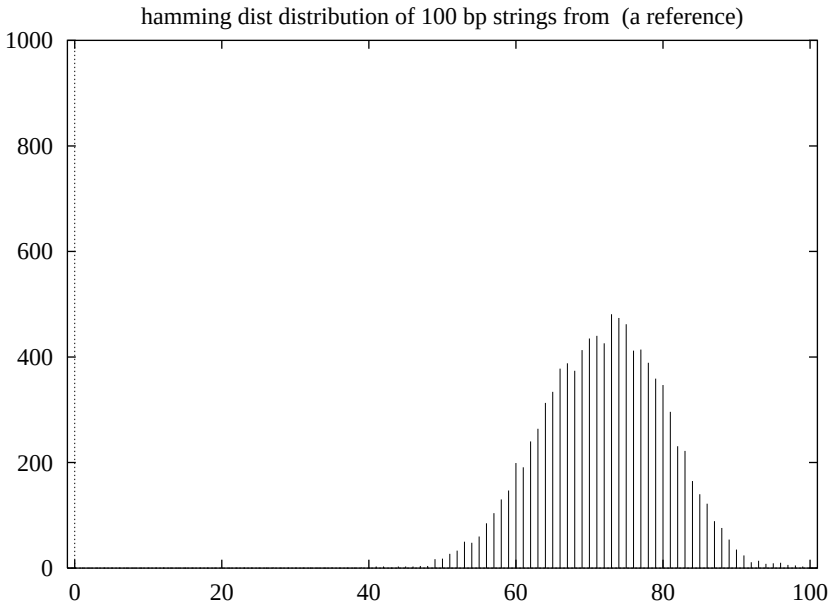

Figure 6: Distribution of Hamming Distances (poly-A reference string)

Multiple Reference Strings

A single reference string is limited in its ability to partition the database, no matter how flat the distance distribution. However, we can choose multiple reference strings and perform the same computations with respect to each. The desired sequences must fall in the intersection of the sequences selected by each reference string. In the best case, distances from distinct reference strings are uncorrelated. If each reference string selects a fraction $f < 1$ of the database, then k reference strings will, among them, select f^k of the database. This exponential improvement implies that we can, by choosing k large enough, arrange to search an arbitrarily small fraction of the database *even when f is close to 1.* For example, even if $f = .9$, with 10 reference strings we would need to search roughly a third of the database. In practice, of course, we would prefer to have f as small as possible; since it contributes exponentially, even a small decrease has a major effect.

Exponential subdivisions of data are the basis of many algorithms. Typ-

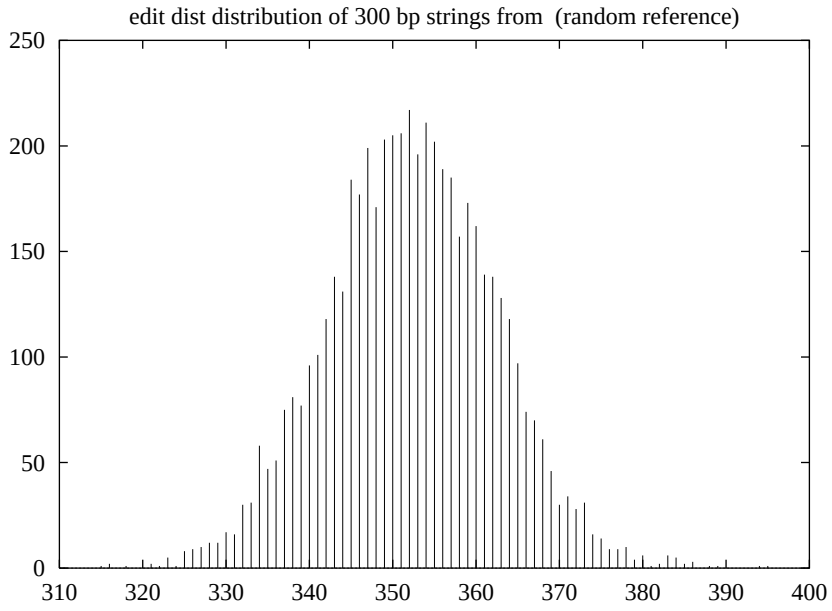

Figure 7: Distribution of Edit Distances

ically, one allows k to vary with the number of members of the database. If k grows as the log of the database size, then for fixed f the number of elements left after the final subdivision remains constant.

An index corresponding to the k reference strings $r_1, \ldots, r_k$ can be implemented using a tree structure. Each interior node in the tree has n children, where n is approximately $1/f$ (but at least 2). For simplicity of presentation, we will assume $n = 2$. Stored in a node at level 1 is a partition value v, and two pointers, $P_<$ and $P_>$, to other nodes. All sequences whose distance from r_i is less than or equal to v will be in the subtree "under" $P_<$ the others will be "under" $P_>$. v is chosen so that about half the sequences actually "under" the current node fall in each subtree.

At the bottom of the tree, the nodes at level k point, not to additional tree nodes, but to *buckets* which contain the actual sequences.

The algorithm for searching such a tree is simple. Very briefly, we maintain a set of nodes (and eventually buckets) to be searched; initially, the set consists of just the root node. For each node in the set at level ℓ, we compute

the distance from the query string to r_ℓ. From this distance and v, we can determine that all possible sequences of interest are in one or the other of the two subtrees, or that both subtrees are possible. We replace the selected node in the set with the one or two subtree nodes and continue. When only buckets remain in the set, we search them. Note that every time we have to search both subtrees, we gained nothing from the computation *at that particular level*. Over *all* the levels, we may still gain a large reduction in the number of sequences to be searched.

K_d Trees: The approach sketched in the previous section is closely related to a structure that has been studied in the computer science literature under the name K_d *tree*. The studies involve research on accompanying algorithms for storing multidimensional data and solving the nearest neighbor problem. K_d trees were originally proposed by Bentley [2] and their applicability to nearest neighbor problems is described in another paper co-authored by him [4]. Generally speaking, the nearest neighbor problem is the search for the N nearest points to given a point with respect to a distance measure. The connection to our problem should be obvious.

Bentley, *et al* go on to describe several optimizations which can be made during the building of the K_d tree to ensure the optimal distribution of sequences in the tree regardless of the distribution in their range. The optimizations include searching the subfile at each node when building the tree to determine which key has the largest spread in values. This key is then chosen as the discriminating key. The median of those values for the discriminating key is then chosen to partition the sequences into subfiles at that node.

K_d trees are particularly intriguing if the data is feared to have a narrow distribution. An additional attractive feature of K_d trees is that they work with many distance measures. Because of the work invested in distributing the data in the K_d tree, it is reported that the expected searching time is proportional to $\log N$. Unfortunately, this speed doesn't come for free, since the time to build the tree is $dN \log N$ where d is the dimension or number of keys.

Multiple Dimensions: A K_d tree works just as well if different distance measures are used at different levels of the tree. For example, we might use different string edit costs in the computations at different levels. It may be

possible to ensure better subdivision by building a tree of this form. The tradeoff is that it is no longer easy to describe exactly what constraint the sequences found will satisfy. It will take some experimentation to determine whether the resulting match sets are useful.

Linear Combination Technique

Consider again the Hamming distance. It results in a highly peaked distance distribution. Recall the width of the peak is $\sqrt{Npq}$. For fixed N, this expression is maximized for $p = q = .5$.

Consider two sets of abstract alphabets over the four letter alphabet $\{A, C, G, T\}$ as follows. Let:

$$X = \{A, C\}, \quad Y = \{G, T\}, \quad U = \{A, G\}, \quad V = \{C, T\}$$
and
$$\sigma_1 = \{X, Y\}, \quad \sigma_2 = \{U, V\}$$

We define two distances, d_1 and d_2, among letters as follows:

$$\begin{array}{lll} d_1(I,J) = 0 & \iff & I, J \in X \text{ or } I, J \in Y, 1 \qquad otherwise \\ d_2(I,J) = 0 & \iff & I, J \in U \text{ or } I, J \in V, 1 \qquad otherwise \end{array}$$

Clearly, both d_1 and d_2 are distances. Consider the case when one of the letters is A and the other is unknown, say α.

$$\begin{array}{lllll} d_1(A,\alpha) = 0 & \wedge & d_2(A,\alpha) = 0 & \Rightarrow & \alpha = A \\ d_1(A,\alpha) = 0 & \wedge & d_2(A,\alpha) = 1 & \Rightarrow & \alpha = C \\ d_1(A,\alpha) = 1 & \wedge & d_2(A,\alpha) = 0 & \Rightarrow & \alpha = G \\ d_1(A,\alpha) = 1 & \wedge & d_2(A,\alpha) = 1 & \Rightarrow & \alpha = T \end{array}$$

Knowing that the first letter is A, each of the outcomes is equally likely. Hence,

$$\begin{array}{lcl} Prob[d_1(I,J) = i \wedge d_1(I,J) = k \mid I = I_0] & = & \frac{1}{4} \\ \sum_{I_0 \in A,C,G,T} Prob[d_1 = \ldots \mid I = I_0] & = & \frac{1}{4} * Prob[d_1 = \ldots] \end{array}$$

The 2-dimensional distribution of d_1 and d_2 is therefore uniform, so the two measures are uncorrelated. Further, each has the desired $p = q = .5$ probability distribution.

This construction indicates that it may be possible to modify a given distance measure in such a way that:

- No information is lost: The sequences found correspond exactly to the sequences that would have been found using the original distance measure;
- The probability distribution with respect to the modified measures is improved, so that the search is faster.

Project Overview

The research to be done includes three distinct, but interelated, components:

- Study the theoretical basis for algorithms for rapid search of biological sequence databases;
- Produce implementations of algorithms suggested by these theoretical analyses and study their performance on real biological databases and practical, available hardware and software systems;
- Make the implementations available to selected researchers in the biological community, and determine their usefulness, strengths, and weaknesses.

We expect to re-visit each of these components repeatedly during the course of the research. For example, there are at least two reasons why algorithms that look good in theory may not be useful in practice. First, the model they assume for the biological sequence data may be inappropriate — for example, they may assume a degree of randomness that is not present in the data, and the discrepency may be sufficient to have a serious impact on performance. Secondly, they may not be well matched to existing hardware. To illustrate, if we find it desirable to consider parallel machines, there are algorithms that do well on small numbers of processors but do not scale well to larger numbers of processors for complex reasons. Another practical reason to re-examine an algorithm is that it may provide a very good answer to a question that biologists find they don't wish to ask very often. To take a more optimistic view, we may find that we answer important questions — which simply inspire researchers to ask different and previously unanticipated questions. Such changes of focus are an inherent part of any active research area, and it is impractical to attempt to formulate in detail all the questions that will ever be asked.

In the following sections, we discuss each of the three components listed above, and indicate our approach to it.

Theory

The computer science literature is rich with theoretical studies of what can be included under the general rubric of search algorithms. Some traditional

references include [9] for a variety of techniques, and [10] for databases. There is active research in such areas as geometric databases [17], distributed databases, database-like languages [12], and many others. While some general principles are known, specific problems impose significant constraints on the set of algorithms which may be appropriate. Thus, the general idea of dividing a database with respect to a set of reference points appears in many forms, ranging from B-trees, used in traditional business databases, to K_d-trees, used in geographical databases. However, the choice of reference points, and the exact way they are used, is highly specific to the particular domain. The algorithm we propose for biological sequence databases is in this broad class, but has a detailed structure different from previously studied algorithms.

Implementation

While theoretical work can be used to choose algorithms with good behavior with respect to mathematically tractable models, there is no guarantee that these models will be a sufficiently accurate representation of the real world. Because of the effort involved in building an implementation of the algorithms we wish to study, our approach is to rely on realistic simulation; we will implement the core elements of the algorithms, then apply them to a large extract of an existing sequence database. For example, we have built simple implementations of Hamming and string-edit distance modules, which were then used to study the distribution of these distances. The distributions examined were distances between a reference string and randomly selected substrings from the GenBank primate collection. This empirical study has confirmed theoretical calculations that Hamming distance is not a suitable basis for the construction of the databases we have in mind. On the other hand, string edit distance, which is of much greater biological interest, probably is.

Application

An important goal of this research is to produce a system that will actually be useful to biological researchers in their work. The result of the previous component will be the operational kernel of a database system. To make it useful to biologists, we will design a suitable query language in which they can

phrase their questions, and implement an interface to the operational kernel that embodies it. Initially, we will work with a small number of biologists, perhaps two or three, who will be willing to deal with a system still in its infancy and actively participate in helping us improve it. Later, we will expand the community of users to a larger group of active researchers.

The goals of this component of the effort can be summarized as the production of a *robust*, *usable* system which can provide *timely answers* to *biologically interesting questions*. Critical to the success of this component is the combination of direct interaction of the developers with the users of the system and analysis of measurements of system performance in actual use.

Goals of Proposed Research

The PROXIMAL system aims to achieve the following goals:

- Design and build prototypes for an information retrieval system for molecular biological sequence information:
  - Design an index scheme to find sequences "close" to a given query sequence where "close" is defined in terms of known string edit functions;
  - Create a method of incrementally updating the index so that the addition of new sequences requires little computational overhead;
  - Design a query language for more complex queries;
  - Compile queries into fast programs that implement searches using the index design;
  - Integrate sequence retrieval with other (more traditional) retrieval of associated English text annotation.
- Experiment with and tune prototypes for both nucleic acid and protein sequences.
- Make retrieval system available to the molecular biology community.
- Determine whether the prototypes are useful to their intended audience and modify them as necessary to make them useful.

References

[1] S.F. Altschul, W. Gish, W. Miller, E.W Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[2] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] R.F. Doolittle et al. Simian sarcoma virus onc gene, V-sis, is derived from the gene (or genes) encoding a platelet-derived growth factor. *Science*, 221:275–277, 1983.

[4] J. H. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[5] G. H. Gonnet. *Handbook of Algorithms and Data Structures.* Addison-Wesley, London, 1984.

[6] G. H. Gonnet, M. A. Cohen, and S. A. Benner. Exhaustive matching of the entrie protein sequence database. *Science*, 256:1443–1445, June 1992.

[7] Greg Hamm. personal communication.

[8] P. Klier and R. J. Fateman. On finding the closest bitwise maches in a fixed set. *ACM Transactions on Mathematical Software*, 17(1):88–97, 1991.

[9] Donald E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, Reading, Mass., 1973.

[10] Henry F. Korth and Abraham Silberschatz. *Database System Concepts.* McGraw Hill, New York, 1986.

[11] E. S. Lander, R. Langridge, and D. M. Saccocio. Computing in molecular biology: Mapping and interpreting biological information. *IEEE Computer*, pages 6–13, November 1991.

[12] Jerrold S. Leichter. Shared tuple memories, shared memories, buses and LAN's — Linda implementations across the spectrum of connectivity. Technical Report TR-714, Yale University Department of Computer Science, July 1989. Also a 1989 Yale University PhD thesis.

[13] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.

[14] A.M. Maxam and W. Gilbert. A new method of sequencing DNA. *Proc. Natl. Acad. Sci.*, 74:560–564, 1977.

[15] P. L. Miller, P. M. Nadkarni, and N. M. Carriero. Parallel computation and FASTA: Confronting the problem of parallel database search for a fast sequence comparison algorithm. *Computer Applications in the Biosciences*, 7(1):71–78, January 1991.

[16] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:444, 1970.

[17] Hanan Samet. *Applications of Spatial Data Structures.* Addison-Wesley, Reading, Mass., 1989.

[18] F.S. Sanger, S. Nicklen, and A.R. Couson. DNA sequencing with chain terminating inhibitors. *Proc. Natl. Acad. Sci.*, 74:5463–5467, 1977.

[19] David Sankoff. Time warps, string edits, and macromolecules: The theory and practice of string comparison. 1983.

[20] P. Sellers. Theory and computation of evolutionary distances. *SIAM Journal of Applied Mathematics*, 26:787, 1974.

[21] J. Shavlik, G. Towell, and M. Noordewier. Using artificial neural networks to refine existing biological knowledge. *International Journal of Human Genome Research*, 1:81–107, 1992.

[22] W. Taylor. Sequence analysis: Spinning in hyperspace. *Nature*, 353:388–389, 1991.

[23] G. Vogt and P. Argos. Profile sequence analysis and database searches on a transputer machine connected to a Macintosh computer. *Computer Applications in the Biosciences*, 9(1):25–28, February 1993.

[24] M. S. Waterman. General methods of sequence comparison. *Bulletin of Mathematical Biology*, 46(4):473–500, 1984.

[25] M.S. Waterman, T.F. Smith, and W.A. Beyer. Some biological sequence metrics. *Adv. Math.*, 20:367, 1976.

[26] J.D. Watson and F.H.C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953.