SQL Extensions and Database Mechanisms for Managing Biosequences

Willard J Briggs, Wenguo Liu, Rui Mao, Weijia Xu, and Daniel P. Miranker
Department of Computer Sciences, the University of Texas at Austin
{willard, liuwg, rmao, xwj, miranker} @cs.utexas.edu

Abstract

Biologically effective retrieval and analysis of sequences entails much more than finding matching strings. While identification and storage of biological sequences usually comprises long functional units (e.g. genes, proteins and chromosomes), the analysis and retrieval of those sequences is primarily concerned with finding ordered sets of short matching subsequences (q-grams). This characterization applies both to homology search algorithms, i.e., BLAST searches, as well as a growing toolkit of algorithms in comparative genomics that are tantamount to executing joins on pairs of whole genome sequences (whole genome joins).

To support these two logical views of sequence data, we introduce mSQL, a set of extensions to SQL92. We have implemented mSQL as a component of MoBIoS, the Molecular Biological Information System. We describe the materialization of sets of q-grams as a metric-space index. Such a physical structure provides an access path for indexed-nested loop joins, enabling O(mlogn) comparative genomic analysis. We detail the optimization of paged MVP-trees to support a metric for the retrieval of protein sequences. Empirical results demonstrate O(logn) retrieval times for local alignments.

1. Introduction

Across life sciences, exploration of key data comprises sequentially scanning all of the data that matches even the most basic patterns. This includes genomic and proteomic sequence data (BLAST searches), proteomic mass-spectra and the combinatorial chemistry libraries critical to rational drug design [3, 30, 35, 36]. We are seeking to solve this in a universal way by developing a metric-space database management system, MoBIoS (*Molecular Biological Information System*) [33]. By analogy to spatial databases which extend relational databases by including special access paths for two and three dimensional data types, we define a metric-space database as an extension of a relational database that includes access paths based in metric-space indexing [7, 8, 9, 11, 13].

MoBIoS contains built-in biological data types entailing the semantics of biological dogma and includes metric distance functions enabling O(log n) access-times. Our claim is that the tight integration of biological search predicates into the query engine will simplify the process of bioinformatics discovery by enabling a single programming model for both relational and biological data. Common database application development environments could be used directly without resorting to the use of external pattern-matching utilities, i.e. BLAST [12, 33, 43].

In this paper we detail the components of MoBIoS that are specific to managing biological sequences in a manner that addresses the subtleties required to be biologically effective. The literature on biological sequence analysis speaks of two kinds of alignments, global and local alignment, but only local alignment is of practical interest to biologists. Global alignment is more commonly known in computer science as weighted edit distance. As it applies to biology, the weights between individual pairs of character substitutions entails a model of sequence evolution. Comparing sequences using simple (unweighted) edit distance rarely yields biologically interesting results [44].

A local alignment of two sequences S and T, comprises finding an ordered set of substrings of S, si, and an ordered set of substrings of T, tj, such that the sum of pairwise global alignments over si, tj, is maximal, (or minimal, if distances are used in lieu of similarity scores).

Analysis involving the convolution of two or more complete genomes, (*whole genome join*), is a problem of primary and increasing significance. Each time a new organism is sequenced it must be mapped. This means that the sequence is annotated with the location and, if possible, the function of each gene as well as a number of other important features. As the corpus increases this is more commonly being accomplished by locally aligning the entire new sequence with all previously mapped sequences and deducing that similar substrings have similar function [22]. This is just the tip of the iceberg. With the availability of the data, new genomic analysis protocols requiring whole genome joins are being developed at an increasing rate [32, 37, 29, 41].

In each of these cases the unit of storage and identification is a long sequence, e.g. a gene, protein or chromosome, but the analysis and retrieval is based on finding scored sets of matching substrings. To resolve the dichotomy between these views, we introduce two new operators: *createfragments(),* and its complement, *merge()*. *Createfragments()* decomposes a sequence into fixed length overlapping substrings, a form of q-grams [20]. *Merge()* reconstitutes these q-grams into sequences. Algebraically these are treated very similarly to the unnest and nest operators of an extended-relational algebra [26].

To programmatically handle the two views and to enable a persistent materialization of the q-grams as a special access path to the full sequences we introduce *sequenceviews*. See Section 3.

We have already used these language developments in an application in comparative genomics. In Section 4, we describe how we performed a search for conserved primer pairs between two genomes, showing that it is possible to express important genomic analysis problems in small SQL programs [46]. We also demonstrate how MoBIoS can be used to compute homology searches similar to BLAST.

In Sections 5 and 6, we detail the characteristics of biological workloads and argue that in this domain this form of materialization is clearly the best. We present the results of a performance where we evaluated an instance from each of the three major classes of metric-space indexing algorithms and show that, for our applications, multi-vantage point (MVP) trees perform the best. We detail the optimization of our implementation of an MVP-tree, its execution speed and its accuracy on a yeast homology benchmark suite developed at NCBI [39]. The results reveal scalable execution speeds tending to O(log n) and accuracy comparable to BLAST.

2. Related Work

The hyper-exponential growth and increasing importance of biological sequence data to the conduct of research motivates new approaches of biological sequence management where sequence data is preprocessed off-line and organized in data structures such that on-line queries can be executed quickly.

There has been recent activity involving metric-space indexing methods. The SST system reports 1 and 2 order of magnitude speed improvements over BLAST [16]. SST comprises nearest-neighbor searches based on Hamming distance using a vector space mapping of q-grams and a TSQV tree. Results were reported only for sequence assembly. Chen and Aberer proposed a system composed of M-trees and a metric upper-bound on local alignment scores. No performance figures have been published. The most far-seeing work was a paper by Wang and Shasha in 1990, which was one of the earliest to propose algorithms to search metric spaces by precomputing and storing selected distances [47]. They further characterized the metric-join problem and presented results on finding similar proteins for a set of 151 proteins made up of less than 21 amino acids. The metric comprised a normalization of the PAM distance made possible by first computing all pair-wise distances.

A challenge in these approaches is that biologically effective results must include a model of similarity based on evolutionary or other biochemical properties. Most often the model is captured as a set of weights in a substitution matrix; most notable are the PAM and BLOSUM matrices for amino acid substitution [14]. Even BLASTn uses a weight matrix to generate the scope of the hot-spot neighborhoods around the query.

What is most vexing is that query results are very sensitive to the weighting model. This should not be surprising. Even though there is no explicit dimensionality to this problem, the curse of dimensionality still applies [5]. Thus, just slight distortions of the space due to an inaccurate weighting model can lead to dramatically different answers [1, 27]. This property also makes it difficult for any system that leverages an upper bound on distances to produce effective results.

Our work is predicated on our earlier results where we developed the first biologically validated metric substitution matrix, mPAM [44]. In that paper we compared mPAM to other metrics, including simple edit distance, and showed that simple edit distance does not product biologically effective results for most applications. This is one reason SST has not been extended to a more general algorithm.

The database effort that is closest to our work in structure is by Gravano et al. [18]. That effort comprised a pure SQL system where auxiliary q-gram tables were materialized at runtime and string-matching operations could be conducted using standard queries. They further developed an approximate string matching method that concerned exploiting bounds concerning the density of matching q-grams. Like SST, the lack of an evolutionary model prevents this from being generally applicable to biology. The decision by Gravano et al. to devise a q-gramming method to work on top of already existing databases does not allow for the use of complex weight matrices without resorting to computationally cumbersome User Defined Functions. Furthermore, although mentioned as a possibility, metric-space indexing was not pursued.

A number of efforts have introduced inverted indexes on q-grams [10, 28, 34, 42, 48]. These systems are proving to be very fast and useful when used as either a coarse filtering mechanism or when applied to genomic analysis problems on evolutionarily close sequences [37]. Another specialized index structure, the suffix-tree, has also been tested with respect to both accuracy and scalability on very large data sets [25, 30].

To support query optimization, any system that promises to tie an SQL engine to physical access plans must have an algebra. In that regard this work utilizes elements of both an extended-relational algebra for complex types and the PiQA algebra for querying protein sequences [26, 43]. The relationship of our work to these algebras will be spoken of in the paper.

3. mSQL

We have coined the term *mSQL* to describe our keyword and operator additions to the SQL92 specification. We will use the explanation of a simple query to find homologous regions in two genomes as a running example to detail the merits of the system. The schema for a simple database is defined in Figure 1.

```
CREATE TABLE genomes(
    Organism VARCHAR,
    Acc_num INTEGER,
    DNA_Sequence JAVA_OBJECT(DNA),
    CONSTRAINT PK_genomes PRIMARY KEY (Acc_num));
```

Figure 1 Schema definition for example

In addition to the standard SQL data types, MoBIoS includes built-in data types for DNA and Protein sequences, as well as Spectra (to support Mass Spectroscopy analysis). In the example, we assume for simplicity that each chromosome sequence has been assigned a unique id, often called its accession number. We will also assume that DNA_Sequence is a string. Its complete definition entails biological semantics beyond the scope of this paper (e.g. equivalence of reverse-complements). We will be looking for homologous regions of length q between the Rice and Arabidopsis genomes that differ by at most n nucleotides. For brevity, the Rice 'genome' is illustrated as two functional units (possibly representing chromosomes or genes) and is 11 nucleotides long. The Arabidopsis 'genome' is just 6 nucleotides long. Our database is populated as shown in Table 1.

Table 1 Example database

genomes

Organism	Acc_num	DNA_Sequence
rice	1	AGAAC
rice	2	CCGGAT
arab	3	AACAAC

From the logical perspective, our primary contributions are two new operators: *createfragments()*, its complement, *merge()*. We also introduce a new type of view, *sequenceview*.

Createfragments() decomposes sequences into overlapping substrings, also known as q-grams. Algebraically, *createfragments()* is the unnest operator tailored to sequences in lieu of sets. If the substrings did not overlap, the behavior would be identical to unnest applied to a representation of sequences as a set of substrings.

Merge() maps sets of gap-free q-grams back into larger sequences. There are two forms of the *merge()* operator: A one-dimensional case where *merge()* is applied to the q-grams of a single attribute possibly filtered through a select predicate, and a two-dimensional case where *merge()* is applied to the results of a string matching join between two attributes. Although the semantics may be obvious, completeness of the algebra requires us to include a *groupfragments()* operator which serves to ensure that *merge()* is applied only to groups of q-grams from the same sequence and therefore can be merged.

A sequenceview is a physical database construct analogous to SQL's *view*. Explicit in a *sequenceview* is the materialization of *createfragments()* as a secondary metric-space index.

3.1 *Createfragments()*: Unnesting Sequences

```
SELECT *
     FROM CREATEFRAGMENTS(genomes.DNA_Sequence, 3)
     WHERE Organism = 'rice';
```

Figure 2 *Createfragments()* syntax

Createfragments() takes two arguments: a table name followed by the sequence attribute (in dot notation) and the length of the substring. The specification of a sequence attribute helps maintain generality in the case where a table is defined with more than one sequence attribute. The second parameter serves the purpose of defining a sliding window on the sequences. For our example, we will be looking for homologous regions of length three, so we define the length of the q-gram to be three. Thus the query in Figure 2 on the genomes table in our example database would yield the result shown in Table 2.

Table 2 *Createfragments()* result

CREATEFRAGMENTS(genomes.DNA_Sequence, 3)

Organism	Acc_num	DNA_Sequence
rice	1	{0, AGA}
rice	1	{1, GAA}
rice	1	{2, AAC}
rice	2	{0, CCG}
rice	2	{1, CGA}
rice	2	{2, GAT}
rice	2	{3, ATT}

The details of the mapping mechanism are as follows. Assume a set of sequences, S = {s_1, s_2, …, s_n}, with lengths l_1, l_2, …, l_n. The fixed fragment length is q. Thus, each sequence s_i can be split into $m_i = l_i - q + 1$ fragments. We can compute the total number of fragments $m = \sum m_i$. We number the fragments from 1 to m. Then, given the number of a fragment, we can compute which sequence it is from and its offset in the sequence. For example, we have two sequences s_1="agaac" and s_2="ccggat" of lengths 5 and 6 respectively. The fragment length is 3, and we start a fragment for each character. Thus, s_1 can be split into 3 fragments and s_2 can be split into 4 fragments. The total number of fragments is 7. We number the fragments 1, 2, …, 7. The 6th fragment is from s_2 at offset 2, i.e., "gat". This can be verified in the results from the *createfragments()* operation shown in Table 2.

On the surface, *createfragments()* may appear similar to PiQA's match operator, since both divide a sequence into a set of subsequences. However, in PiQA scoring is done at the match level, such that the match operator takes a set of strings and a string or a regular expression and returns a set of matches. The *createfragments()* operator is much more general in that it is completely separated from any scoring system, i.e., 'matching' is only performed once the *createfragments()* operation has been completed. This allows, for example, the possibility of a two-dimensional merge join.

3.2 *Sequenceview*

By logically representing sequences as tables of q-grams we can use built-in join operators to compare the contents of the sequences to each other. This by itself is not new. Gravano et.al. explicitly materialized q-grams in an auxiliary table [20, 21, 38] such as the one shown in Figure 4. However, materializing q-grams as auxiliary tables at runtime is simply not feasible for strings representing entire genomes. The lengths of the actual Rice and Arabidopsis genomes are ~5×10^8 bp and ~1.5×10^8 bp respectively. With a q-gram size of 18, (the size used in our search for conserved primer pairs), we would be looking at generating ~6.5×10^8 rows of 18 characters each, or ~9.55×10^9 extra characters.

Furthermore, once a *sequenceview* is materialized, it may be accessed multiple times—reducing the impact of the cost of the metric-index build and giving O(log n) access times. Figure 3 illustrates the creation of *sequenceviews* for the example problem.

```
1.CREATE SEQUENCEVIEW rice_sview AS
2.    SELECT *
3.    FROM CREATEFRAGMENTS(genomes.DNA_Sequence, 3)
4.    WHERE Organism = 'rice'
5.USING base_pair_mismatch;

6.CREATE SEQUENCEVIEW arab_sview AS
7.    SELECT *
8.    FROM CREATEFRAGMENTS(genomes.DNA_Sequence, 3)
9.    WHERE Organism = 'arab'
10.  USING base_pair_mismatch;
```

Figure 3 Creation of *sequenceviews*

A *sequenceview* is comprised of two major elements:

1. Sequences that are to be included in this *sequenceview* are derived in the standard method involving a SQL query, including the explicit decomposition of the sequences into q-grams using *createfragments() (lines 2-4).*
2. A [metric] index is specified as the access-path for the q-grams *(line 5).*

Part of the generality of metric-space indexes is that the development of the index mechanism is independent of the metric-distance function (metric). In addition to the usual arguments to create a secondary index, the creation of a metric-space index is parameterized by the choice of metric.

Sequences are traditionally stored as long characters. Once *createfragments()* splits the sequences into a set of fragments based on the input parameter of fragment length, a metric-space index tree can be built over these fragments. This index tree can be used to accelerate matching of fragments. The details of materializing *sequenceviews* are described in Section 5.

3.3 *Merge()*: One-Dimensional Case

Given a set of q-grams, it is necessary to assemble them back into longer sequences. The set of q-grams may have been the argument to relational operators, and not all of the q-grams of the original sequence may be present in the computed result (e.g., q-grams may have been selected on external annotations such as their location within a chromosome). Informally, if two q-grams overlap or merely adjoin with respect to the original sequence we wish to merge them into a longer sequence. The one-dimensional merge operation takes one argument: a table name followed by the sequence name to merge on, in dot notation.

Definition 1 One-dimensional merge: A q-gram with parameters k and m is a substring of a sequence of the form $s_k, s_{k+1}, s_{k+2}, ..., s_m$. Two q-grams q_1 (with parameters k_1 and m_1) and q_2 (with parameters k_2 and m_2) are **mergeable** and can be merged into one longer q-gram q_3 (with parameters k_3 and m_3) iff

- q_1 and q_2 have the same parent sequence,
- $(k_1 \geq k_2$ and $k_1 \leq m_2)$ or $(k_2 \geq k_1$ and $k_2 \leq m_1)$,
- $k_3 = \min(k_1, k_2)$,
- $m_3 = \max(m_1, m_2)$.

The merge of two q-grams q_1 and q_2 can be expressed as $q_3 = \text{merge}(q_1, q_2)$.

It is easy to see that the one-dimensional *merge()* is the inverse of the *createfragments()* operation. Thus, merge(createfragments(*s*))=*s*, where *s* is any set of sequences. In SQL terms, the query:

```
SELECT MERGE(cf.DNA_Sequence)
FROM CREATEFRAGMENTS(genomes.DNA_Sequence, 3) AS cf;
```

returns the contents of the original 'genomes' table.

3.4 *Merge()*: Two-Dimensional Case

A metric-space join is similar in concept to a spatial join. The goal is to determine pairs of objects that fulfill a certain distance predicate. In Figure 4, we illustrate the final SQL program for identifying q-grams from our two genomes that differ by at most one nucleotide.

```
SELECT MERGE(R.fragment, A.fragment)
    FROM rice_sview as R, arab_sview as A
     WHERE    distance('base_pair_mismatch',    R.fragment,
           A.fragment) <= 1.0
GROUP BY R .fragment, A.fragment;
```

Figure 4 Metric Join

Analogous to spatial extensions of SQL, mSQL allows distance predicates for selects and joins. The syntax has been expanded to allow the specification of a metric. In addition to built-in metrics, users may extend MoBIoS with their own definitions of new metrics.

In the final result, we are interested in pairs of highly similar regions, not collections of q-grams. Overlapping q-grams can only be merged if they exhibit like overlap in both genomes. Thus, we have a two-dimensional *merge()* operation. If we want to merge the results from some join operation, the *merge()* operation actually happens in each separate sequence, but needs to take into account the offset difference.

The two-dimensional *merge*() operator has two forms, each taking either two or four arguments. The first two arguments are always the attribute names for the fragment columns to be merged, the third argument is an (optional) gap function for similarity computation between two fragments, and the fourth argument is the (optional) threshold used to filter the merged fragments within similarity specified by a numerical value. See Figure 5.

Join Results (after GROUP BY)

Org.	Acc_ num	DNA_ Sequence	Org.	Acc_ num	DNA_ Sequence
rice	1	{0, AGA}	arab	3	{1, ACA}
rice	1	{1, GAA}	arab	3	{2, CAA}
rice	1	{2, AAC}	arab	3	{3, AAT}
rice	1	{2, AAC}	arab	3	{0, AAC}
rice	2	{2, GAT}	arab	3	{3, AAT}

MERGE(R1.DNA_Sequence, R2.DNA_Sequence)

Org.	Acc_ num	DNA_ Sequence	Org.	Acc_ num	DNA_ Sequence
rice	1	{0, AGAAC}	arab	3	{1, ACAAT}
rice	1	{2, AAC}	arab	3	{0, AAC}
rice	2	{2, GAT}	arab	3	{3, AAT}

Figure 5 Results of *merge()* operation

Definition 2 Two-dimensional merge(): Two pairs of q-grams (q_1, q_2) and (q_3, q_4) are mergeable and can be merged into one longer q-gram pair (q_5, q_6) iff (assume q-gram q_i has parameters k_i and m_i)

- q_1 and q_2 are of the same length, and q_3 and q_4 are of the same length,
- q_1 and q_3 are mergeable,
- q_2 and q_4 are mergeable,
- $k_3 - k_1 = k_4 - k_2$,
- $q_5 = \text{merge}(q_1, q_3)$,
- $q_6 = \text{merge}(q_2, q_4)$,
- if a gap function *g* and threshold *d* are specified, then $g(q_5, q_6) < d$,
- q_5 and q_6 are of the same length.

Only when the offset differences are the same can the fragments be merged. In Figure 7, for the first row and second row of the join results, the offset difference for the fragments of sequence 1 is 1, and the offset difference for the fragments of sequence 2 is also 1. Thus the first two fragments can be merged into one fragment for sequence 1 and sequence 2, respectively.

Through *merge()* operations, the query result size can be reduced without a loss of information.

3.5 *Groupfragments()*

A *groupfragments()* operator is included to ensure that *merge()* is applied only to q-grams derived from the same sequence and can therefore be merged. The *groupfragments()* operator is used to group sets of fragments, where each group can be merged into exactly one larger piece of fragment.

Definition 3 Fragment group: A set S of q-grams $\{q_1, q_2, ..., q_m\}$ is in the same fragment group iff

- S is of size 1, or
- $\forall q_i \in S, \exists q_k \in S\ (i \neq k)$, q_i and q_k are mergeable.

The SQL92 *GROUP BY* keyword is overloaded to represent the *groupfragments()* operator as illustrated in Figure 4.

The *groupfragments()* operator is quite different from the traditional SQL group operator. The traditional group operator is based on equality, i.e., only equal objects should be in the same group. *groupfragments* is based on the mergeable property of different objects as defined in the prior section.

Merge() and *groupfragments()* also have an analog in the PiQA algebra, where they are implemented as a single match extension operator [43]. However, as in the case with the match operator, the PiQA match extension operator does not consider querying in high-dimensional cases—joins, for example.

4. Application Examples

Two application examples (conserved primer pair discovery and homology search) are given in the following sections to justify the functionality of *sequenceview* and the operators defined above.

4.1 Conserved Primer Pair Discovery

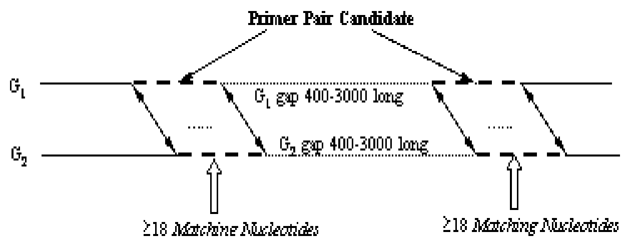

Figure 6 Finding conserved primer pairs

The query involves joining the genomes of Rice and Arabidopsis in search for shared substrings that fulfill a number of experimental properties. (In biological terms, we are seeking universally conserved PCR primer pairs spanning a quickly evolving region.) The pattern-based description of the goal is to find shared substrings between genomes, such that a substring 400-3000 nucleotides long is bounded by a pair of substrings at least 18 nucleotides long. The pair of short bounding substrings must have a minimal number of mismatches between the two species, specified in the query. The long middle stretch does not need to match between the genomes or be the same length. The schema is the same as outlined for our sample database in Figure 1. The specification of the problem and the mSQL query are shown in Figures 6 and 7.

Note that the mSQL query for this problem is only slightly more complex than that for our simple example. Problems of this kind are increasingly frequent and researchers continually implement ad-hoc computer programs to solve it [32, 46]. We solve the problem in four steps, using the MoBIoS platform to conduct the analysis:

```
1.SELECT merge(R1.DNA_Sequence, A1.DNA_Sequence)
2.FROM rice_sview as R1, rice_sview as R2, arab_sview as A1,
    arab_sview as A2
3.WHERE
4. distance('base_pair_mismatch', R1.DNA_Sequence.fragment,
A1.DNA_Sequence.fragment) <= 1.0 AND
5. distance('base_pair_mismatch', R2.DNA_Sequence.fragment,
         A2.DNA_Sequence.fragment) <= 1.0 AND
6. R2.DNA_Sequence.offset - R1.DNA_Sequence.offset >= 400
      AND
7. R2.DNA_Sequence.offset - R1.DNA_Sequence.offset <= 3000
      AND
8. A2.DNA_Sequence.offset - A1.DNA_Sequence.offset >= 400
      AND
9.  A2.DNA_Sequence.offset - A1.DNA_Sequence.offset <= 3000
10.GROUP BY R1.DNA_Sequence, A1.DNA_Sequence;
```

Figure 7 mSQL query for conserved primer pair discovery

- Create *sequenceviews rice_sview* and *arab_sview* on the rice and arab genomes, respectively. This is the same as for our sample in Figure 5, except the fragment length is set to 18 instead of 3.
- Utilize the metric-space index to metric join *rice_sview* and *arab_sview*, allowing at most a one nucleotide mismatch in primer regions *(lines 4 and 5)*.
- Ensure the conserved regions are 400-3000 nucleotides apart *(lines 6 through 9)*.
- Merge the resulting q-grams into long sequences *(line 1)*.

Optimization can be done at the metric join step with the help of a metric-space index defined on two *sequenceviews rice_sview* and *arab_sview*. Our current implementation comprises indexed nested-loops. The problem was solved in less than 2 days using four concurrent processes on a Sun 6800, and we anticipate a join-operator inspired by merge-join to solve this problem in a few hours on a single processor. The results are currently being validated in a wet lab.

4.2 Homology Search

Homologous sequence search is a basic task performed in bioinformatics where homologous means high similarity between sequences. The similarity score for two genomic sequences is calculated on pair-wise alignment, where local alignment is commonly used. For a genomic sequence database, homologous search can be viewed as a range query.

Definition 4 Homology search problem: Given a sequence *q* and a collection of sequences *S*, the homology search problem, *HS(S, q, d)*, is to identify all sequences $s \in S$ where LocalSimilarity(s, q)<d.

Note that there is an intrinsic exponential in the size of a local alignment problem in which any ordered subset of the elements of *S* should be considered.

The general framework for computing local-alignments by matching q-grams was first proposed and analyzed by Myers, contemporaneously with the development of BLAST [49, 34]. The basic algorithm consists of building a sequence database *S* offline, (i.e., creating a *sequenceview S_sview)*, and then performing the online search query. The search query will be converted into a set of range queries, based on q-grams of the query sequence, whose results will be merged together to form the final answer. Given *sequenceviews S_sview and q_sview*, for a collection of sequences *S* and query sequence *q*, respectively, the mSQL query to solve the homology search problem is illustrated in Figure 8.

```
SELECT merge(R.fragment, A.fragment, g, d)
    FROM S_sview as R, q_sview as A
    WHERE distance(metric_name, R.fragment, A.fragment)<=
radius
```

Figure 8 Homology search

5. Materializing Sequenceviews

Views may be materialized at query time or materialized and maintained as additional database tables or database indexes [6]. The advantages of each method are measured as trade-offs in time and space relative to a workload. In the application of *sequenceview* to biological databases the workload strongly suggests that a *sequenceview* be materialized and maintained as an index.

The data in biological sequence databases is almost always write-once and monotonically increasing. Such databases embody a growing knowledge of biological molecules. Once identified, they become a permanent part of the corpus. If mistakes are found, they are usually corrected through versioning. The culture of biology does not permit erasing entries in laboratory notebooks, and these databases are their modern electronic equivalent.

Under this workload it is safe to exclude the cost of maintaining a view in the face of updates and deletes to the base relations. Under some simple models of q-gram similarity it may work out that q-grams could be materialized and compared at query time. The push in genomic databases today is to remove the linear scan that that would entail. These facts together allow us to move straight to an assessment of the materialization of a *sequenceview* as an index.

5.1 Approaches to Metric Spaces

There are three categories of index algorithms for metric-space index trees: radius-based trees (RBTs), generalized-hyperplane trees (GHTs), and vantage point trees (VPTs). See Chavez et al. for an excellent survey [11].

Radius-based methods were inspired by R-trees [23]. In a radius-based method, a data point c is chosen as a center and a radius r determines a bounding sphere. All points p, such that d(c,p)<r are contained in the sphere. Ciaccia et al.'s M-trees are radius-based. Ciaccia et.al.'s effort stands out as the single investigation of an external metric-space index structure with all of the properties expected of a database index [10, 11]. It is ??? paged mapped and capable of supporting a dynamic series database side effects.

In anticipation of integrating M-trees into MoBIoS, we evaluated M-trees for the indexing of protein q-grams. We were not satisfied with the results and made some improvements [31]. The challenges we noted in M-trees are that bounding spheres may overlap, diminishing the pruning behavior of the search. The performance of the M-tree is sensitive to the initial clustering of the data. We optimized the internal node structure and search mechanism from an RB-tree (RBT) and developed an improved bulk-loading scheme [19, 20].

In a generalized-hyperplane tree, data points are selected to be centers, but radii are not computed. The tree uses the hyperplane between clusters as the pruning criterion for search. Given centers c1 and c2, query object q, and range query radius r, the cluster defined by c1 is entered if d(q, c1) – r < d(q,c2) + r. The difference between an RBT and a GHT is that a GHT has no radius and its clusters do not overlap. Brin describes GNAT trees, which comprise a GHT structure and distance ranges from each center to each cluster [9].

A multi-vantage point tree, (MVP-tree), is built by first selecting a given number of points as the vantage points. The distance range from each vantage point is broken into intervals. The Cartesian product of the intervals forms a set of data partitions. Each data element is allocated to a partition by calculating its distance to each vantage point [3, 4, 8, 50]. The construction is applied recursively to form an index tree. See Bozkaya and Ozsoyoglu for details [3, 4].

5.2 Empirical Comparison of RBT, GHT and MVPT

To obtain better performance, we continue to compare RBTs, GHTs and VPTs. Our implementations of a GHT and VPT are slightly different from the original structures. Originally, both GHTs and VPTs reside in main memory, which makes them unscalable for large datasets. In our implementations, both trees are paged so that their index nodes can be fit in disk pages. Thus, the number of index nodes visited can be a measurement of the amount of I/O, and our goal is to minimize the number of index nodes visited. Moreover, our implementation actually combines a GHT and an RBT together. Specifically, the radii are stored, and both the RBT search rules and hyperplane search rules are used.

In our implementation of a multiple vantage point tree (MVPT), the number of vantage points and the split number of each vantage point are decided by the disk page size. Furthermore, we use a farthest-first-traversal (FFT) [24] algorithm to select vantage points, and the dataset is evenly divided. FFT is a k-center algorithm, which is guaranteed to generate a clustering in which the maximum cluster radius is within a factor of 2 of optimal [24].

The three index structures were tested by running range queries on Yeast protein datasets, which were indexed for global alignment of 5-grams using the mPAM weight matrix. The experiment results are presented in Figure 9. In this figure, for each index tree, we show the relationships between the range search radius and the number of distance calculations and the amount of I/O. From Figure 9 we can see that the RBT has the largest number of distance calculation and the MVPT has the largest number of I/O for large radii. However, the MVPT yields the best performance for small proximity search radii.

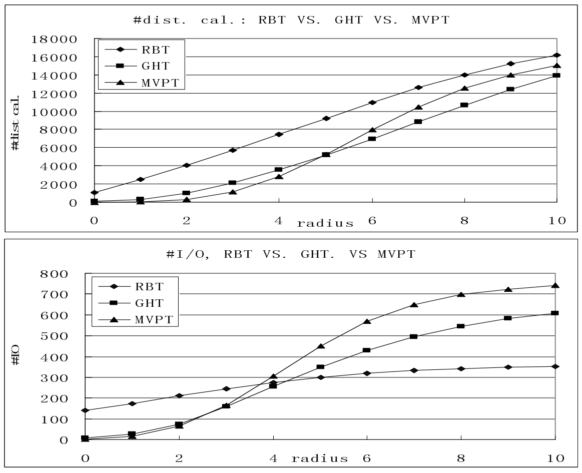

Figure 9 Comparison of metric-space index structures: RBT, GHT, and VPT

5.3 Paged MVP-tree for Materializing Sequenceviews

In our applications, biologically effective results are gained at small search radii. Thus we select MVP-trees as the index structure for protein sequences. We use a fragment size of 5. The size of the alphabet of peptides is 20 (There are 20 different amino acids). Thus, there are 20^5 = 320,000,000 different fragments. For large datasets (millions of amino acids), multiple fragments will have the same contents. Therefore, a search computes many unnecessary distance calculations.

To solve this problem, we bucket the fragments in index leaves. For each leaf index node, fragments with the same content are put into one bucket. Only one distance to each vantage point is stored for all of the fragments in one bucket. When a query is executed, if a bucket cannot be pruned, the query must only compute distance with one fragment in the bucket to decide whether or not all of the fragments in the bucket are valid results. The index structure of the MVPT with bucketing is shown in Figure 10.

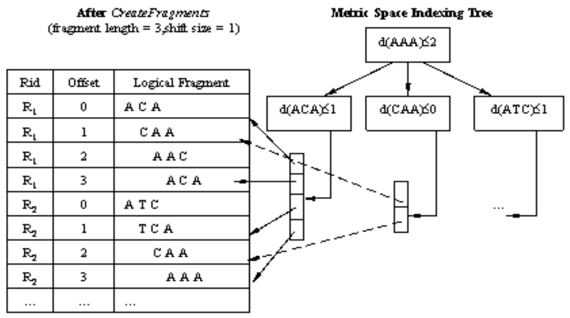

Figure 10 MVPT with bucketing index structure

6. Empirical Results

Although several variations of metric-space indexing have been successfully applied to nucleotide sequences, none of them has shown general applicability to peptides due to a larger alphabet size and the complex relationships among amino acids. However, sequence searches on peptides occur 10% more than on nucleotides, according to a survey conducted by Goble et. al. [17].

The primary challenge of indexing peptides in metric space is to properly define the distance between amino acids. Sellers first proposed this problem in 1974 [40]. In our previous work, we derived an amino acid substitution matrix, mPAM, which satisfies the metric distance properties [44]. Using a metric distance function defined by mPAM for sequence fragments with fixed length, we built an index structure for protein sequences based on the MVP-tree. Unlike other similar work where only exact or near exact matching fragments can be searched, our method searches fragments within a given radius based on global alignment. The trade-off between speed, accuracy and selectivity has been studied and reported in [45].

In this section, we focused on our experimental results conducted on a protein sequence domain using our implementation. The results show that our implementation has scalable, sensitive search performance on biological sequences.

6.1 Methods and dataset

We used two datasets in our studies. To assess sensitivity, we used an accuracy benchmark suite curated and furnished by NCBI. The dataset contains 6433 yeast protein sequences (about 2,892,155 residues). The query set contains 103 sequences whose true positive hits have been identified by human experts and whose curation is continually refined [39]. The benchmark suite was downloaded in August 2002 (ftp.ncbi.nlm.nih.gov/pub/impala/blastest).

For each query sequence s of length k**,** s is divided into a set of q-grams, $\{f_i| \text{ i=1..k-q+1}\}$, referred to as query fragments. We collect all the results from the range query $Q_{SD}(f_i,r)$ for all i and a greedy chaining algorithm is used to compute the final answers. The accuracy is measured using receiver-operating characteristic (ROC) scores, a popular measure used in biology [18]. A similar method was also used for measuring the accuracy of PSI-Blast [39]. For each query, the ROC_{50} value is computed by comparing the result list with the list of true positive hits. The ROC_{50} value has been computed as follows:

$$ROC_n = \frac{1}{nT} \sum_{i=1}^{n} t_i \qquad (1)$$

where t_i is the number of true positive hits ranked ahead of the ith false positive, and T is the total number of true positives.

The data used for the scalability study was downloaded from Genbank in July 2003 (ftp://ftp.ncbi.nih.gov/genbank/genpept.fsa.z). The dataset contains FASTA formatted amino acid translations extracted from GenBank/EMBL/DDBJ records that are annotated with one or more CDS features. A set of databases was built with different subsets of the data that were taken sequentially from the full dataset. The same set of queries from the yeast benchmark was used for all of the databases.

6.2 MVP Tree Parameter selection

There are three parameters associated with our MVP tree implementation: the number of vantage points in each node, the number of children of each vantage point and the maximum number of data points in each node. Figures 11 and 12 show the experimental results for various parameter combinations. There is always a trade-off between the number of leaves visited and the number of distance calculations needed. The more data points a node has, the less leaf nodes need to be visited. However, a tree with bigger nodes requires more distance calculations on each search. Based on this data, we decided to use two vantage points per node, two children per vantage point and a maximum number of 100 data points per leaf node.

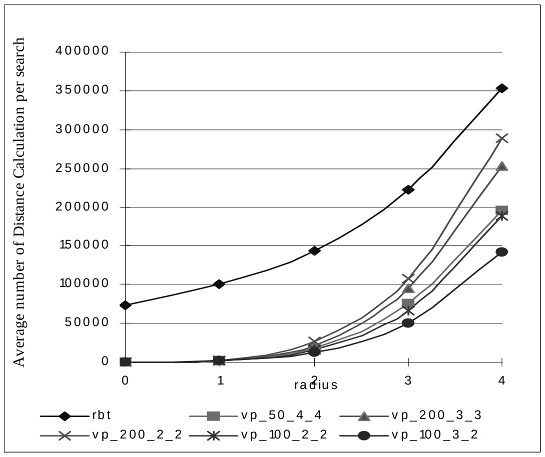

Figure 11 Average number of distance calculations per query for various tree structures

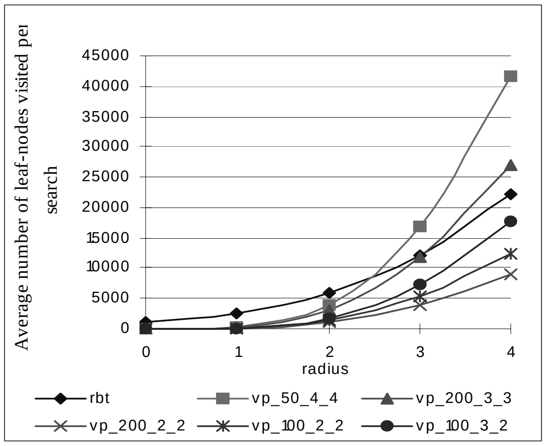

Figure 12 Average number of leaf nodes visited per query for various tree structures

6.3 Quality of Search Result

Table 3 compares the average ROC_{50} score for each query from our algorithm and the results using other searching algorithms with the same benchmark. In the AutoRadiusSearch, the search radii are automatically adjusted based on the prediction of the number of matching q-grams. We show that, using metric space indexing, protein sequence homology search could yield accuracy comparable to BLASTp on the same benchmark. Note that the accuracy of results is actually affected by many factors, such as the value of the substitution matrices, searching strategies and chaining strategies, etc [2, 39].

Table 3 Comparison of average of ROC$_{50}$ value for various searches

Search Method		Matrix	Average ROC$_{50}$
Sequential Search with Smith-Waterman local alignment algorithm		mPAM	0.48
		PAM250	0.59
		PAM70	0.5
Indexed Search	Radius 3	mPAM	0.45
	Radius 4		0.53
	AutoRadiusSearch		0.5
BLASTP		PAM250	0.53
		PAM70	0.42

6.4 Scalability

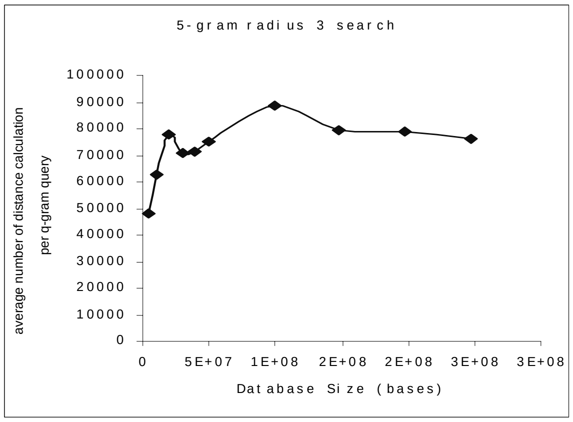

Figure 13a Average number of distance calculations per q-gram search for different size datasets

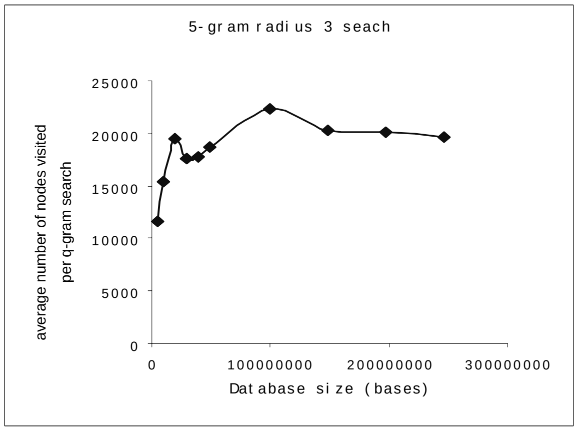

Figure 13b Average number of leaf nodes visited per q-gram search for different size datasets

To evaluate the scalability of our algorithm, we use the same query set as used in the accuracy benchmark against various sizes of databases using AutoRadiusSearch. The average number of distance calculations and the average number of leaf nodes visited are plotted in Figure 13. Both figures reveal scalability with the size of the database. It's also interesting to note that both numbers slightly decreased for a larger dataset.

We have reason to believe that as the database grows the logical locality of the clusters starts to correspond better to the physical clustering on pages [31]. The affect is that entire contents of sub-trees could be found and returned in their entirety without further distance calculations, thus reducing the number of distance calculations. Similarly, entire sub-trees can be pruned reducing search cost.

6.5 Bucketing Duplicated Entries

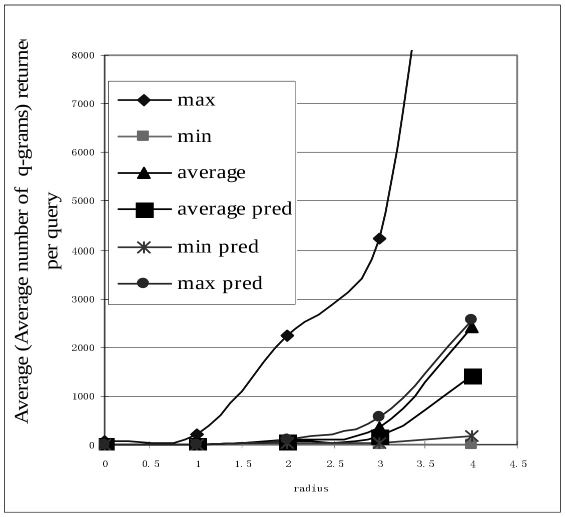

Figure 14 Duplicated entries in the benchmark dataset.

Based on the experimental results, we decided to use q-grams of length 5 for peptide indexing [45]. Since the alphabet size of a peptide sequence is 20, the complete space of q-grams will quickly be covered as the number of q-grams increases. Furthermore, some q-grams have a significantly higher repetition rate than others because of the non-uniform distribution of amino acids.

Figure 14 shows the comparison between actual and predicted results based on a uniform distribution of the average number of q-grams per query for varying radii. To avoid repetitive computation and redundant storage, we used a bucket-like data structure to store a list of different locations where the q-gram occurred more than once. Figure 15 shows that the bucketing structure can decrease the average number of distance calculations even for the benchmark dataset, which barely covers the space of q-grams with length 5. Such a data structure will be more effective and assure scalable search performance for a larger dataset.

7. Conclusion and Future Research

We have spoken specifically of language and physical

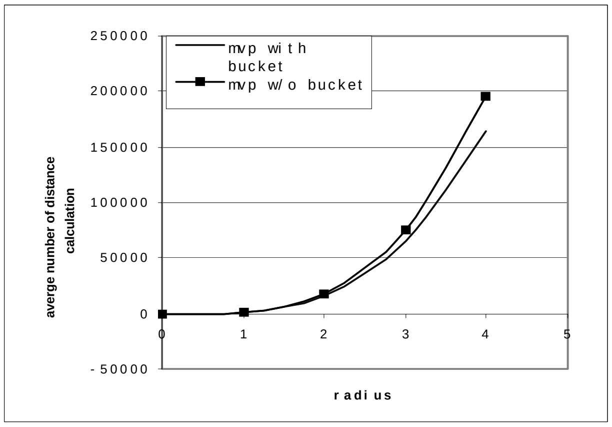

Figure 15 Average number of distance calculations using a bucketing structure with MVP-trees

structures that will enable database management systems to directly support biological sequence analysis. The approach facilitates both homology searches and comparative genomic analysis. We have shown the scalability of MVP-trees on protein q-grams. However, many open problems remain. Our current implementation of a metric-space join is composed of indexed nested loops. When n is in the range of 10^7-10^{10}, even O(n log n) algorithms become computationally challenging. Given a tree-structured access path one can anticipate merge-join like algorithms that would tend toward O(n) execution time, (assuming output size is small). In the case of a self-join a simple recursive descent of the index will work for most, if not all, tree-based methods of metric-space indexing. We anticipate that comparative genomics problems similar to the one we computed in 8 processor days will take a few hours to complete, including building the indexes. Solutions to the more general problem are much more difficult and cannot be diverged from the indexing method.

Also in question is the broader applicability of the language and physical structures. Q-gram approaches are endemic to information retrieval [48]. Q-gram methods first derived for speech recognition are now being extended toward the retrieval of music files by humming [15]. It is plausible that *sequenceviews* and their supporting structures could facilitate these and more traditional sequence applications.

8. References

[1] Alex C.W. May. *Towards more meaningful hierarchical classification of amino acid scoring matrices*. Protein Engieering, 12(9): 707-712, 1999.

[2] Altschul, S.F., and Gish, W. *Local alignment statistics*. Methods Enzymol.266: 460-480, 1996.

[3] Altschul, S.F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. *Basic local alignment search tool*. J. Mol. Biol. 215: 403-410, 1990.

[4] Altschul, S.F., Bundschuh, R., Olsen, R., and Hwa, T. *The estimation of statistical parameters for local alignment score distributions*. Nucleic Acids Res., 29: 351-361, 2001.

[5] Bellman, R. *Adaptive Control Processes: A Guided Tour*. Princeton University Press. U.S. 1961.

[6] Blakeley J. A., Larson P. A., and Tompa F. W. Efficiently Updating Materialized Views. In Proc. ACM SIGMOD, 61-71, Washington D.C., June 1986.

[7] Bozkaya T., and Ozsoyoglu M. *Distance-based indexing for high-dimensional metric spaces*. In Proc. ACM SIGMOD International Conference on Management of Data (1997) 357-368, 1997.

[8] Bozkaya, T., and Ozsoyoglu, M. *Indexing Large Metric Spaces for Similarity Search Queries*. Association for Computing Machinery Transactions on Database System, 11-34, 1999.

[9] Brin, S. *Near neighbor search in large metric spaces*. In Proc. 21st Conference on Very Large Database (VLDB'95), 574-584, 1995.

[10] Califano, A., and Rigoutsos. I. *FLASH: A fast look-up algorithm for string homology*. In International Conference on Intelligent Systems for Molecular Biology, 56-64, 1993.

[11] Chavez, E., Navarro, G., Baeza-Yates, R., and Marroquin, J.L. *Searching in metric spaces*. ACM Computing Surveys. 33(3): 273-321, 2001

[12] Chen, W., and Aberer, K. *Efficient Querying on Genomic Databases*. In Proc. of 8th Int. Work on Database and Expert System Applications, 1997.

[13] Ciaccia, P., Patella, M., and Zezula, P. *M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces*. Proc. VLDB, 1997.

[14] Dayhoff M.O., Schwartz R., and Orcutt B.C. *Atlas of Protein Sequence and Structure*. Vol. 5. Suppl. 3: 345-358, 1978

[15] Ghias, A., J. Logan, D. Chamberlin, and B. C. Smith. *Query by humming - musical information retrieval in an audio database*. In ACM Multimedia 95, 1995.

[16] Giladi, E., Walker, G. M., Wang, J.Z., and Volkmuth, W. *SST: an algorithm for finding near-exact sequence matches in time proportional to the logarithm of the database size*. Bioinformatics. 18(6): 873-879, 2002.

[17] Goble, R.S.C., Baker, P., and Brass, A. *A classification of tasks in bioinformatics*. Bioinformatics, 17: 180-188, 2001.

[18] Gribskov, M., and Robinson, N. L. *Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching*. Computers and Chemistry. 20(1): 25-33, 1996.

[19] Grossman, D. A., and Frieder, O. *Information Retrieval*. Kluwer Academic Publishers, 1998.

[20] Gravano, L., Panagiotis Ipeirotis, P.G., Jagadish, H.V., Koudas, H. ,Muthukrishnan, S., and Srivastava, D.

Approximate String Joins in a Database (Almost) for Free. VLDB 491-500, 2001

[21] Gravano, L., Panagiotis Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S. , Pouri, L, and Srivastava, D. *Using q-grams in a DBMS for Approximate String Processing*. IEEE Data Engineering Bulletin 24(4): 28-34, 2001.

[22] Gusfield, D. *Algorithms on Strings, Trees and Sequences Computer Science and Computational Biology*. Press Syndicate of the University of Cambridge, USA, 449-454, 1997.

[23] Guttman. A. *R-trees: A Dynamic Index Structure for Spatial Searching*. Proc. of SIGMOD, 1984.

[24] Hochbaum, D. S., and Shmoys, D. B. *A best possible heuristic for the k-center problem*. Mathematics of Operational Research, 10(2):180-184, 1985.

[25] Hunt, E., Atkinson, M.P., and Irving. R.W. *A database index to large biological sequences*. In VLDB, 139-148, Roma, Italy, September 2001.

[26] Jaeschke, G., Schek, H. J. *Remarks on the algebra of non first normal form relations*. Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems, March 29-31, 1982, Los Angeles, California.

[27] Johnson, M. S., and John, P. *Overington A Structural Basis for Sequence Comparisons: An Evaluation of Scoring Methodologies*. J. Mol. Biol. 233: 716-738, 1993.

[28] Kent, W. J. *BLAT-The BLAST like alignment tool*, 2002.

[29] Lenhard, B., Sandelin A., Mendoza1 L., PEngstrm1, Jareborg N., and Wasserman W. W. *Identification of conserved regulatory elements by comparative genome analysis*. Journal of Biology 2(13), 2003.

[30] Meek, C., Patel, J.M., and Shruti Kasetty, S. *OASIS: An Online and Accurate Technique for Local-alignment Searches on Biological Sequences*. VLDB 2003: 910-921Genome Res. 12: 656-664, 2003.

[31] Mao, R., Xu, W., Singh, N. & Miranker, D. P. *An Assessment of a Metric Space Database Index to Support Sequence Homology*. In the proceeding of the 3rd IEEE Symposium on Bioinformatics and Bioengineering, March 10-12, 2003, Washington D.C

[32] Marcotte, E.M., and Date, S.V. *Exploiting Big Biology: Integrating Large-scale Biological Data for Function Inference*. Briefings in Bioinformatics 2(4): 363-374, 2001.

[33] Miranker, D. P. Xu, W. & Mao, R. *Architecture and Application of MoBIoS, a Metric-Space DBMS to Support Biological Discovery*. 15th International Conference on Scientific and Statistical Database Management. (SSDBM03) 241-244, 2003.

[34] Myers, E.W. *A sublinear algorithm for approximate keyword searching*. Algorithmica. 12(4/5): 345-374, 1994.

[35] Pearson, W. R., and Lipman, D. J. *Improved tools for biological sequence comparison*. Proc. Natl Acad. Sci. USA, 85: 2444-2448, 1988.

[36] Pevzner P.A., Mulyukov Z, Dancik V, and Tang CL. *Efficiency of database search for identification of mutated and modified proteins via mass spectrometry*. Genome Res. 11(2): 290-9, 2001.

[37] Rouchka, E. C. Gish, W., and States D. J. *Comparison of whole genome assemblies of the human genome*. Nucleic Acids Res., 30(22): 5004 – 5014, 2002.

[38] Sahinalp, C., Macker, S.J., Tasan, M., and Ozsovoglu, M. *Distance Based Indexing for String Proximity Search*, to appear ICDE 2003.

[39] Schaffer, A. A. Aravin, L. Madden, T. L., Shavirin, S., Spouge, J. L., Wolf, Y. I., Koonin, E. V., and Altschul, S.F. *Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements*. Nucleic Acids Res., 29(14): 2994-3005, 2001.

[40] Sellers, P.H. *On the theory and computation of evolutionary distances*. J. Appl. Math. (SIAM). 26: 787-793, 1974.

[41] Stuart J. M., Segal E., Koller D., and Kim S. K. *A Gene Coexpression Network for Global Discovery of Conserved Genetic Modules*. Science 302: 249-55, 2003.

[42] Tan, Z., Cao, X., Ooi, B.C., and Tung, A.K.H. *The ed-tree: an index for large DNA sequence databases* In Proc. 15th International Conference on Scientific and Statistical Database Management (SSDBM 2003) 151-160, 2003.

[43] Tata, S., and Patel, J. *PiQA: An Algebra for Querying Protein Data Sets* 15th International Conference on Scientific and Statistical Database Management. (SSDBM03) 141-151, 2003.

[44] Xu, W., and Miranker, D.P. *A metric model for amino acid substitution*, Bioinformatics, 2004. (http://www.cs.utexas.edu/users/mobios/)

[45] Xu, W., Miranker, D. P., Mao, R., and Wang, S. *Indexing Protein Sequences in Metric Space*. TR. (http://www.cs.utexas.edu/users/mobios)

[46] Xu, W., Briggs, W. J, Padolina, J., Liu, W., Linder, C. R., and Miranker, D. P. *Using MoBIoS' Scalable Genome Joins to Find Conserved Primer Pair Candidates Between Two Genomes.* in press ISMB, 2004.

[47] Wang, T.L., and Shasha, D. *Query processing for distance metrics*. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, Proceedings of the 16th International Conference on Very Large Databases, 602-613, Brisbane, Australia, August 1990.

[48] Williams, H. E., and Zobel,.J. *Indexing and Retrieval for Genomic Databases*. In IEEE Transactions on Knowledge and Data Engineering, 14(1): 63-78, 2002.

[49] Wu, S., Manber, U., Myers, G., and Miller, W. *An O(NP) Sequence Comparison Algorithm*. Information Processing Letters, 35(6): 317-323, 1990.

[50] Yianilos, P. *Data structures and algorithms for nearest neighbor search in general metric spaces*. In Proc. 4th ACM-SIAM. Symposium on Discrete Algorithms (SODA'93) 3s11-321, 1993.