Bulletin of the Technical Committee on

Data Engineering

September 2004 Vol. 27 No. 3

IEEE Computer Society

Letters

Letter from the Editor-in-Chief . . . *David Lomet* 1
Letter from the Special Issue Editor . . . *Jignesh M. Patel* 2

Special Issue on Querying Biological Sequences

Biosequence Use Cases in MoBIoS SQL . . . *Daniel P. Miranker, Willard J. Briggs, Rui Mao, Shulin Ni, and Weijia Xu* 3
Querying BLAST within a Data Federation . . . *Barbara A. Eckman and Arthur Kaufmann* 12
ODM BLAST: Sequence Homology Search in the RDBMS . . . *Susie Stephens, Jake Y. Chen, and Shiby Thomas* 20
Indexed Searching on Proteins Using a Suffix Sequoia . . . *Ela Hunt* 24
Progressive Searching of Biological Sequences . . . *Tamer Kahveci and Ambuj Singh* 32
Novel Approaches to Biomolecular Sequence Indexing . . . *Emre Karakoc, Z. Meral Ozsoyoglu, S. Cenk Sahinalp, Murat Tasan, and Xiang Zhang* 40

Conference and Journal Notices

ICDE Conference . . . back cover

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

There are two Data Engineering Bulletin web sites: http://www.research.microsoft.com/research/db/debull and http://sites.computer.org/debull/.
The TC on Data Engineering web page is
http://www.ccs.neu.edu/groups/IEEE/tcde/index.html.

Letter from the Editor-in-Chief

International Conference on Data Engineering (ICDE'05)

I want to draw your attention to the "call for participation" for ICDE'05 that is on the back cover of this issue. ICDE (also called simply the Data Engineering Conference) is sponsored by the IEEE Technical Committee on Data Engineering (TCDE). It is the the flagship conference of the IEEE in the area of database technology. The next Data Engineering Conference is in Tokyo in April, timed to coincide with the cherry blossums. The conference is very selective, ensuring a fine technical program. I encourage you to find out more about the conference at its web site (http://icde2005.is.tsukuba.ac.jp/).

Large technical conferences such as ICDE do not happen by magic. There is always a local committee that does an enormous amount of work so that the conference can run smoothly. There is also a standing committee within the IEEE, called the ICDE Steering Committee, that oversees the process, selects among conference proposals, checks the budgetting, etc. Erich Neuhold is the chair of the ICDE Steering Committee, and can be reached via email at neuhold@ipsi.fhg.de. The Steering Committee is always interested in hearing proposals for conferences in future years.

The Current Issue

Our world is rapidly being transformed by the advances of biotechnology. These advances are the result of the fundamental understanding achived over the past 50 years or so in the area of genetics and DNA sequences. Much of the work involved in this enterprise is pure biology, involving elaborate laboratories and careful biological experimentation.

However, advances in our understanding of the biology of the genes depends in an essential way on computer data processing of sequence information. Indeed, it is the ongoing automation of much of this, exploiting indexing technology and, increasingly, databases as well, that enables the rapid strides that we have grown accustomed to over the past several years. The clear relevance of databases to the genetic sequencing enterprise has triggered substantial work in the database community to respond to this challenge.

The current issue reports on but a sampling of the work going on in our field to respond to the challenge of dealing with genetic sequence data. Progress in making databases deal well with biology is changing the way that biologists do their science. The papers in this issue come from a mix of commercial and academic researchers from widely distributed institutions. This testifies to the great interest that this area has generated. I want to thank Jignesh Patel for the fine job he has done in assembling this issue, which could not be more timely. In a very real sense, the moment for our community to have impact is now. So I would urge you to study this issue of the Bulletin, and consider contributing to the revolution in biology.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

The current and ongoing revolution in life sciences research has lead to fantastic achievements such as the sequencing of entire genomes of various organisms. Hidden in these vast sequences of nucleic acids are the codes that govern the behavior of the cellular machinery, and clues to how modern organisms have evolved. Sequences are ubiquitous in life sciences applications, and this issue of data engineering highlights some of the work on querying biological sequences that is ongoing in the database community.

The first article by Miranker, Briggs, Mao, Ni and Xu presents a framework, based on SQL extensions, that can be used to pose a variety of complex queries on biological sequences. Current methods for posing such queries are largely procedural, and this work highlights the benefits that declarative querying can bring to the life sciences community.

The tremendous potential of extending the SQL framework to allow querying on biological sequences has also been noticed by commercial relational database vendors. BLAST is the most common tool for querying biological sequences, and the next two articles outline how IBM and Oracle have integrated BLAST querying into their relational frameworks. Eckman and Kaufmann present the approach taken by IBM DB2 Information Integrator, and Stephens, Chen, and Thomas present the approach taken by Oracle Database 10*g*.

Since approximate matching of a query sequence is perhaps the most common query in life sciences, it is natural (especially to database researchers) to look for index-based methods for evaluating this operation. Surprisingly, existing tools for evaluating biological sequence matching often don't use indices. The last three articles in this issue present various index-based methods for sequence matching.

The article by Hunt presents a novel index, called the suffix sequoia, which can be used to dramatically reduce the cost of executing a common fully-sensitive sequence search algorithm. The article by Kahveci and Singh describes how a multi-dimensional index can be used for *progressive* sequence searching, providing a better user paradigm than current blocking methods. The final article by Karakoc, Ozsoyoglu, Sahinalp, Tasan, and Zhang makes the case that distance-based indexing methods can be applied for sequence searching, but suffer from the curse of dimensionality, and in the worst case are comparable to brute-force methods. This final article also sketches how certain similarity metrics be approximated by Hamming distance, which are more amenable to indexing.

The mysteries of life are hidden in various types of data that are used in life sciences applications. The sequence data type, which is the focus of this issue, is just one of the many complex data types that are used in life sciences applications. Life science researchers around the world are working diligently on cracking the hidden codes in these vast, and rapidly growing, biological data sets. Many of the queries that these scientists want to pose require functionality that is well beyond the scope of traditional relational database engines. Database researchers and vendors have a lot to contribute to this field, and hopefully this special issue inspires more interest in the database community to explore and contribute to this exciting area.

Jignesh M. Patel
University of Michigan
Ann Arbor, MI

Biosequence Use Cases in MoBIoS SQL

Daniel P. Miranker, Willard J Briggs, Rui Mao, Shulin Ni and Weijia Xu
Department of Computer Sciences
University of Texas at Austin
{miranker, willard, rmao, shulin, xwj}@cs.utexas.edu *

Abstract

The sequencing and annotation of entire genomes has enriched the content of biological sequence databases such that new methods of sequence analysis, comparison and retrieval are being invented and rerun on an increasingly regular basis, generating new and more complete biological information. Examples include full genome comparisons and phylogenetic footprinting. Simple identification of homologous sequences based on BLAST searches is now just one option for querying the contents of a sequence database.

These developments underscore the need for more general methods of sequence data management and concomitant programming models that simplify biological discovery. MoBIoS, the Molecular Biological Information System, with mSQL, its set of SQL extensions, is such a system. MoBIoS supports two views of sequence data. Sequences are identified and stored based on long functional units (e.g. genes, proteins and chromosomes). Matching and analysis of sequences exploits distance-based methods comparing short-overlapping substrings. We show that a number of sequence analysis problems can thus be succinctly expressed as mSQL queries.

1 Introduction

MoBIoS, the Molecular Biological Information System (pronounced mobius), is a metric-space database management system targeting life-science data. Analogous to spatial databases which extend relational systems with index-structures and data types that support two and three-dimensional data and form the basis of geographic information systems (GISs), MoBIoS comprises built-in biological data types and index structures to support fast object storage retrieval based on the relative distance between objects determined by metric-distance functions (metrics) [MXM03, CNBYM01]. Figure 1 illustrates the MoBIoS platform. MoBIoS is built in Java on top of the McKoi open-source DBMS[McK]. McKoi includes a JDBC interface, allowing MoBIoS to integrate seamlessly with web-application tool stacks.

Work to date includes development or identification of effective metric-models of biological similarity for peptide sequences (mPAM), mass-spectrometer signatures and 3-d electrostatic models of proteins and respective applications [MXM03, XM04, ZBB].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This research was supported by grants from the Texas Higher Education Coordinating Board and the National Science Foundation contract DBI-0241180, IIS-0325116, EIA-0121680, EF-0331453

In the case of sequences, more database machinery is needed than with atomic representations of mass-spectrometer signatures or 3-D protein models. In the latter two cases distance-based range queries and join queries are precisely analogous to spatial databases, except that absolute position in Euclidean space is replaced with relative distance determined by a metric. This is not to say that the community's understanding of query processing and index support of metric-space databases is at all mature. Competing indexing methods continue to be refined [Bri95, MXSM03, STMO03]. There is still no clear winner. Metric-space joins have barely been addressed [WS90, DGZ03].

A management challenge for biological sequences is that a biologist's view of a sequence is different than the computational view. Identification of biological sequences comprises long functional units (e.g. genes, proteins and chromosomes). Excluding the Smith-Waterman algorithm as an important exception, most comparative sequence analysis algorithms are structured such that they first break sequences into short overlapping substrings. Further processing compares substrings and may ultimately reassemble them into longer units. Thus far these algorithms rarely consider additional substructure; for example, the location of introns, exons, and transcription factor binding sites. These functional subunits are enumerated by name and position in the Genbank features table. As a group, these are referred to as sequence annotations.

Our speculation is that the granularity of sequence management in Genbank and related systems is largely responsible for the disassociation of annotation from sequence comparison. In common practice, a set of sequences is retrieved from Genbank by virtue of common annotations and/or BLAST-based similarity. The set of sequences are culled by further analysis of sequence content. Additional inspection or filtering of those sequences based on annotations requires ad-hoc scripts to map the resulting sequences back to their Genbank entries. Thus, we claim that there is ample motivation to integrate sequence analysis with database query engines and enable optimized query plans to interleave primary structure (sequence) and functional comparisons.

In addition to metric-distance based access paths, MoBIoS includes syntactic, logical and physical database extensions to manage biological sequences. The primary syntactic extension is called a *sequenceview*. *Sequenceviews* enable SQL programmers to specify that, in addition to storing and retrieving biological sequences as long functional units, a sequence may also be operated upon as a set of overlapping q-grams. Furthermore, users may specify one of a number of built-in metrics for comparing the similarity of q-grams, or they may specify their own in a manner similar to writing a stored procedure. Three logical operators make *sequenceviews* possible, *createfragments()*, *groupfragments()* and *merge()*. The corresponding physical operators and supporting structures are discussed elsewhere [BLM$^+$03].

In this paper we discuss our query language and illustrate its use to capture a number of sequence analysis protocols emerging in bioinformatics. The data model specified in Figure 2 will serve as the basis for each of the examples.

The first two tables are used to store DNA and protein sequences, where the column 'Seq' holds the sequences. The sequence id, SID, serves as a foreign key to a table of sequence annotations. Per the vernacular of the area this is called the feature table. Rooted in ASN.1, the semi-structured foundation of Genbank, a feature is a substring of a sequence, denoted by the offset of the first and last sequence element and labeled with one of a moderated list of tags [BKML$^+$02].

2 mSQL

mSQL is the name we have chosen for our data type and operator extensions to the SQL standard. mSQL introduces data types to manage sequence data, mass spectral data, and secondary and tertiary protein data. The primitive data types introduced to handle sequence data are called DNA, RNA, and Peptide, all of which are subtypes of a generic Sequence data type. In general, a Sequence can be thought of as a string with two important differences, stemming from the biological nature of sequences. First, the alphabet is limited to a certain set of characters depending upon the type of sequence, i.e. ACTG for DNA sequences. Second, we must

```
DNA_Sequence(SID, Organism, seq)
Protein_Sequence(SID, Organism, seq)
Feature_table(SID, FID, tag, start, end)
```

Figure 2: Tables and Attributes of the Example Schema

```
Select SID, Organism, Createfragments(seq, 3)
From DNA_Sequence;
```

Figure 3: *Createfragments()* Query

```
Select *
From Createfragments(DNA_Sequence.seq, 3);
```

Figure 4: *Createfragments()* Query

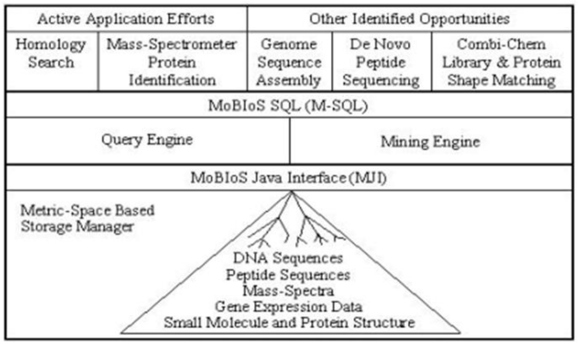

Figure 1: Architecture of the MoBIoS Platform

introduce the *revcomp()* operator to compute the reverse complement of DNA and RNA sequences; applied to ordinary strings, *revcomp()* would have no logical meaning.

mSQL also introduces two new SQL-level operators to convert sequence information between its two logical perspectives: *createfragments()* and *merge()*. These operators are rather similar to the unnest and nest operators popular in the extended-relational algebras of the early to mid-80s [JS82]. Their differences lie in the pre-processing and post-processing steps necessary for each to logically view sequences as sets of overlapping subsequences.

Createfragments() is a two-step operation. The first step takes a sequence of characters and a fragment length as input and returns a set of 2-tuples. The first attribute of each tuple is the offset from the original sequence, and the second attribute is the fragment. Each of these 2-tuples is an instance of an additional internal data type, SubSequence. SubSequence contains these two fields, offset and fragment, as well as an operator to obtain the length of the sequence. The vast majority of operations on sequence data will be performed using these SubSequences. After the first step of *createfragments()*, the set is unnested to yield the final usable result.

For example, assume that the DNA_Sequence table described above is populated with the following two rows: {R1, Rice, ACAA}, {R2, Rice, ACTCA}. The query in Figure 3 would yield the result shown in Table 1. This set is then unnested, yielding the final result in Table 2

Table 1: Intermediate Results of *Createfragments()* Operation

SID	Organism	Createfragments(Sequence, 3)
R1	Rice	[[0, ACA], [1, CAA]]
R2	Rice	[[0, ACT], [1, CTC], [2, TCA]]

Table 2: Final Results of *Createfragments()* Operation

SID	Organism	Createfragments(Sequence, 3)
R1	Rice	[0, ACA]
R1	Rice	[1, CAA]
R2	Rice	[0, ACT]
R2	Rice	[1, CTC]
R2	Rice	[2, TCA]

Note that these fragments are not guaranteed to be in sequential order. In the implementation of mSQL, the two steps of *createfragments()* have been combined into one, with the syntax demonstrated in Figure 4.

This yields the same results as shown in Table 2.

The *merge()* operation is nearly the reverse of *createfragments()*. An additional step is also needed at the beginning, due to the fact that the results of a query may not necessarily be in sequential order. For this step, we have overloaded the standard SQL GROUP BY operator to order these fragments by their offsets and then separate them into groups. Two fragments are considered to be in the same group if the difference between their offsets is less than the length of the fragments. The second step is the nest operation, which is applied individually to each of these groups. The final step is the actual merge, which merges each set of tuples back into one larger sequence. The sequences are ordered by offset and then the overlapping sections are removed, yielding one long SubSequence. The offset from the first fragment of the set is maintained as the offset from the original sequence for the entire subsequence.

The *merge()* operation can occur in either a one-dimensional or a two-dimensional case. We have just described the one-dimensional case, which assumes that all of the fragments are from the same parent sequence. The two-dimensional case is used on the results of a metric join. In this case the results are first grouped by the fragments from the first sequence, then by the fragments from the second, with the additional rule that two fragments must be from the same parent sequence to be in the same group. The nest and the merge are performed as usual.

It is not feasible or necessary to materialize the results of the *createfragments()* operation. With a fragment length of q and a sequence length of n, materializing *createfragments()* would require storing an additional q(n-q+1)-n characters. For this we introduce the concept of the *sequenceview*, analogous to SQL's view, which materializes the results of *createfragments()* as a secondary metric space index. Implementation details are discussed elsewhere [BLM$^+$03]. *Sequenceviews* can be used in the same manner as standard SQL views, without the same space or time overhead. In this way indices can be pre-built offline, speeding online queries.

3 Application Examples

3.1 Electronic PCR

A sequence-tagged site consists of a pair of primers which can uniquely identify a site in the genome. Electronic PCR is used to computationally find sequence-tagged sites (STSs) in DNA sequences by searching for subsequences that closely match the PCR primers and have the correct order, orientation, and spacing that they could plausibly prime the amplification of a PCR product of the correct molecular weight [Sch97].

In-lieu of a procedural utility program, the Electronic PCR problem can be solved as an mSQL query (Figure 5). For brevity, we introduce some simplifications, i.e., we have not checked for the possibility of matching reverse complements. We have coded the problem as described below.

- Create *sequenceviews* for forward and reverse primers in a STS table; create a *sequenceview* for the genome of an organism. *(lines 1-3; lines 4-6; lines 5-9)*
- Utilize the metric-space index to find matching fragments of primers and genome sequences. Find pairs of merged fragments that match forward and reverse primers with the following conditions:
  - The primers are fully matched. *(lines 15-16; lines 20-21)*
  - The two genome fragments come from the same sequence. The two primers belong to the same STS. *(lines 24-26; lines 27-28)*
  - The spacing between the two genome fragments is within 50 bases of the length of the PCR product. *(lines 29-30)*

3.2 Conserved Primer Pair Discovery

To help solve the question as to whether evolution is adequately modeled by bifurcating trees, or if/when network models are critical, we used MoBIoS to determine a candidate set of PCR primers that would enable biologists to sample, amplify and sequence the DNA of any flowering plant in a large number of places. The query, which we hand-compiled, involves joining *sequenceviews* of the Rice and Arabidopsis genomes in search of shared substrings that fulfill the electronic PCR properties. The number of nucleotides, and therefore the number of logical rows, is in excess of half a billion. Our current implementation comprises an indexed nested-loop join,

```
CREATE SEQUENCEVIEW Forward_view AS
SELECT *
FROM CREATEFRAGMENTS(STS.ForwardPrimer, 5)
CREATE SEQUENCEVIEW Reverse_view AS
SELECT *
FROM CREATEFRAGMENTS(STS.ReversePrimer, 5)
CREATE SEQUENCEVIEW Org_view AS
SELECT *
FROM CREATEFRAGMENTS(DNA_sequence.seq, 5)
WHERE DNA_sequence.orgnism = 'Org'
SELECT Gseq1.offset, Gseq2.offset, STS.sid
FROM
(SELECT merge(G.seq AS Gseq1, F.seq AS Fseq)
FROM Org_view G, Forward_view F
WHERE distance('base_pair_mismatch',
    G.seq.fragment, F.seq.fragment) <= 0.0
GROUP BY G.seq, F.seq),
(SELECT merge(G.seq AS Gseq2, R.seq AS Rseq)
FROM Org_view G, Reverse_view R
WHERE distance('complement_base_pair_mismatch',
    G.seq.fragment, R.seq.fragment) <= 0.0
GROUP BY G.seq, R.seq),
STS
WHERE Fseq.sid = STS.sid
AND Rseq.sid = STS.sid
AND Gseq1.sid = Gseq2.sid
AND Fseq.length = STS.ForwardPrimer.length
AND Rseq.length = STS.ReversePrimer.length
AND abs(G2.seq.offset – G1.seq.offset
    – G1.seq.length – STS.length) < 50
```

Figure 5: Electronic PCR

which is $O(nlogn)$. We solved the problem in less than 2 days using 4 processors of a Sun 6800. The results are currently being validated in a wet lab [XBP+04]. This type of computation is usually the province of very-large clusters, running parallel copies of BLAST as the inner loop of an $O(n^2)$ solution. Please see Xu et al for the mSQL code for this query [XBP+04].

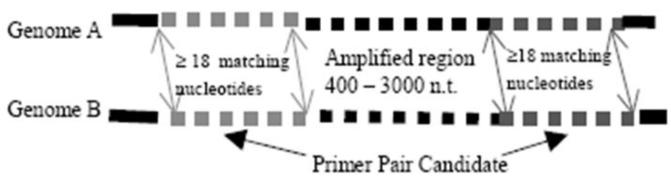

Figure 6: Finding Conserved Primer Pairs

```
1. SELECT merge(R.fragment, A.fragment, g, d)
2. FROM S_view R, q_sview A
3. WHERE distance(metric_name, R.fragment,
A.fragment) <= radius
4. GROUP BY R.fragment, A.fragment
```

Figure 7: Homology Search

3.3 Homology Search

The mSQL query to solve the homology search problem is illustrated in Figure 7. BLAST-like matching of hot-spots is accomplished by a metric-space join. A two-dimensional merge operator merges the matching q-grams [BLM+03, XMMW03]. An optional gap function, g and distance threshold, d, enables hot-spot extension.

3.4 All-way Genomic Conservation

The availability of whole genome sequence data makes it possible to discover conserved features across multiple organisms. Ultraconserved elements are sequence segments that are absolutely conserved (100% identity with no deletion or insertion) between orthologous regions of a number of genomes. See Figure 8. A recent computational study of the human, rat, and mouse genomes has found that ultraconserved elements play important roles in RNA processing, transcription regulation and development [BPM+04].

Again, for brevity and simplicity, we don't limit fragment matching to orthologous regions in the mSQL query for finding ultraconserved elements from three genomes. We first create *sequenceviews* with fragments of length 200 from each genome. Thus the minimum length of an ultraconserved element is 200. The query is formulated as in Figure 10.

Notice in lines 12 and 14, the join stipulates exact matches per the past work and the ability of the software. However, in MoBIoS we could easily repeat the study for varying amounts of sequence divergence by increasing the join distance.

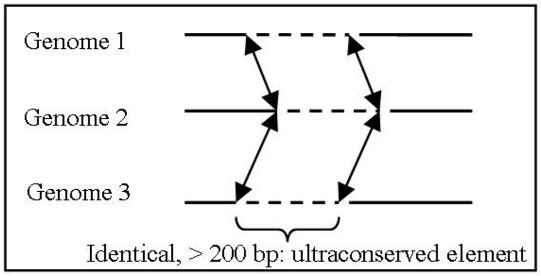

Figure 8: Ultraconserved Element

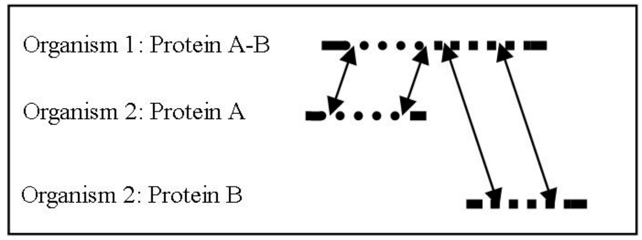

Figure 9: Rosetta Stone Protein Search

```
1. Create sequenceview Human_view as
2. Select *
3. From createfragments(DNA_Sequence.seq, 200)
4. Where Organism = 'Human';
5. Create sequenceview Rat_view as
// Same as Human_view;
6. Create sequenceview Mouse_view as
// Same as Human_view
7. Select merge(merge(Rat_view.seq, Mouse_view.seq),
Human_view.seq))
8. From Human_view,
9.     (Select *
10.    From Rat_view, Mouse_view
11.    Where distance('hamming', Rat_view.seq.fragment,
12.          Mouse_view.seq,fragment) = 0)
13. Where distance('hamming', Human_view.seq.fragment,
14.    Rat_view.seq.fragment) = 0
15. Group By Human_view.seq, Rat_view.seq, Mouse_view.seq;
```

Figure 10: Three-way Genome Comparison

3.5 Rosetta Stone

It has been observed that if two proteins A and B in one organism are both homologous to a single protein A-B in another organism (See Figure 9), there is a good chance that A and B interact or share related biological functions [MPN+99]. Such a protein A-B is termed a 'Rosetta Stone' protein. Sequence alignment methods can be used to find if two proteins in one organism have non-overlapping alignments on a single protein in another organism. The following mSQL query is to obtain the sequences IDs of these protein triplets (Figure 11).

```
1. CREATE SEQUENCEVIEW sview AS
2. SELECT *
3. FROM CREATEFRAGMENTS(Protein_sequence.seq, 5)
4. CREATE VIEW M AS
5. SELECT MERGE(S1.seq AS seq1, S2.seq AS seq2, g, d)
6. FROM sview S1, sview S2, Protein_sequence P1,
7. Protein_squence P2
8. WHERE  distance('mPAM', S1.seq.fragment,
9. S2.seq.fragment) <= radius
10. AND S1.seq.sid = P1.sid AND S2.seq.sid = P2.sid
11. AND P1.organism != P2.organism
12. GROUP BY S1.seq, S2.seq
13. SELECT M1.seq1.sid, M1.seq2.sid, M2.seq2.sid
14. FROM M M1, M M2, Protein_sequence P1,
15.    Protein_sequence P12,  Protein_sequence P22
16. WHERE M1.seq1.sid = M2.seq1.sid
17. AND P1.sid = M1.seq1.sid
18. AND P12.sid = M1.seq2.sid
19. AND P22.sid = M2.seq2.sid
20. AND P12.organism = P22.organism
21. AND M1.seq1.offset + M1.seq1.length < M2.seq1.offset
22. AND P1.seq.length - (M1.seq1.length + M2.seq1.length) <
23.    max_gap_length1
24. AND P12.seq.length - M1.seq2.length < max_gap_length2
25. AND P22.seq.length - M2.seq2.length < max_gap_length2
```

Figure 11: Rosetta Stone Query

The proteins come from multiple organisms. There are three steps in our query.

- First, create a single *sequenceview* for protein sequences of all organisms. *(lines 4-12)*
- Use the metric space index to make local alignments between pairs of proteins from any two distinct organisms. The results are pairs of merged matching protein fragments. Create a view from the results. *(lines 4-12)*

- Produce the Rosetta Stone protein triplets using the following conditions:
  - Two proteins from one organism are aligned to a single protein from another organism without overlap. *(lines 16-20)*
  - The difference between the length of a protein and the total length of its aligned fragments must be less than a given constant. *(lines 21-25)*

```
Create sequenceview miRNA_view as
Select *
From createfragments(miRNA.sequence, 21)

Create sequenceview genome_view as
Select *
From createfragments(genome.sequence, 21)
Where Organism =' orgA';

Select merge (genome_view.seq, miRNA_view.seq)
From miRNA_view,
    genome_view
Where
    Distance ('miRNA_metric', genome_view.seq,
            miRNA_view.seq) <=5
AND
      //Genome_view.seq is not in coding region
      (Select COUNT (*)
      From feature_table
      Where
           Feature_table.sid = Genome_view.seq.sid
           AND
  feature_table.start>=Genome_view.seq.offset +
   Genome_view.seq.length
           AND
           feature_table.stop<=
 Genome_view.seq.offset) =0
```

(a) MiRscan query

```
Create sequenceview miRNA_view as
Select *
From createfragments(miRNA.sequence, 7)
Create sequenceview genome_view as
Select *
From createfragments(genome.sequence, 7)
Where Organism =' orgA';
 Select merge (genome_view.seq, miRNA_view.seq)
 From genome_view, miRNA_view
Where
       ((miRNA_view.seq.offset =2
        AND
         Distance ('RNA_complementary_metric',
                   genome_view.seq, miRNA_view.seq) =0)
       OR
         (miRNA_view.seq.offset >2
   AND
   Distance ('RNA_complementary_metric',
             genome_view.seq, miRNA_view.seq)<=3))
   AND
  (Select COUNT (*)
   From feature_table
   Where
          feature_table.sid = Genome_view.seq.sid
        AND
          feature_table.start>=Genome_view.seq.offset +
                               Genome_view.seq.length
        AND
          feature_table.stop<= Genome_view.seq.offset) =0
```

(b) miRNA target site query

Figure 12: RNAi Queries

3.6 RNA Interference

RNA interference (RNAi) refers to the post-transcriptional gene silencing (PTGS) induced by the direct introduction of double stranded RNA. In the past few years, RNAi has become a popular tool in molecular biology to knock out genes in a variety of organisms [Gur00, HCH01].

MicroRNAs (miRNAs), an important class of interfering RNA, are endogenous RNAs that are about 22 nucleotides long. MiRscan is an miRNA gene prediction tool in which all experimentally verified miRNA genes were compared with a 21nt windows passing through each conserved stem loop of the genome sequence [LGY$^+$03]. TargetScan is a tool that predicts target sites conserved across multiple genomes. The first step of the algorithm is to search a set of orthologous 3' UTR sequences from one organism for perfect Watson-Crick complementary matches to bases 2-8 (from the 5' end) of the miRNA, and then extend matches [LSJR$^+$03]. We expect such searching processes can also be expressed in mSQL as shown in Figure 12.

In both queries, the *sequenceviews* for known miRNA genes and sequences of one genome are created with fragment lengths 21 and 7, respectively *(lines 1-3; lines 4-7)*. To find the miRNA candidate (Figure 12(a)), we use 'miRNA_metric' as the metric distance function to measure the closeness of two miRNA segments *(lines 12-13)*. In Figure 12(b), the 'RNA_complementary_metric' is the metric distance function used between the reverse complement of the first RNA fragment and the second RNA fragment *(lines 11-19)*. The purpose of lines 16-25 in Figure 12(a) and lines 22-31 in Figure 12(b) are to exclude fragments that are derived from the coding region. The results from the above queries are subject to further evaluations such as meeting the energy required for RNA folding.

4 Discussion and Conclusion

MoBIoS, and especially its mSQL component, remains a work in progress. While we have successfully found a solution to the Conserved Primer Pair problem using the MoBIoS platform, none of the above queries have yet been implemented at a SQL level. In presenting them it is our goal to show that the future of genomics research goes far beyond the homology search now possible with programs such as BLAST; that as new, interesting problems arise with greater and greater frequency, biologists need tools that are powerful enough to adapt quickly to changing demands; and finally, that these tools must be easy to use and rely on already established standards. MoBIoS with mSQL promises to address all of these concerns.

The above queries are meant to represent what will soon be possible with a cohesive biological database management system such as MoBIoS. We have demonstrated the feasibility of performing useful queries on sequence data within a database management system itself, offering an alternative to the chains of programs previously necessary to solve complex genomics problems. However, many questions remain unanswered. We have yet to address the issue of regular expressions in queries. We also have not focused our attention on how to handle secondary and tertiary structure information. Elements of other bioinformatics and string algebras are under consideration to support these goals [TP03].

References

[BKML+02] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler. Genbank. *Nucleic Acids Res.*, 20(1):17–20, 2002.

[BLM+03] W. J Briggs, W. Liu, R. Mao, W. Xu, and D. P. Miranker. SQL extensions and database mechanisms for managing biosequences. Technical Report TR-04-05, The University of Texas at Austin, Department of Computer Sciences, December 2003.

[BPM+04] G. Bejerano, M. Pheasant, I. Makunin, S. Stephen, W.J. Kent, J.S. Mattick, and D. Haussler. Ultraconserved elements in the human genome. *Science*, 304(5675):1321–5, May 2004.

[Bri95] S. Brin. Near neighbor search in large metric spaces. In *Proc. 21st Conference on Very Large Database (VLDB'95)*, pages 574–584, 1995.

[CNBYM01] E. Chavez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[DGZ03] V. Dohnal, C. Gennaro, and P. Zezula. Similarity join in metric spaces using ed-index. In *Proc. of the14th International Conference Database and Expert Systems Applications (DEXA 2003), Volume 2736 of Lecture Notes in Computer Science*, pages 484–493. Springer, May 2003.

[Gur00] T. Guru. A silence that speaks volumes. *Nature*, 404:804–808, 2000.

[HCH01] S.M. Hammond, A.A. Caudy, and G.J. Hannon. Post-transcriptional gene silencing by double-stranded RNA. *Nature Rev Gen*, 2:110–119, 2001.

[JS82] B. Jaeschke and H. J. Schek. Remarks on the algebra of non first normal form relations. In *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, 1982.

[LGY+03] L. P. Lim, M. E. Glasner, S. Yekta, C. B. Burge, and D. P. Bartel. Vertebrate microRNA genes. *Science*, 299(5612):1540, Mar 2003.

[LSJR+03] B. P. Lewis, I. Shih, M. W. Jones-Rhoades, D. P. Bartel, and C. B. Burge. Prediction of mammalian microRNA targets. *Cell*, 115:787–798, 2003.

[McK] http://www.mckoi.com.

[MPN+99] E. M. Marcotte, M. Pellegrini, H. L. Ng, D. W. Rice, T. O. Yeates, and D. Eisenberg. Detecting protein function and protein-protein interactions from genome sequences. *Science*, 285(5428):751–3, 1999.

[MXM03] D. P. Miranker, W. Xu, and R. Mao. Architecture and application of MoBIoS, a metric-space DBMS to support biological discovery. In *15th International Conference on Scientific and Statistical Database Management. (SSDBM03)*, pages 241–244, 2003.

[MXSM03] R. Mao, W. Xu, N. Singh, and D. P. Miranker. An assessment of a metric space database index to support sequence homology. In *Proc. of the 3rd IEEE Symposium on Bioinformatics and Bioengineering*, Washington D.C, March 2003.

[Sch97] G. D. Schuler. Sequence mapping by electronic PCR. *Genome Research*, 7(5):541–550, May 1997.

[STMO03] S. C. Sahinalp, M. Tasan, J. Macker, and Z. M. Özsoyoglu. Distance based indexing for string proximity search. In *Proc. of IEEE Data Engineering Conference, ICDE 2003*, pages 125–136, Banglore, India, 2003.

[TP03] S. Tata and J. Patel. PiQA: An algebra for querying protein data sets. In *15th International Conference on Scientific and Statistical Database Management. (SSDBM03)*, pages 141–151, 2003.

[WS90] T.L. Wang and D. Shasha. Query processing for distance metrics. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proc. of the 16th International Conference on Very Large Databases*, pages 602–613, Brisbane, Australia, August 1990.

[XBP+04] W. Xu, W. J Briggs, J. Padolina, W. Liu, R. Timme, C. R. Linder, and D. P. Miranker. Using MoBIoS' scalable genome join to find conserved primer pair candidates between two genomes. *Bioinformatics*, 20:i355–i362, 2004.

[XM04] W. Xu and D. P. Miranker. A metric model of amino acid substitution. *Bioinformatics*, 20:1214–1221, 2004.

[XMMW03] W. Xu, D. P. Miranker, R. Mao, and S. Wang. Indexing protein sequences in metric space. Technical Report TR-04-06, The University of Texas at Austin, Department of Computer Sciences, October 2003.

[ZBB] Xiaoyu Zhang, Chandrajit L. Bajaj, and Nathan Baker. Fast matching of volumetric functions using multi-resolution dual contour trees. (submitted for publication).

Querying BLAST within a Data Federation

Barbara A. Eckman
IBM Life Sciences
1475 Paoli Pike
West Chester, PA 19380
baeckman@us.ibm.com

Arthur Kaufmann
IBM Silicon Valley Laboratory
555 Bailey Rd
San Jose, CA 95141
artkauf@us.ibm.com

Abstract

BLAST (Basic Local Alignment Search Tool) is one of the most widely used algorithms in bioinformatics and genomics. Frequently Life Science researchers wish to integrate the BLAST algorithm with other related data sources, either to supply BLAST query sequences or to provide additional annotations on sequences that are found to match. Another frequent need is to filter BLAST alignments based on match stringency or more complex criteria (e.g., the amino acid composition of the alignments). Further, in the context of data federation, wide-ranging multi-source queries involving BLAST searches often return unmanageably large result sets, requiring approaches that go beyond vanilla SQL to exclude extraneous data. Since 2001, IBM has provided the ability to access BLAST from within SQL queries, thus integrating BLAST results with relevant data from a wide variety of data sources, both local and remote, regardless of their format. In this paper we describe IBM® DB2® Information Integrator, IBM's federated database product, along with its BLAST wrapper and a suite of functions supporting complex non-relational queries over the composition of blast alignments.

1 Introduction

The BLAST (Basic Local Alignment Search Tool) [AGM$^+$90] sequence comparison algorithm has a venerable history in bioinformatics and genomics, and is still likely the most widely used tool in these research communities. Frequently researchers wish to integrate BLAST with other related data sources, either to supply BLAST query sequences or to provide additional annotations on sequences that are found to match. While publicly funded organizations like the National Center for Biotechnology Information (www.ncbi.nlm.nih.gov) and the Swiss Institute for Bioinformatics (www.expasy.org) offer browser-based integration between BLAST and data sources such as GenBank [WCE$^+$04] and Swiss-Prot/TrEMBL [BBFG04], they do not support integrating BLAST and related data sources via a declarative query language. Another frequent need among researchers is to filter BLAST alignments based on match stringency or more complex criteria (e.g., the amino acid composition of the alignments). Popular BLAST implementations themselves support only a limited set of filters over BLAST results (e.g., an E-value threshold). Since wide-ranging multi-source queries involving BLAST often return unmanageably large result sets, excluding extraneous data and thereby limiting the number of results that an expert scientist has to examine is a high priority. These filter criteria can be much more complex than a simple E-value threshold, often surpassing what can be expressed in vanilla SQL.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Our solution to these critical needs is to provide access to BLAST within a federated [HM85] relational query engine, thus providing full SQL query capability over BLAST results and enabling joins between BLAST inputs and outputs and other data sources in the federation. In addition, special-purpose functions provide regular expression pattern-matching capability over the composition of BLAST alignments.

2 IBM DB2 Information Integrator

DB2 Information Integrator (DB2 II) is a federated RDMBS that traces its lineage from the Garlic Project at IBM's Almaden Research Center [CHS^{+}95], IBM's DataJoiner® product (www.ibm.com/software/data/datajoiner), and our DiscoveryLink® solution [HSK^{+}01]. A federated strategy is appropriate in the bioinformatics research community, since information is not contained solely in databases, but instead exists in a multiplicity of formats and is accessed by a wide variety of tools and algorithms. A typical question asked by a researcher may span many separate data stores, as illustrated in the following scenario.

Voltage-sensitive calcium channel proteins mediate the entry of calcium ions into cells, and are involved in such processes as neurotransmitter release. The discovery of a novel channel gene that codes for a protein mediating calcium or similar ions into cells would potentially be of great interest to pharmaceutical researchers. A popular method of novel gene discovery is to search EST (expressed) sequence databases for sequences similar to known genes or proteins.

To identify potential new human neurological drug targets, a researcher might wish to identify mouse genes annotated as channels that are expressed in central nervous system tissue from the Mouse Genome Database, retrieve the sequences of their protein products from SwissProt, BLAST the protein sequences against the human EST database, return only BLAST alignments that meet certain stringency criteria (e.g., > 60 % identical over > 100 base pairs), and finally retrieve sequence annotations on the hit sequences from GenBank.

Answering this question manually is a daunting and error-prone task, as it entails visiting hundreds of web pages and manually collating and filtering the results. The question can also be answered using custom scripts, but this requires specialized skills and even a minor change in the question can require a substantial reprogramming effort. Another approach to answering the question might entail transferring or copying data from each data store into a single repository (e.g., a relational database) and performing the query from there. This introduces a number of problems, including data quality issues (currency and the potential for errors to be introduced during the transfer process) and an inability to use search tools that were not implemented to work with the repository (e.g., BLAST.)

DB2 II is specifically intended to address issues like these. In a federated system, diverse data stores and search tools (called *data sources*) remain outside the federator, avoiding data quality problems and loss of functionality. Through the use of software bridges called *wrappers* the data sources appear as if they are local to the federating software. The federator can then perform the actions that might otherwise be done manually or through custom programming.

In DB2 II, data sources are represented by a number of persistent objects, the most significant of which is a *nickname*. Nicknames have all the characteristics of relational tables and may be used in an SQL query anywhere a relational table may be used. Nicknames have the additional property that, rather than representing data stored in a relational table, they can represent data stored in many places and accessed through many tools. The sources and tools may be as diverse as relational DBMS's (e.g., Oracle® , Microsoft® SQL Server, Sybase® SQL Server), databases accessible via ODBC (e.g., Microsoft® Access, MySQL), web sites (e.g., Entrez Nucleotide and PubMed at www.ncbi.nlm.nih.gov), web services using SOAP/WSDL (e.g., XEMBL, www.ebi.ac.uk/xembl/XEMBL.wsdl, and KEGG, soap.genome.ad.jp/KEGG.wsdl), Microsoft® Excel spreadsheets, XML files, document repositories (e.g., IBM® Lotus® Extended Search), tabular text files (e.g., comma-separated value files), and search algorithms (e.g., BLAST, HMMER [Edd00].)

Thus, with DB2 II the question posed by the hypothetical researcher above may be answered by a single

SQL query:[1]

```
SELECT
    g.accnum, g.sequence
FROM
    genbank g, blast b, swissprot s, mgd m
WHERE
    m.exp = "CNS"                 /* expressed in central nervous system tissue*/
    AND m.defn LIKE  "%channel%"  /* genes annotated as channel genes */
    AND m.spid = s.id             /* join between MGD and SwissProt */
    AND s.seq = b.query           /* join between SwissProt and BLAST, to
                                     supply BLAST query sequence */
    AND b.hit = g.accnum          /* join between BLAST output and GenBank, to
                                     retrieve full-length sequence */
    AND b.percentid > 60          /* BLAST stringency filters */
    AND b.alignlen > 50
```

Besides ease of use, there are two significant advantages to the federated approach. First, the federator is able to apply sophisticated query optimization techniques to improve performance. The DB2 II Cost Based Optimizer interacts with the wrappers and their specific knowledge of each data source to produce an efficient plan to answer each query. The second advantage is one of compensation. Most non-relational data sources lack advanced filtering capabilities. The federator can compensate for this by applying filters itself. For instance, the *blastall* program cannot filter results by percent identity. In DB2 II, the federator can apply such a filter.

2.1 The Request-Reply-Compensate protocol

The DB2 II optimizer and wrapper cooperate during query planning to produce an optimal plan. This is done through the request-reply-compensate protocol [RS97][IBM03]. The process begins with the optimizer decomposing a query into parts that can be processed locally and parts that must be processed, at least in part, by wrappers and their remote data sources. In the case of a simple, single-table (or nickname) query, this decomposition will be simple. In the case of a query involving multiple tables and nicknames, there may be multiple possible decompositions.

Each decomposed part is called a query fragment. Fragments that reference nicknames for a particular wrapper are presented to that wrapper as a Request. Each Request consists of a list of projected values (head expressions), quantifiers (nicknames) and predicate expressions. The wrapper examines the Request and builds a corresponding Reply, consisting of the elements that the wrapper and data source can process. A predicate may represent a filter that the data source is unable to process. For example, a query fragment sent to the BLAST wrapper may contain a predicate on the length of the alignment. In this case, the predicate will not be added to the Reply.

In some cases, a Reply may be entirely empty. For instance, if the original query contains a join between two nicknames managed by the same wrapper, but that wrapper or data source does not process joins, the entire Request will be rejected. Another example involves data sources like BLAST and most web sites, which have required predicates (e.g., the BLAST input query sequence). If all required predicates are not included in the fragment, the wrapper will reject the Request.

When a Reply is received by the optimizer, it determines which elements from the Request have been left off, and adds local processing to compensate for those missing elements. Once a full set of replies for a single query has been received and local compensations determined, the optimizer uses costing information associated with the replies and the local processing to determine the best plan.

[1] Slightly simplified, with intra-source join predicates omitted for clarity.

3 The DB2 II BLAST wrapper

The BLAST wrapper is one of many wrappers for non-relational Life Science data sources written by IBM and provided with DB2 II. It is designed to work with NCBI *blastall* as well as other similar BLAST engines such as TurboBLAST® (www.turboworx.com). The only requirement for compatibility is that the BLAST search tool support the same command line arguments and return results with the same XML schema as NCBI *blastall*. In wrapping the native BLAST search engines we are able to take advantage of their speed, while incurring minimal overhead.

Each BLAST nickname represents a potential search using a particular algorithm (blastN, blastP, blastX, TblastN, or TblastX) against a particular BLAST-able database (e.g., SwissProt, EMBL [KAA+04], GenBank nr) Inputs to and outputs from the BLAST program are represented by columns of the nickname.

Inputs to the BLAST program are applied through predicates on designated columns. For instance, the sequence to be used in a search is provided through a predicate on the BlastSeq column:

... WHERE BlastSeq = 'ACTCGATC'

Outputs from the BLAST program may be projected out of the query, used in selection predicates, or joined with local tables or other data sources. The BLAST wrapper provides the ability to parse the definition line (*defline*) associated with each entry in a BLAST-able database. This allows the user to extract values such as GI number, accession number, and organismal taxonomy. For example, a scientist may join BLAST output with GenBank Nucleotide on accession number, using the DB2 II Entrez wrapper, to retrieve sequence annotation associated with the BLAST hits, or she may filter BLAST output on organismal taxonomy, returning only hits from mammalian species.

3.1 Improving Performance: Materialized Query Tables

Whether performed in the context of a federated system or on their own, BLAST queries tend to be very expensive, in terms of CPU utilization and elapsed time. Materialized Query Tables (MQTs), a feature of a soon-to-be-released version of DB2 II, enable the reuse of query results, thus yielding cost savings.

MQTs must be explicitly defined by a user or database administrator. An MQT definition consists of a table specification and a query. At some point in time (this varies, depending on the MQT definition), DB2 II executes the MQT query and populates a local table with the results. Then, when DB2 II receives another query that can be partially, or completely, satisfied by the data in the MQT, the DB2 II optimizer produces a plan that references the local table rather than the data source. When the query to the data source is expensive, this results in a savings of time and resources. The ability quickly to re-execute queries over previously generated BLAST results is especially important in cases where the initial queries' predicates were too stringent, resulting in no results returned.

There are caveats with the use of MQTs. The first is data currency. With MQTs that involve nicknames, currently DB2 II must be explicitly instructed to refresh the information in the local table (non-nickname MQTs may be updated automatically in some situations.) The second issue relates to how the MQT is defined: if the definition is not sufficiently broad, the MQT will not be reused and a net loss of resources will result (the time to populate the local table and the storage the local table occupies.)

4 How Scientists use BLAST Within DB2 Information Integrator

Scientists often want to see more annotations on BLAST hit sequences than the BLAST application itself can return, since it is limited to displaying the annotations encoded in the sequences' definition lines in the FASTA input file. They also often want to filter BLAST output based on the content of the definition line, which is not standardized but varies with each individual BLAST-able sequence database. For example, customers at the Center for Medical Genomics and Research and Academic Computing at Indiana University were interested in the following: Given a query sequence, BLAST the GenBank nucleotide database (NT) and return only hits against sequences that were not associated with a cloning vector. For each hit, retrieve the Cluster ID and Title from Unigene (a relational version in DB2), in addition to the GenBank accession number, description and E-Value. Display only the top five hits (i.e., the five lowest E-values). The following SQL query accomplishes this:

```
SELECT
   nt.GB_ACC_NUM, nt.DESCRIPTION, nt.E_VALUE, useq.CLUSTER_ID, ugen.TITLE
FROM
   ncbi.BLASTN_NT nt, unigene.SEQUENCE useq, unigene.GENERAL ugen
WHERE
    BLASTSEQ = 'GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGC
                CGAGGCGGGCGGATCACGAGGTCAGGAGATCGAGACCATCCTGGCTA
                ACACGGTGAAACCCCGTCTCTACTAAAAATACAAAAATTAGCCGGGC
                GTGGTGGCGGGCGCCTGTAGTCCCAGCTACTC'
    AND  nt.DESCRIPTION  NOT LIKE  '%cloning vector%'
    AND nt.GB_ACC_NUM = useq.ACC
    AND useq.CLUSTER_ID = ugen.CLUSTER_ID
ORDER BY E_VALUE
FETCH FIRST 5 ROWS ONLY
```

Accessing BLAST alignments via SQL also permits filtering BLAST results in ways that BLAST itself does not support, such as by percent identity or alignment length. In addition, DB2 II also permits complex queries over the composition of alignments using IBM's regular expression pattern-matching user-defined functions (UDFs). For example, scientists sometimes want to filter alignments based on motifs found in the target sequence. In the paper introducing PHI-BLAST [ZSM+98], CED4, the C elegans regulator of cell death, is used as the query sequence vs. nr, the non-redundant NCBI protein database. The scientist wishes to return only alignments in which the target sequence includes the P-loop ATPase domain, specified by the PROSITE [HSS+04] motif '[GA]-x(3)-G-K[ST]'. The following query expresses this economically:

```
SELECT
    accnum, definition
FROM
    Blastp b, GBseq gs, gbfeat gf, gbqual gq
WHERE
    gs.primary_accession = 'X69016'          /* CED4 nucleotide sequence */
    AND gs.sequencekey = gf.sequencekey
    AND gf.featurejoinkey = gq.featurejoinkey
    AND gf.FeatureKey = 'CDS'               /* get its coding sequence */
    AND gq.QualifierName = 'translation'    /* translate it to protein */
    AND gq.QualifierValue = b.BlastSeq      /* use protein as blast query */
    AND db2ls.LSPatternMatch(               /* filter by PROSITE motif */
            HSP_H_Seq,
            db2ls.LSPrositePattern('[GA]-x(3)-G-K[ST]')
        ) > 0;
```

This query calls two DB2 Life Science-specific built-in functions: LSPatternMatch(*string*, *Perl regular expression*) takes as input a string and a pattern in Perl regular expression syntax, and returns the number of maximal matches in the string. LSPrositePattern(*string*) takes a string expressing a pattern in PROSITE's motif syntax and returns the equivalent Perl regular expression.

In another example from our prior work with the OPM/TINet federated system [EKL01], when BLASTing with a proline-rich query sequence, a scientist may want to return only alignments that contain $< 25\%$ prolines among their perfect matches. To address this query, we have created the following elegantly-named DB2 function, which takes as input three strings and three patterns, finds the location(s) in the three strings where the three patterns are simultaneously satisfied, and outputs the offsets and the maximal matching substrings in each of the three input strings:

```
    CREATE FUNCTION LSMultiMatch3(
        string1     VARCHAR,
        pattern1    VARCHAR,
        string2     VARCHAR
        pattern2    VARCHAR,
        string3     VARCHAR,
        pattern3    VARCHAR)
    RETURNS TABLE (
        position        INTEGER,
        string1_match   VARCHAR,
        string2_match   VARCHAR,
        string3_match   VARCHAR).
```

To exclude overly proline-rich alignments from the BLAST output, the scientist may simply use the following query. Here *p* aliases the number of prolines that match perfectly in the alignment, and *m* aliases the total number of perfect matches. The LSBarCode( ) function reformats the midline string of the alignment, replacing all perfect matches with the symbol ”—” to facilitate pattern matching.

```
SELECT
    b.* , float(p)/ float(m) AS percent_prolines
FROM
    BlastOutput b,
    table(SELECT COUNT(*) AS p FROM table(
                db2ls.LSMultiMatch3(
                    b.HSP_Q_Seq, 'P',
                    db2ls.LSBarCode(b.HSP_Midline), '\|',
                b.HSP_H_Seq, 'P')
                 ) AS f
            ) AS y,
    table(SELECT COUNT(*) AS m FROM table(
                db2ls.LSMultiMatch3(
                    b.HSP_Q_Seq, '.',
                    db2ls.LSBarCode(b.HSP_Midline), '\|',
                    b.HSP_H_Seq, '.')
                 ) AS f
            ) AS z
WHERE
    BLASTSEQ =
        "MFETEADEKREMALEEGKGPGAEDSPPSKEPSPGQELPPGQDLPPNKDSPSGQEPAPSQE
        PLSSKDSATSEGSPPGPDAPPSKDVPPCQEPPPAQDLSPCQDLPAGQEPLPHQDPLLTKD
        LPAIQESPTRDLPPCQDLPPSQVSLPAKALTEDTMSSGDLLAATGDPPAAPRPAFVIPEV
        RLDSTYSQKAGAEQGCSGDEEDAEEAEEVEEGEEGEEDEDEDTSDDNYGERSEAKRSSMI"
    AND float(p) / float(m) < 0.25;
```

5 Related Work

The OPM/TINet and K2/Kleisli [DCB$^+$01] federated multidatabase query systems both include wrappers/drivers for BLAST. Instead of the relational data model of DB2 II, they are based on an object-relational data model and a functional programming language model, respectively. These richer data models have advantages in terms of expressivity, but make query optimization more difficult. OPM/TINet caches BLAST results with automatic expiration and refresh, but does not cache on the view level (e.g., a view joining BLAST output with GenBank annotations), as DB2 MQTs do.

Oracle's approach to integrating BLAST in their Oracle® Database10g release (www.oracle.com) requires users to maintain a local copy of sequence data in an Oracle data warehouse. The parsing and loading steps required to maintain such a warehouse constitute a potential point of failure, may introduce errors in the data, and may significantly delay scientists' access to the sequences. Oracle reimplements BLAST as a series of functions that are executed within the database against pairs of sequences; consequently, BLAST output statistics like E-value that are based on the size and composition of the BLAST-able database will differ from the "gold standard" of NCBI BLAST. In addition, any enhancements that NCBI makes to its BLAST implementation must be reimplemented in order to be available to Oracle's BLAST users. Performance will generally be slower than the native BLAST, since the native BLAST optimizations will not be available in the reimplemented version. On the other hand, since BLAST-able datasets are defined dynamically by a SQL query rather than created ahead of time with *formatdb*, if the dynamically defined dataset is much smaller than the closest *formatdb*ed database, there may be a net performance gain, especially with expensive searches like TBlastN. Finally, Oracle's federated capability is limited in terms of optimization strategies and the number and range of Life Science-specific data sources that are currently federated.

PHI-Blast and Bla [TK94] both enable users to filter BLAST output based on motifs found in the target sequences. PHI-Blast uses the motif of interest to restrict the similarity search space, yielding the advantages that in general it can be expected to be faster than a normal BLAST search, and its statistical analysis is tailored to this approach. On the other hand, Bla is similar to our approach in post-processing BLAST results to find motif matches. Neither of these approaches, however, is as flexible as ours, since they allow only limited filters on BLAST output while we provide the full power of SQL in specifying filters, in addition to full Perl regular-expression pattern-matching. Instead of generating a new BLAST executable, we apply our federated query capability to the output of standard BLAST–a more maintainable and scalable strategy.

6 Challenges and Future Directions

As noted, the DB2 II BLAST wrapper is currently certified on NCBI BLAST and TurboBLAST®. Since the BLAST wrapper architecture is designed to support any BLAST engine that implements the NCBI interface, certifying it on additional BLAST engines in the bioinformatics community is an obvious next step.

Beyond the BLAST wrapper itself, there are plenty of areas in which further research is needed. For the query engine, a key topic is the exploitation of parallelism to enhance performance. There is also a need for additional tools and facilities that enhance the basic DB2 II offering. We have done some preliminary work on a system for data annotation that provides a rich model of annotations, while exploiting the DB2 II engine to allow querying of both annotations and data separately and in conjunction. We are also building a tool to help users create mappings between source data and a target, integrated schema [MHH00] to ease the burden of view definition and reconciliation of schemas and data that plagues today's system administrators.

7 Conclusion

The BLAST algorithm is used nearly universally in the bioinformatics community to identify sequence similarity in support of a wide variety of research aims, e.g., gene function characterization, pharmaceutical target identification, protein structure prediction, vaccine development, personalized approaches to healthcare, and bioterrorism detection. In all these activities, integrating BLAST with related data sources, regardless of their location and format, is a critical need. IBM's DB2 Information Integrator is an effective means to efficiently and accurately integrate BLAST into a scientific data source federation, while safeguarding the scientific integrity of the results.

References

[AGM+90] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[BBFG04] A. Bairoch, B. Boeckmann, S. Ferro, and E. Gasteiger. Swiss-Prot: Juggling between Evolution and Stability. *Brief Bioinform*, 5(1):39–55, 2004.

[CHS+95] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In *Proceedings of the 5th International Workshop on Research Issues in Data Engineering-Distributed Object Management (RIDE-DOM'95)*, page 124. IEEE Computer Society, 1995.

[DCB+01] S.B. Davidson, J. Crabtree, B.P. Brunk, J. Schug, G.C. Overton, and Jr. C.J. Stoeckert. K2/Kleisli and GUS: Experiments in Integrated Access to Genomic Data Sources. *IBM Systems Journal*, 40(2):512–531, 2001.

[Edd00] S. Eddy. HMMER: Profile Hidden Markov Models for Sequence Analysis. http://hmmer.wustl.edu. 2000.

[EKL01] B. A. Eckman, A. S. Kosky, and L. A. Laroco. Extending Traditional Query-Based Integration Approaches for Functional Characterization of Post-genomic Data. *Bioinformatics*, 17(7):587–601, 2001.

[HM85] D. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Trans Office Inf Systems*, 3:253–278, 1985.

[HSK+01] L. M. Haas, P. M. Schwarz, P. Kodali, E. Kotlar, J. E. Rice, and W. C. Swope. DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources. *IBM Systems Journal*, 40:489–511, 2001.

[HSS+04] N. Hulo, C.J. Sigrist, S. Le Saux, P.S. Langendijk-Genevaux, L. Bordoli, A. Gattiker, E. De Castro, P. Bucher, and A. Bairoch. Recent Improvements to the PROSITE Database. *Nucleic Acids Res*, 32:D134–D137, 2004.

[IBM03] IBM. DB2 Information Integrator V8.1 Developer's Guide. http://publibfp.boulder.ibm.com/epubs/pdf/c1873590.pdf, 2003.

[KAA+04] T. Kulikova, P. Aldebert, N. Althorpe, W. Baker, K. Bates, P. Browne, A. van den Broek, G. Cochrane, K. Duggan, R. Eberhardt, N. Faruque, M. Garcia-Pastor, N. Harte, C. Kanz, R. Leinonen, Q. Lin, V. Lombard, R. Lopez, R. Mancuso, M. McHale, F. Nardone, V. Silventoinen, P. Stoehr, G. Stoesser, M.A. Tuli, K. Tzouvara, D. Vaughan, R.and Wu, W. Zhu, and R. Apweiler. The EMBL Nucleotide Sequence Database. *Nucleic Acids Res*, 32:D27–D30, 2004.

[MHH00] Renee J. Miller, Laura M. Haas, and Mauricio Hernandez. Schema Mapping as Query Discovery. *Proc. of the Conf. on Very Large Data Bases (VLDB), Cairo, Egypt*, pages 77–88, 2000.

[RS97] M. Tork Roth and P Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. *Proceedings of the 23rd VLDB Conference, Athens, Greece*, 1997.

[TK94] R. L. Tatusov and E. V. Koonin. A Simple Tool to Search for Sequence Motifs that are Conserved in BLAST Outputs. *Computer Applications in the BioSciences (CABIOS)*, 10(4):457–459, 1994.

[WCE+04] D.L. Wheeler, D.M. Church, R. Edgar, S. Federhen, W. Helmberg, T.L. Madden, J.U. Pontius, G. D. Schuler, L.M. Schriml, E. Sequeira, T.O. Suzek, T. A. Tatusova, and L. Wagner. Database resources of the National Center for Biotechnology Information: update. *Nucleic Acids Res*, 32:D35–D40, 2004.

[ZSM+98] Z. Zhang, A. A. Schaffer, W. Miller, T. L. Madden, D. J. Lipman, E. V. Koonin, and S. F. Altschul. Protein Sequence Similarity Searches Using Patterns as Seeds. *Nucl. Acids Res.*, 26(17):3896–990, 1998.

ODM BLAST: Sequence Homology Search in the RDBMS

Susie Stephens
Oracle Corporation
10 Van de Graaff Dr.
Burlington, MA 01803
susie.stephens@oracle.com

Jake Y. Chen *
Indiana University
School of Informatics
Indianapolis, IN 46202
jakechen@iupui.edu

Shiby Thomas
Oracle Corporation
10 Van de Graaff Dr.
Burlington, MA 01803
shiby.thomas@oracle.com

Abstract

Performing sequence homology searches against DNA or protein sequence databases is an essential bioinformatics task. Past research efforts have been primarily concerned with the development of sensitive and fast sequence homology search algorithms outside of the relational database management system (RDBMS). Oracle Data Mining (ODM) BLAST enables BLAST to be performed in a RDBMS. ODM BLAST relieves the burden of moving data out of the RDBMS, eliminates the need to parse data files, and allows BLAST results to be integrated with existing RDBMS data. Oracle has simplified BLAST searches to a single SQL statement. ODM BLAST shifts algorithm development from bioinformaticians to the RDBMS provider.

1 Introduction

Sequence homology searching is an essential bioinformatics task. Tools such as BLAST [AGM+90] and FASTA [Pea00] can be used to search a query sequence against a target database of sequences. If matched sequences are found, users can further examine sequence similarity to determine the identity of the query sequence, or characterize its functions by homology. With the rapid accumulation of genomic sequences, sequence homology searching has become a daily routine for genome annotation, comparative genomics, and evolutionary biology studies.

High-throughput biological data from sequencing machines, microarrays and protein mass spectrometers present new challenges for sequence homology searching. Web-based sequence homology search tools are popular; however, users can only perform searches one at a time. To perform the large-scale batch searches that are now required, software developers have had to build stand-alone sequence homology search servers. Frequently the software needed to perform such tasks has been written for individual groups, and consequently has poor portability and customizability. This re-invention of software functions across many organizations is an ineffective use of resources. Unless there is a robust strategy to integrate the results of homology searches with the in-house biological data that are managed in relational databases, the interpretation of gigabytes of sequence data becomes intractable.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Also at: Purdue University, School of Science, Department of Computer and Information Science, Indianapolis, IN 46202.

Past research efforts have been primarily concerned with the development of sequence homology search algorithms outside of the RDBMS [Ken02, Sim99]. Additionally, a relational database operator, *Similar_Join*, was developed to make an abstraction of batch sequence homology searches [CC03], and DiscoveryLink relied upon application and data source wrappers to make results from tools such as BLAST available to SQL analysts [HSK$^+$01].

ODM BLAST enables data analysts to use SQL to invoke BLAST functions in Oracle Database 10*g*. This work is built on the idea of extending the capability of a general-purpose RDBMS into the biology domain. With ODM BLAST being integrated into the RDBMS, data can remain in the RDBMS for analysis, which has performance and data management advantages. Once data have been entered into the database, no more parsing of the data is required, regardless of the group that is accessing the data. A strong RDBMS environment provides security, auditing, and high availability of data. With ODM BLAST, it becomes feasible to ask questions including "Retrieve similar sequences, where the sequence was entered into Genbank after 2002, and the sequence is from E. coli". Batch and automated SQL queries also become simpler, for example, "Query all human sequences against all yeast sequences".

2 Implementation

Oracle implemented BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX inside Oracle Database 10*g*. BLAST_MATCH can be invoked to retrieve the sequence identifier and similarity results; and BLAST_ALIGN can be invoked to retrieve the sequence identifier, similarity results and full alignment information. Detailed overview of ODM BLAST is available at [ora04]:

The ODM BLAST implementation takes advantage of the Oracle table function feature. This feature is part of the Oracle RDBMS extensibility framework, which allows developers to write code that is invoked using SQL queries. A table function returns its results as virtual tables, which can be manipulated like other relational tables. This implementation allows ODM BLAST to be invoked either by *ad hoc* SQL queries or by embedding the functionality into applications.

The ODM BLAST table function accepts a query sequence, a reference cursor that specifies the sequences that the query sequence needs to be searched against, and several other parameters that control the search. The query sequence is passed to the underlying server side programming code as a Character Large OBject (CLOB). The reference cursor, which specifies the target sequences, must contain two attributes: a sequence identifier of the data type VARCHAR and a sequence data string as a CLOB. The native programming code then takes these two input parameters, performs the search, and sends the results as a virtual table to the invoking ODM BLAST table function. Since the server-side process runs BLAST by loading query and target sequences from disk directly to memory, the overhead of copying files to different disk locations is eliminated.

2.1 Description

In Figure 1, a BLASTP_MATCH query was invoked to perform a protein sequence homology search against the target protein database *target_db*. The query sequence in *Block A* and the target database in *Block B* are specified as SQL sub-queries. The query sequence in *Block A* is specified on the fly. The query sequence in *Block B* is specified as a cursor for the subset of rows from the table *target_db*. Target sequences beginning with 'NP' are specified [PM01]. The top-level WHERE clause states that the search results must have an E-value of less than 1E-6 ($t.expect < 0.000001$). Finally, in the top-level SQL query FROM clause, *target_db* (table g) and BLASTP results (table t) were *JOIN*ed by *g.refseq_id = t.t_seq_id*. This enables search results to be integrated with annotation data in the RDBMS.

Writing SQL based BLAST queries should be simpler for bioinformaticians than writing PERL or Java code. The simple syntax should enable biologists to write basic queries. Queries could be further simplified if query

I. ODM BLAST Query Example

```
SELECT t.t_seq_id, t.score, t.expect, g.function_desc, g.locus_symbol, g.map_location
  FROM target_db g,
       TABLE                                                          -- BEGIN BLASTP
          (BLASTP_MATCH
              ((SELECT 'AKRELKKLKIEARSVERYKKLRIGMENKIMQLQRKVDEQNKDYKCLVEKLTNLEGIYNSETEKL
                        RSDLERLQLSEEEAKVATGRVLSLQEEIAKLRKDLEQTRSEKKCIEEHADRYKQET
                 FROM DUAL),                                                            Block A
               CURSOR (SELECT x.refseq_id, x.sequence_string
                         FROM target_db x
                        WHERE x.refseq_id LIKE 'NP%')
              )                                                  Block B
          ) t                                                          -- END BLASTP
 WHERE t.expect < 0.000001 AND g.refseq_id = t.t_seq_id
/
```

II. ODM BLAST Query Result Table

NP_000250	593	1.8956E-62	Class 5A myosin; motor protein that may be involved in organelle transport	MYO5A	15Q21

Figure 1: Oracle 10*g* BLAST Query and Result.

protein sequences are stored in the relational table *query_db*, and if *Block A* in the example above is replaced with the following SQL sub-query:

(**SELECT** sequence_string
FROM query_db
WHERE sequence_id = 100
).

3 Conclusions

ODM BLAST enables BLAST queries to be performed in Oracle Database 10*g*. ODM BLAST removes the overhead of moving sequence data out of the RDBMS, relieves the need to parse data files, and enables BLAST results to be integrated with existing relational data. Challenging queries become tractable with ODM BLAST. Batch BLAST queries can now be easily performed and ODM BLAST can span diverse data sets and incorporate annotations stored in relational databases. As large-scale integrative biology gains popularity, we expect ODM BLAST to become an essential database toolkit for bioinformaticians with challenging integrated sequence homology oriented queries.

References

[AGM+90] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[CC03] Jake Yue Chen and John V. Carlis. Similar_Join: Extending DBMS with a Bio-specific Operator. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 109–114, 2003.

[HSK+01] L. M. Haas, P. M. Schwarz, P. Kodali, E. Kotlar, J. E. Rice, and W. C. Swope. DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources. *IBM Systems Journal*, 40:489–511, 2001.

[Ken02] W. James Kent. BLAT: The BLAST-like Alignment Tool. *Genome Research*, 12(4):656–664, 2002.

[ora04] Oracle Data Mining Application Developer's Guide 10g Release 1. http://otn.oracle.com/industries/life_sciences/index.html, 2004.

[Pea00] W. R. Pearson. Flexible Sequence Similarity Searching with the FASTA3 Program Package. *Methods Mol. Biol.*, pages 185–219, 2000.

[PM01] K. D. Pruitt and D. R. Maglott. RefSeq and LocusLink: NCBI Gene-centered Resources. *Nucleic Acids Res.*, 29:137–140, 2001.

[Sim99] P. Simakov. Sequence Server Samurai. *Science*, 285:1226–1227, 1999.

Indexed Searching on Proteins Using a Suffix Sequoia

Ela Hunt
Department of Computing Science, University of Glasgow, Glasgow, G12 8QQ, UK
ela@dcs.gla.ac.uk

Abstract

Approximate searching on protein sequence data under arbitrary cost models is not supported by database indexing technology. We present a new data structure, suffix sequoia, which reduces the time complexity of the dynamic programming (DP) matrix calculation required in approximate matching. The data structure is compact. It uses just over 4 Bytes per symbol indexed. We show that time complexity of the DP calculation is $O(qg^d)$ for a pattern of length q, alphabet size g, and indexing window size d. The DP calculation requires no disk access, and can be executed efficiently. The second phase of the algorithm is based on sequential disk access, and appears to be effective. Approximate matching experiments are promising and offer a lot of scope for algorithm refinement and data structure engineering.

1 Introduction

Protein sequence searching is executed daily in many biological labs, and is a foundation of bioinformatics training [DEKM98, Mou01]. The target data used for searching may be mirrored locally, or accessed via a web form, for instance at the European Bioinformatics Institute `www.ebi.ac.uk` where BLAST [A^+90, AMS^+97] and FASTA [PL88] are made available. The purpose of a protein sequence search may be simply to find out if a protein is already known and characterised, and to associate a sequence with its identifier. Alternatively, and more commonly, the biologist wants to find biological information about the query by looking at similar sequences, with the hope that those sequences have already been analysed and may throw some light on the new sequence. Finally, an analysis of very large datasets of sequences may be undertaken, with the view to clustering the proteins into families, or identifying common sequence motifs. This type of requirement can only be carried out on a computer cluster or by using special hardware.

The amount of data available for protein searching is growing steadily. Recent releases of the protein databases Swiss-Prot and TrEMBL, at `www.expasy.org`, total 1.5 mln protein sequences containing 480 million letters of protein code. As organisms are getting sequenced at a fast rate, this data is expected to increase in the foreseeable future. To compare, in 2001 [HAI01] this database contained 200 mln letters of protein code. Proteins are stored as flat files, and the most common format, FASTA [PL88], contains a header line for each protein which contains its name and unique identifier, followed by lines of protein code. Protein code uses 20 letters, but FASTA files also contain three ambiguity codes, and this produces 23 symbols altogether.

Sequence searching methods for proteins use approximate matching, with either an edit cost model, or a similarity cost model. Exhaustive searching methods were developed by Needleman and Wunsch [NW70] and

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Smith and Waterman [SW81]. These algorithms guarantee that all relevant matches, given a cost model, will be returned as answers. Similarity between two sequences is calculated by building a matrix and comparing every symbol in the target with every symbol in the query, and recording the values in the matrix. The highest scoring alignment can then be traced back in the matrix and shown to the user. The observation that the dynamic programming matrix used in the similarity calculation is only sparsely filled has led to optimisations, including the SWAT algorithm, available from `http://bozeman.mbt.washington.edu/phrap.-docs/phrap.html`. Another optimisation, by Myers and Durbin [MD02] is due to the sparsity of positive values in the cost matrix. This optimised version computes around 4% of the dynamic programming matrix. It has only been tested with small data sets, of less than 10% of the current volume of proteins and is still quite slow, and therefore infeasible in most contexts. Heuristics based on finding exact matching words first [PL88], and then extending these seed matches by dynamic programming have led to the development of BLAST, which is the gold standard of protein sequence comparison. In blastp, blast for proteins, a query is split into windows, and for each window a significant number of similar windows are generated, where a number of symbols has been "mutated" to contain a different letter. These window families are then used in exact searching. If two matches are found close together, dynamic programming is used in an attempt to close the gap between those matches.

The type of protein searching we focus on is just a fraction of the entire spectrum of search types using the protein sequence. Other types of searching include regular expressions or protein profile construction, based on Hidden Markov Models [GKHC01]. Gusfield [Gus97] and Durbin et al. [DEKM98] provide details of the domain-specific algorithms.

2 Problem statement

A file S of n sequences of aminoacids (AA) of total length $|S|$ is given, along with a query q containing $|q|$ AA symbols. The alphabet has size $g = 23$ and the cost model is given as a matrix containing a similarity measure for each pair of symbols [Mou01]. Our simplified model does not consider gap costs. We build an index of all text windows of length d. The task of *approximate matching* aligns q with all sequences in S exhaustively. It computes a score for all overlapping query windows against all overlapping windows from all sequences, and derives a summative similarity measure for each target sequence in S. Top scoring sequences are then listed with the associated scores.

3 Suffix sequoia

Suffix sequoia indexes all windows of length d in a given string, or set of strings. It is inspired by the suffix tree which indexes all string suffixes (substrings starting anywhere within the text and finishing at the last text character) [Wei73, McC76, Ukk95], but is more compact, as it uses one integer per character indexed, plus some additional data structures. For the recent dataset of Swiss-Prot and TrEMBl, of 470 MB of AA, or 560 MB in FASTA format, the index uses 2 GB of disk.

In a suffix sequoia all overlapping text windows are indexed, so a sequence of length t will be decomposed into $t - d + 1$ windows of length d. Each window is represented as an integer in the range $[0, g^d - 1]$. As each new text character a is read from the source file, it is *mapped* to an integer in the range $[0, g - 1]$, and the code of the window it is being appended to is calculated from the code for the previous window, based on the formula

$$code(a_1 a_2..a_d) = \sum_{k=1}^{d} map(a_k) * g^{d-k}.$$

A full scan of S results in an array of size g^d where each array cell holds a list of integers representing *code* *positions* in S. Each such list is ordered because it is generated in a left to right traversal of S. In the experiments we describe, the ordinal number of the sequence, i.e. $1..n$, where the code appears is used, instead of the actual code position within S.

The suffix sequoia is created by scanning a FASTA file, while collecting information about protein names and the sequences where all overlapping AA windows of length d occur. The outline of index creation is given in ALGORITHM CREATEINDEX(S). Four types of files are written to disk. We create an index of sequence names, *Names*, by writing an array containing the first 90 characters from each sequence header. We write a *bitmap* of codes present in the sequence. The bitmap has one bit for each code in the range $[0, g^d - 1]$. If a bit c in that range is set this means that S contains a text window corresponding to code c. Additionally, we have two sets of associated files: *offsets* and *positions*, which we split into 23 files of each kind, for convenience. These files record the occurrence of particular codes in sequences $1..n$. Files *offsets0* and *positions0* correspond to all codes representing index windows starting with the letter A. *Offsets1* and *positions1* correspond to windows starting with B, and so on. As codes are not uniformly distributed in the sequence space, we need the *offsets* file to tell us where in the *positions* file to start reading, to get all the positions of a given index window. *Offsets* contain disk addresses pointing to the *positions* files. There is also a relationship between the bitmap and *offsets*. Each bit in the bitmap corresponds to an offset. If the code is present in S, its bit is set in the bitmap, and its offset is set to an integer address of its first position (except for the first offset in each file which is 0). We always read the bitmap first before reading or writing the offsets.

ALGORITHM CREATEINDEX(S), traverses S twice
given: a body of n AA sequences S, return: index on disk

```
create an array counts of size g^d
for i = 1 to n - d + 1
   code = code for window of length d starting at i
   counts[code]++
endfor
initialise a sequence counter x (range 1..n)
for i = 1 to n - d + 1
   update sequence counter x if a new sequence starts
   code = code for window of length d starting at i
   append x to a list[code] which stores code occurrences
endfor
initialise bitmap of size g^d, and pointer p to positions
for code in 0..g^d - 1
   if counts[code] > 0 then
      set bitmap for this code to TRUE
      write current position p in positions to offsets
      write the list list[code] to positions while incrementing p
   else write 0 to offsets
endfor
```

In this algorithm which makes a linear traversal of S, sequence numbers $1..n$ where the codes occur are ordered by code and position. All those numbers are placed consecutively in the *positions* file, and this allows us to look up the last position of each group of codes by looking up the next marked bit in the bitmap, and then looking up the corresponding code in *offsets*, and subtracting 1. The algorithm to look up all the sequences for a given code is as follows.

ALGORITHM: CODE LOOKUP(Y)
given: $y \in [0, g^d - 1]$, return: *list of sequence numbers* $\in [1..n]$ *where* y *occurs*
1. `if` bitmap(y) is not set return empty list
2. positions/offsets file to use: $pos = (int)\ (y\ \div\ g^{d-1})$
3. offset to read: $rem = y\ mod\ g^{d-1}$
4. $start$ = read $offsets_{pos}$ at rem
5. $next$ = get first set code from bitmap, starting at $y+1$
6. offset to end reading: $rem1 = (int)\ (next\ \div\ g^{d-1})$
7. end = (read $offsets_{pos}$ at $rem1$) - 1
8. $list$ = read all integers from $start$ to end in $positions_{pos}$, return $list$

4 Approximate matching

Sequence similarity is calculated using dynamic programming (DP). This involves filling in a matrix C of size $|t| \times |q|$ which aligns a text t to a query q, over an alphabet Σ, using a similarity function $S : \Sigma \times \Sigma \rightarrow integer$. In the biological scenario a gap cost model is used, which we currently leave out of the calculation. Our algorithm uses the following formula:

$$C(i,j) = max\{0,\ C(i-1,j-1) + S[q(i),t(j)],\ C(i-1,j),\ C(i,j-1)\}.$$

LEMMA 1: The unordered suffix sequoia index (where the code windows have not been ordered), based on window size d, reduces the size of the DP matrix to be filled in to

$$d \times |q| \times g^d.$$

PROOF: Build an index of all the possible text windows of length d. There are at most g^d such windows. Use DP to fill in all g^d arrays of size $d \times |q|$. The size of the DP computation is now bounded by $d \times |q| \times g^d$. □

THEOREM 1: The ordered suffix sequoia index reduces the size of the DP matrix to be filled in to

$$|q| \times (g + g^2 + ... + g^d) = \frac{|q|g(g^d - 1)}{g - 1}.$$

PROOF: Order all index windows alphabetically, which is equivalent to ordering them numerically. Slide a window of size d over the query. An example index over the alphabet of A, B and window size $d = 4$ has windows with codes 0..15, i.e. AAAA, AAAB, AABA, AABB, ABAA, ABAB, ABBA, ABBB, BAAA, AAAB, AABA, BABB, BBAA, BBAB, BBBA, BBBB. Window 0 of the index is aligned to a query window by computing a DP matrix of size d^2. Move on to index window 1. In the same matrix only the d cells corresponding to the last character B of window 1 need to be recomputed. Move to window 2 where two new index letters require recalculation and $2d$ cells are filled in anew. In window 3 only the last index letter changes and d cells are recalculated. The observed pattern of recalculation leads to a formula for the number of matrix cells to fill in. The first index letter is only aligned to the query twice, once in **A**AAA and once in **B**AAA and g cells are calculated. The second character is aligned to the query, $g * g$ times, the third $g * g * g$ times, and the last g^d times, in total $g + g^2 + ... + g^d$. Since the number of query windows is less than $|q|$, the total size of DP matrix is shown to be bounded by $|q| \times (g + g^2 + ... + g^d) = \frac{|q|g(g^d-1)}{g-1}$. □

ALGORITHM APPROXIMATESEARCH(Q,TH)
given: query q, return: list of sequences which have a score for any query window of size d above a threshold th
0. create an array $codeScores$ of size g^d to hold a score per index code
1. `for` $i = 1$ to $q - d + 1$ take query $window$ of length d starting at i
2. INDEXEDDP(WINDOW,TH) updates $codeScores$ array
3. `endfor`
4. create array of $sequenceScores$ of size n
5. `for` each $code > 0$ in $codeScores$
6. $sequenceList$ = CODELOOKUP(CODE)
7. `for` each seq in $sequenceList$
8. $sequenceScores[seq] += codeScores[code]$
9. `endfor`
10. `endfor`
11. traverse $sequenceScores$ and output the sequences

ALGORITHM INDEXEDDP(WINDOW,TH)
given: query $window$, threshold th, return: updated $codeScores$ where the score for codes matching query window has been incremented by the newly calculated score
0. initialise DPmatrix[d+1,d+1]
1. `for` each $code$ in the $bitmap$ of indexed codes
2. pos = position where this $code$ differs from the previous code for which DP was calculated (e.g. $3 = AAABA$ differs from $2 = AAAAB$ at $pos = 4$, and for $0 = AAAAA$ $pos = 1$)
3. calculate DP slice $[pos, d]$ for this $window$ and $code$
4. `if` $score > threshold$
5. $codeScores[code] += score$
6. `endfor`

5 Experiments

Tests were performed on IBM xSeries 235 server, with two 2.8GHz Intel Xeon^{TM} processors and 6 Gb of RAM, running Linux. Software was developed in Java, version 1.4. For access to files java.nio which provides memory-mapped files was used. As benchmark we used SWAT, version 0.990329, from Phil Green, phg@u.washington.edu. Default parameters were used.

A suffix sequoia for the dataset of 471,091,668 AAs was built, using window size 5. It was not possible to use the benchmark SWAT algorithm on the entire data set as the maximum number of sequences to search against is set at 64,000. We searched against the first 64,000 sequences (24,688,351 AAs) and then multiplied the measured time by 19.08, which is the result of dividing the total AA data 471,091,668 by 24,688,351. We compare the performance of SWAT with that of the suffix sequoia, where the threshold for a single window match was set at 10, in Figure 1. It appears that the suffix sequoia is much faster than the projected time for Smith-Waterman calculation, when the query length is above 250 AAs. This suggests that further work with this data structure might deliver fast approximate matching. In Figure 2 we analyse the contribution of the DP calculation and disk access times to the performance of indexed searching. It turns out that disk access time dominates DP calculation time. This opens up the possiblility of data structrure tuning, and the use of parallel IO in this context.

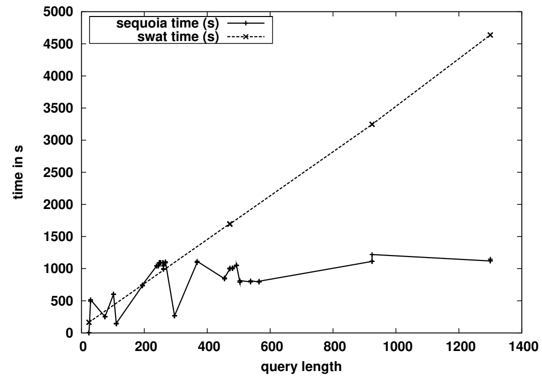

Figure 1: Sequoia query response time for 27 queries up to 1300 AAs. Each query was run 5 times. For SWAT we report interpolated time based on 3 runs each for 4 sequences of lengths 1300, 924, 472 and 24 AAs.

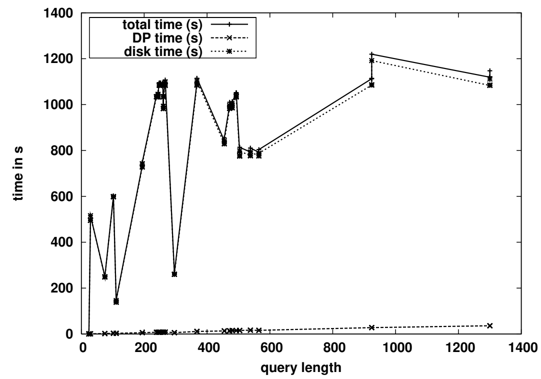

Figure 2: Comparison of DP calculation time with disk access time and total time for search, using suffix sequoia.

6 Previous work

Smith-Waterman algorithm can be accelerated by the use of multimedia instructions (wide registers) to accommodate several symbols which can be compared in parallel [R01], by the use of bit-vector algorithms, as proposed in [Mye99], or by the use of FPGAs [YMMK02]. Alternatively, coarse-grained parallelism can use computer clusters to distribute the computation.

In indexing solutions the following developments are seen to be of significance. IBM developed FLASH [AI93] which allows for fast searching, but does not exactly correspond to the biological scoring model. Approaches based on q-grams [Ukk92, NBY98, BCF+99, NSTT00] are fast and proven, but cannot deliver matches that have low similarity to the query [Nav00] and are not appropriate for protein matching. We demonstrated approximate matching on suffix trees [HAI02] and Meek and colleagues explored approximate searching on short protein strings [MPK03], based on suffix trees. This last paper provides an up-to-date overview of new database techniques.

7 Conclusions and further work

Our preliminary analysis of the suffix sequoia performance demonstrates that DP matrix size reduction improves significantly on the previous algorithms in this area. Using window of size 5 and the alphabet of 23 letters we would calculate the maximum 1.4% of the DP matrix per query character. We actually use an optimisation which starts DP only for the codes which would have a positive score for the first character of each query window, so we must be calculating a smaller percentage of the matrix than Myres and Durbin [MD02] who cite 4%. They claim to be twice as fast as SWAT, which for longer sequences will still be slower than our solution. In fact the optimised DP calculation time pales into insignificance when compared with the observed disk access times. This opens new avenues for engineering research using variants of suffix sequoia. We are planning to explore algorithms which might deliver exactly what the biologists want, i.e. Smith-Waterman algorithm, but much faster.

Acknowledgements

This work was funded by the Medical Research Council, UK, and by the Royal Society, UK. Thanks to Rob Irving and Juris Viksna for their comments on this manuscript.

References

[A+90] S.F. Altschul et al. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–10, 1990.

[AI93] A.Califano and I.Rigoutsos. FLASH: A Fast Look-up Algorithm for String Homology. In *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 56-64, Bethesda, MD, 1993.

[AMS+97] S.F. Altschul, T.L. Madden, A.A. Schaeffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[BCF+99] S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron. q-gram Based Database Searching Using a Suffix Array. In S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 77–83, Lyon, France, 1999. ACM Press.

[DEKM98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis. Probabilistic models of proteins and nucleic acids.* CUP, 1998.

[GKHC01] J. Gough, K. Karplus, R. Hughey, and C. Chothia. Assignment of homology to genome sequences using a library of hidden Markov models that represent all proteins of known structure. *Journal Molecular Biol.*, 313:903–919, 2001.

[Gus97] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

[HAI01] E. Hunt, M. P. Atkinson, and R. W. Irving. A database index to large biological sequences. In *Proc. 27th Conf. on Very Large Databases*, pages 139–148. Morgan Kaufmann, 2001.

[HAI02] E. Hunt, M. P. Atkinson, and R. W. Irving. Database Indexing for Large DNA and Protein Sequence Collections. *The VLDB Journal*, 11:256–271, 2002. DOI 10.1007/s007780200064.

[McC76] E.M. McCreight. A space-economic suffix tree construction algorithm. *Journal of the A.C.M.*, 23(2):262–272, April 1976.

[MD02] G. Myers and R. Durbin. Accelerating Smith-Waterman Searches. In *WABI 2002*, LNCS 2452, pages 331–342. Springer, 2002.

[Mou01] D. W. Mount. *Bioinformatics. Sequence and Genome Analysis.* Cold Spring Harbor Laboratory Press, 2001.

[MPK03] C. Meek, J. M. Patel, and S. Kasetty. OASIS: An Online and Accurate Technique for Local-alignment Searches on Biological Sequences. In *VLDB 2003*, pages 910–921, 2003.

[Mye99] E. W. Myers. A Fast Bit-Vector Algorithm for approximate String Matching Based on Dynamic Programming. *Journal of the ACM*, pages 395–415, 1999.

[Nav00] G. Navarro. A Guided Tour to Approximate String Matching. *ACM Computing Surveys*, 33(1):31–88, 2000.

[NBY98] G. Navarro and R. Baeza-Yates. A practical q-gram index for text retrieval allowing errors. *CLEI Electronic Journal*, 1(2), 1998.

[NSTT00] G. Navarro, E. Sutinen, J. Tanninen, and J. Tarhio. Indexing Text with Approximate q-grams. In Giancarlo R. and Sankoff D., editors, *Combinatorial Pattern Matching 2000, 11th Annual Symposium*, LNCS 1848, pages 350–365. Springer, 2000.

[NW70] S. B. Needleman and C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of two Proteins. *J. Mol. Biol.*, 48:443–453, 1970.

[PL88] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85:2444–8, 1988.

[SW81] T.A. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[R01] T. Rognes ParAlign: a parallel sequence alignment algorithm for rapid and sensitive database searches. *Nucleic Acids Research*, 29:7:1647–1652, 2001.

[Ukk92] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–212, 1992.

[Ukk95] E. Ukkonen. On-line construction of suffix-trees. *Algorithmica*, 14(3):249–260, 1995. TR A-1993-1, Department of Computing Science, University of Helsinki, Finland.

[Wei73] P. Weiner. Linear pattern matching algorithm. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, Washington, DC, 1973.

[YMMK02] Y. Yamaguchi, Y. Miyajima, T. Maruyama, and A. Konagaya. High Speed Homology Search Using Run-Time Reconfiguration. In *FPL 2002*, LNCS 2438, pages 281–291. Springer, 2002.

Progressive Searching of Biological Sequences

Tamer Kahveci
Department of Computer and
Information Science and Engineering
University of Florida
Gainesville, FL, 32611-6125
tamer@cise.ufl.edu

Ambuj Singh
Department of Computer Science
University of California
Santa Barbara, CA, 93106
ambuj@cs.ucsb.edu

Abstract

We consider the problem of progressive searching, and propose two k-NN (k-Nearest Neighbor) search algorithms for biological sequence databases, one for local alignment and one for global alignment. We develop a method to compute the confidence on the partial results. We also propose an early pruning strategy to reduce the total search time. Our experiments show that our techniques can achieve 75 % accuracy within the first 2.5-31 % of the iterations and 90 % accuracy within the first 12-37 % of the iterations.

1 Introduction

We investigate the problem of providing progressive access to genome sequences. We develop two models to quickly predict the proximity of database subsequences to a given query sequence. Our first model employs the Karlin-Altschul statistics [KA90] directly to database subsequences. This model computes the *unexpectedness* of an alignment by considering the distribution of letters in the entire database and query sequence. Our second model transforms database subsequences into vector space through the use of *frequency vectors* (count of each letter in a sequence) [KS01]. The frequency vectors are then clustered into MBRs (Minimum Bounding Rectangles). Then, we estimate the distance distribution of the subsequences in each MBR to the query subsequences with the help of *order statistics* [Cas88].

Using these models, we build a set of progressive k-NN (k-Nearest Neighbor) search techniques for both global (i.e., entire query sequence is aligned) and local alignment (i.e., query subsequences are aligned). Our first algorithm considers the local alignment problem. It partitions the database into overlapping blocks. Later, these blocks are ranked with the help of Karlin-Altschul statistics, and searched in this order using a highly optimized search tool, such as BLAST [AGM$^+$90]. The intermediate results are immediately reported.

Our next algorithm considers the global alignment problem. It hierarchically finds the next MBR of the index structure in ascending order of its kth-order statistics. Later, these MBRs are inspected iteratively using Needleman-Wunsch method [NW70]. We reduce the total search time of this technique by pruning the MBRs on the fly that do not contain results better than the ones found so far. We also develop a novel technique to

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

estimate the probability that the uninspected MBRs do not contain any results better than the results reported so far. This value is then reported as a *confidence estimate*.

Experimental results on DNA and protein sequences show that our methods achieve 75 % accuracy within the first 2.5-31 % of the iterations and 90 % accuracy within the first 12-37 % of the iterations. Our pruning strategy can prune up to 38 % of the database.

A recent method (named OASIS) [MPK03] uses suffix trees for accurate online searches for short query sequences. Our work differs from this paper in four ways. 1) Our index structures are very compact (i.e., 1-2 % of the database size [KS01]). 2) Our methods can handle long queries. 3) We can perform both local and global alignment. 4) We provide confidence estimates.

The rest of the paper is as follows. Section 2 presents background on sequence searching. Sections 3 and 4 discuss our progressive methods for local and global alignment respectively. Section 5 presents the experimental evaluation of our methods. Finally, we end with a brief discussion in Section 6.

2 Background on sequence searching

A sequence x can be transformed into another sequence y using three *edit operations*, namely *insert*, *delete*, and *replace*, on individual characters of the sequence x. The *edit distance* between two sequences is generally defined as the minimum number of edit operations to transform one sequence into the other. A special case of edit distance, *Gapless Edit Distance (GED)* or *Hamming Distance*, can be defined as follows.

Definition 1: Let x and y be two sequences of the same length. Let $x[i]$ be the ith character of x. We define the *Gapless Edit Distance*, *GED*(x, y), as the number of character pairs $(x[i], y[i])$ for which $x[i] \neq y[i]$. □

GED is an upper bound to the edit distance, and is widely used for statistical analysis [AG96, AW94]. We will also use *GED* in our analysis in Section 4.

Similarity between sequences can also be computed based on scores. An *alignment* of sequences x and y is obtained by matching each character of x to a character in y in increasing order. All the unmatched characters in both sequences are matched with space. Each character pair is assigned a score based on their similarity. These scores are stored in a *score matrix*. Each gap incurs a (potentially large) *gap open penalty* for the first space and a (smaller) *gap extend penalty* for all the spaces. The value of an alignment is defined as the sum of the scores of all of their character pairs. *Global alignment* of sequences x and y is their highest scoring alignment. On the other hand, their *local alignment* is alignment of their subsequences with the highest score. Both global and local alignments can be determined in $O(|x| \cdot |y|)$ time and space using dynamic programming [NW70, SW81].

Several heuristics that run faster than the dynamic programming, such as BLAST [AGM$^+$90], are widely used for sequence alignment. BLAST starts by finding exactly matching seeds with the help of a hash table. Later, these seeds are extended in both directions and appended to find longer matches. (See [KLS04] for a brief discussion of other important sequence alignment tools.)

3 Progressive local alignment

There is a substantial amount of work that estimates the score of the best local alignment of two sequences using global distribution of the letters in these sequences. We first discuss Karlin Altschul statistics [KA90] which is used by BLAST to compute the unexpectedness of a result. Then we explain how we employ it for progressive local alignment.

3.1 Background on Karlin-Altschul statistics

Let $\Gamma = \{\alpha_1, \cdots, \alpha_\gamma\}$. Assume that the letters are sampled with probabilities $\{p_1, \cdots, p_\gamma\}$ in the data sequence, and $\{q_1, \cdots, q_\gamma\}$ in the query sequence. Let the score of matching letters α_i and α_j be $s(\alpha_i, \alpha_j)$ such that $\sum_{i,j} p_i p_j s(\alpha_i, \alpha_j) < 0$ and $max_{i,j}\{s(\alpha_i, \alpha_j)\} > 0$. These conditions must be satisfied for any valid scoring scheme [KA93].

Let $f(\lambda) = \sum_{i,j} p_i p_j e^{\lambda s(\alpha_i, \alpha_j)}$. BLAST uses the unique positive solution, λ^* to the equation $f(\lambda) = 1$ in the statistical computation [KA90]. Karlin and Altschul [KA90] show that the expected value for the score of the maximal alignment of two sequences can be approximated as $\frac{\ln(mn)}{\lambda^*}$, where m and n are the lengths of the two compared sequences. Finding this value requires the computation of λ^*. We propose to use a direct solution technique based on the following observations: a) $f(0) = 1$, b) $f'(0) < 0$, c) $f''(0) > 0$, where f' and f'' are the first and second derivatives of f. These three conditions imply that f is a convex function. One of the solutions to $f(\lambda) = 1$ is obviously at $\lambda = 0$, and the other root is positive. We define $g(\lambda) = f(\lambda) - 1$, and use the Newton-Raphson Method [Ham87] to find the positive root of $g(\lambda) = 0$. In our experiments, this method converged in a few iterations.

3.2 Using Karlin-Altschul statistics for progressive local alignment

Here, we develop our first progressive search technique based on Karlin-Altschul statistics. We call this technique *Karlin-Altschul Statistics-based incremental search (KAS)*. KAS partitions the database sequences into overlapping blocks (i.e., subsequences) by placing a window of length n on the database sequences. Consecutive blocks are obtained by shifting the window by Δ letters. Each such positioning of the window produces one data base block. We discuss the choice of n and Δ in Section 5. For each block, we store the frequency of all the letters.

The score of the maximal local alignment for each block s_i is estimated as $\ln(n \cdot |q|)/\lambda_i^*$, where λ_i^* is defined the same as in Section 3.1 for s_i. Since n and $|q|$ are fixed, this estimate becomes larger when λ_i^* is smaller. Given a query sequence q, KAS works in 3 steps:

Step1: (Prediction) Compute λ_i^* for q and the subsequence s_i in partition i using the Newton-Raphson method, for all i.

Step 2: (Sorting) Sort λ_i^* in ascending order.

Step 3: (Searching) Search the partitions s_i in ascending λ_i^* order using an efficient search tool like BLAST. Report intermediate results to the user for each partition.

4 Progressive global alignment

Computing statistics for global alignment is more difficult than computing statistics for local alignment. Here we present a method for estimating global alignment statistics using frequency vectors and GED distributions.

4.1 Background on frequency vectors and MRS index structure

Let s be a sequence from an alphabet Γ, where $|\Gamma| = \gamma$. The *frequency vector*, $f(s)$, of s is defined as the γ-dimensional vector whose entries are the number of occurrences of the letters in Γ. For instance, the frequency vector of a DNA sequence has 4 dimensions since DNAs contain the letters A, C, G, and T. For example, for the DNA sequence $s = \texttt{GGTTCCTCTA}$, $f(s) = [1, 3, 2, 4]$.

The MRS index structure [KS01] maintains summary information for database sequences at resolutions of powers of two. At a given resolution w, the summary is constructed by sliding a window of length w on the database sequence. Each positioning of the window defines a frequency vector and a set of consecutive vectors defines an MBR. The size of this set determines the *capacity* of an MBR. The rest of the MBRs at this resolution are created similarly. Note that all the frequency vectors that constitute an MBR lie on the same multi-dimensional plane, called the *data plane*. This is because the entries of the frequency vectors in an MBR add up to the resolution of that MBR.

We will utilize the MRS index structure for our search techniques because of its compactness and efficiency. However, our techniques will extend to other index structures that cluster subsequences of database sequences.

4.2 GED distribution for two vectors

At the time of a database search, the query sequence is known but only the frequency vectors of database sequences are available in the index structure. Therefore, in order to plan the progressive query, we need to determine the distribution of the *GED* between two sequences using their frequency vectors.

Each frequency vector v defines an equivalence class, $\mathcal{S}_v$, of the set of sequences whose frequency vectors are equal to v. For example, if $v = [3, 0, 0, 1]$, then $S_v = \{\texttt{AAAT}, \texttt{AATA}, \texttt{ATAA}, \texttt{TAAA}\}$. Let $v = [v_1, \cdots, v_\gamma]$ be a frequency vector, where γ is the alphabet size, then the size of the equivalence class defined by v is $(\sum v_i)!/(\prod(v_i!))$.

In order to plan queries and compute statistics, we need to know the distribution of $\{GED(q, s) | s \in \mathcal{S}_v\}$. For simplicity, we will assume that s is uniformly distributed over $\mathcal{S}_v$. We do not make any assumptions on q. Using the Central Limit Theorem [Cas88], one can prove that the *GED* has a normal distribution as follows: Let the random variable Z_i represent the *GED* achieved by aligning the ith letters of both sequences for $1 \leq i \leq |s|$ (i.e., if $q[i] = s[i]$ then $Z_i = 0$. Otherwise, $Z_i = 1$). The *GED* of an alignment can be calculated as the sum of the values of all Z_is for that alignment. Since the sampling rate of all the letters is fixed and determined by the frequency vector of that sequence, the Z_is are iid (independent and identically distributed) random variables. Therefore, from the Central Limit Theorem, we conclude that the distribution of the *GED* between a query sequence and the sequences in the equivalence class of a frequency vector can be approximated using normal distribution for large $|s|$ (i.e., $|s| \geq 10$). Hence, one can compute this approximation if the mean and variance of the distribution are known. Theorems 2 and 3 develop formulas for the mean and the variance of this distribution that can be computed efficiently in $O(\gamma)$ time.

Theorem 2: (*Mean*): Let q be a sequence from alphabet $\Gamma = \{\alpha_1, \cdots, \alpha_\gamma\}$. Let $x = [x_1, \cdots, x_\gamma]$ be the frequency vector of q, and $y = [y_1, \cdots, y_\gamma]$ be a frequency vector, where $\sum_{i=1}^{\gamma} y_i = |q|$. Let $\mu_{q,y}$ be the mean of the *GED* distribution between q and the sequences in $\mathcal{S}_y$, then

$$\mu_{q,y} = |q| - \frac{\sum_{i=1}^{\gamma} x_i \cdot y_i}{|q|}. \qquad \square$$

Theorem 3: (*Variance*): Let q be a sequence from alphabet $\Gamma = \{\alpha_1, \cdots, \alpha_\gamma\}$. Let $x = [x_1, \cdots, x_\gamma]$ be the frequency vector of q, and $y = [y_1, \cdots, y_\gamma]$ be a frequency vector, where $\sum_{i=1}^{\gamma} y_i = |q|$. Let $\sigma_{q,y}^2$ be the variance of the *GED* distribution between q and the sequences in $\mathcal{S}_y$, then

$$\sigma_{q,y}^2 \approx \sum_{i=1}^{\gamma} \left(\frac{x_i \cdot y_i}{|q|} \cdot \left(1 + \frac{(x_i - 1)(y_i - 1)}{|q| - 1} - \frac{x_i \cdot y_i}{|q|}\right)\right). \qquad \square$$

4.3 GED distribution of a query to an MBR

So far, we considered the GED distribution between the frequency vector of a query sequence and that of a database sequence. Now, we consider the GED distribution between a query frequency vector and a set of database frequency vectors, specifically those contained in an MBR of an index structure. This computation poses two problems. First, it requires knowing the *GED* distributions of all frequency vectors contained in the MBR. However, an MBR does not store the individual frequency vectors it contains. It only maintains their span. Second, even if all the frequency vectors within an MBR are known, computing the GED distribution of all of them separately is time consuming.

We resolve these two problems by choosing a representative frequency vector for each MBR and assuming that an MBR contains a number of iid random variables based on the representative frequency vector. The representative frequency vector of an MBR has to be the vector which is closest to the rest of the vectors in the MBR. Furthermore, the sum of the entries of this vector must be equal to w, where w is the resolution of the MBR. This vector is found in two steps as follows: First, the centroid of the MBR is calculated as the average of the lower and higher coordinates of the MBR. Later, this centroid is projected onto the plane which contains frequency vectors for resolution w.

Once the representative frequency vector of an MBR has been chosen, we construct a random variable, X, having the GED distribution between the given query vector and the representative vector of that MBR as explained in Section 4.2. If the MBR contains c frequency vectors at resolution w, then the *GED* distribution of the query to the MBR is approximated by assuming $\lceil c/w \rceil$ random variables independent and identical to X. This is justified since the MBRs of the MRS index structure contains $\approx c/w$ independent subsequences.

The *GED* distribution between a frequency vector and a set of iid frequency vectors can be calculated using order statistics. The kth-order statistics of a set of random variables $\mathcal{X} = \{X_1, \cdots, X_n\}$ is defined as the random variable $Y_k \in \mathcal{X}$ that has the kth smallest value. Let X be a random variable having the same distribution as X_is ($1 \leq i \leq n$), then the cumulative distribution function of Y_k can be calculated as

$$P(Y_k \leq d) = \sum_{j=k}^{n} C_j^n \cdot P(X \leq d)^j \cdot (1 - P(X \leq d))^{n-j},$$

where C_j^n is n choose j. The probability mass function for Y_k is computed as

$$k \cdot C_k^n \cdot P(X \leq d)^{k-1} \cdot (1 - P(X \leq d))^{n-k} \cdot P(X = d).$$

Let M_d be the number of X_is ($1 \leq i \leq n$ and $0 \leq d$) with value less than or equal to d, then

$$P(M_d \leq m) = 1 - P(Y_{m+1} \leq d).$$

We use this distribution function to predict the number of sequences that are within a specified GED to a given query sequence in an MBR.

4.4 Using frequency statistics for progressive global alignment

Our second progressive search technique uses statistical information about the database subsequences. We call this technique the *Frequency Statistics-based incremental search (FS)* technique.

FS builds an MRS index structure on the database. Each MBR of this index defines a partition (i.e., subsequence) of $w + c - 1$ letters, where w and c are the resolution and the capacity of that MBR. Similar to KAS, FS works in three steps:

Step 1: (Prediction) In this step, the frequency vectors of all possible w-letter subsequences of the query are determined and their representative q_w is computed as their mean. For each MBR, the mean of the kth order statistics is then computed using q_w. This value is then normalized by w. This process is repeated for all resolutions w in the index, and the results are accumulated to determine the cumulative order statistics for each partition.

Step 2: (Sorting) The partitions are sorted in descending order of cumulative order statistics.

Step 3: (Searching) The partitions are aligned to the query sequence based on the ordering obtained in previous step, using an appropriate method, such as Needleman-Wunsch dynamic programming [NW70] The partial results are reported to the user immediately along with the confidence value.

Confidence estimate exhibits *how confident the user should be with the current partial results*. Confidence estimates are computed based on the results discovered so far and the *GED* distributions of the uninspected MBRs as follows. Let d_k be the GED to the kth closest sequence reported so far. $P(Y_1 \geq d_k)$ for an MBR represents the probability that that MBR does not contain any match whose distance is less than d_k. This value is computed for each uninspected MBR using the formulas given in Section 4.3. Later, these results are multiplied to find the probability that the closest subsequence in the remaining MBRs is not closer than the kth most similar subsequence found so far.

FS ranks database subsequences according to their similarity to the given query sequence in the frequency domain. These subsequences are then examined in this order, thus potentially finding better results earlier. However, FS still has to search the entire database in order to ensure the absence of false dismissals. The total search time of FS can be reduced by pruning the lower ranking subsequences that do not contain any result better than the kth-NN found so far. We propose to prune low quality subsequences in two steps:

Step 1: Let S_k be the score of the kth best match found so far. An upper bound, L, on the length of the kth best match is estimated. Define w to be the minimum resolution available in the MRS index structure, for which $w \geq L$.

Step 2: An upper bound, S_{upper}, to the score of the best alignment between the query and database subsequences contained in the next uninspected MBR at resolution w is computed. If $S_{\text{upper}} < S_k$, then this MBR is pruned. Otherwise, it is inspected for alignments. An efficient computation of S_{upper} is developed in our earlier paper [KLS04].

5 Experimental results

We used two classes of sequence datasets in our experiments: **DNA dataset** (`ftp://ftp.ncbi.nih.gov`) contains chromosome 18 (*chr-18*, acc. `NT_000864`), from *homo sapiens* database and the genetic code of *Escherichia coli* (*E.Coli* acc. `U00096`). each containing more than 4 M base pairs. **Protein dataset** (`ftp://ftp.expasy.ch`) contains all the proteins in the SWISSPROT database (available by January 2004). This dataset contains approximately 68 M residues.

We downloaded the source code of BLAST, and implemented the MRS index structure for window sizes $256 \leq w \leq 2048$, and box capacity $c = 1000$. We used BLAST for the local alignment of DNA sequences and the standard dynamic programming algorithm for the global alignment of protein sequences. For KAS, we used $n = 3047$, $\Delta = 1000$. These parameters produce the same blocks as FS.

We extracted query sets from chr-18 dataset for $|q| = \{500, 1000, 2000, 4000\}$ each containing 100 queries. Later, we generated six new query sets from each of these query sets by modifying these queries with 5, 10, 20, 30, 40, and 50 % mutation probability. We generated two query sets from the protein dataset for $|q| = \{256, 512\}$. Later, we created three more query sets by modifying these queries with 5, 10 and 20 % mutation rates. We used BLAST's default scoring scheme for DNAs in our experiments. For protein dataset, we used BLOSUM62 score matrix.

We calculate the *accuracy* of the partial results for k-NN queries as the sum of the scores of the k best matches found so far divided by the sum of the scores of the actual k best matches.

Local alignment results: First, we inspect the performance of KAS and FS for local alignment of varying proximity of query sequences to database. We use the query sets from chr-18 in this experiment set. We generate a larger dataset, namely *chr-18/E.Coli* by appending two dissimilar datasets chr-18 and E.Coli, and perform queries on this dataset. The purpose of this experiment is to see whether our techniques can distinguish the distant regions in E.Coli from the homologous regions of chr-18. Figure 1 shows the average score found by FS, FS-prune, and KAS for 1-NN queries at different iterations for 5% mutation rate of $|q| = 4000$ query set. Since chr-18 and E.Coli datasets have approximately same number of base pairs, they have similar number of MBRs. As evident from this figure, all of our techniques find the optimal results before half of the database is inspected.

Table 3 summarizes the number of iterations performed to achieve accuracies of 0.75 and 0.9 for various mutation rates of the query set. For all mutation rates, all the techniques can obtain 0.9 accuracy after 25-31 %

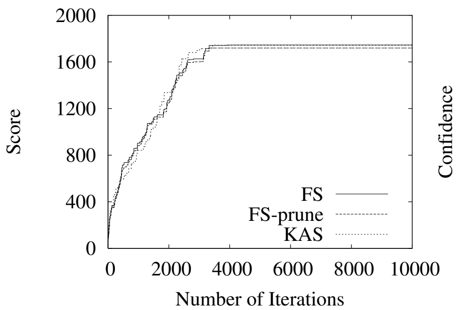

Figure 1: The score found at different iterations for query set generated from chr-18 (5 % mutation rate) on chr-18/E.Coli dataset.

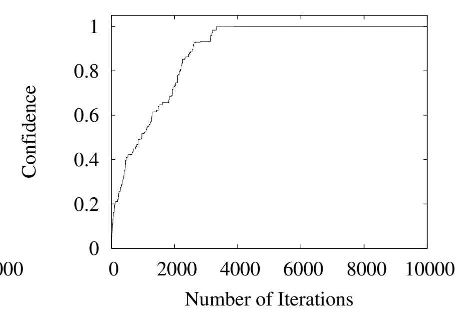

Figure 2: The confidence of FS for 1-NN queries for the experiment in Figure 1.

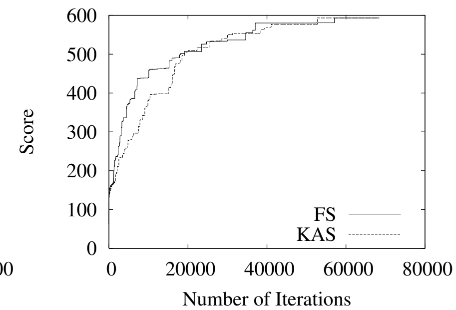

Figure 3: The score found on protein dataset for $|q| = 256$, and 20 % mutation rate.

Table 3: The number of iterations to achieve 0.75 and 0.9 accuracy for various mutation rates. The number of iterations for non-progressive techniques is 8863. (i.e., the number of database blocks.) The results for accuracy = 0.9 are shown in parenthesis.

Mutation Ratio	5	10	20	30	40	50
FS	2095 (2588)	2123 (2719)	1866 (2739)	1771 (2537)	1903 (2758)	1766 (2917)
FS- prune	2096 (2560)	2140 (2719)	1866 (2739)	1771 (2537)	1903 (2758)	1766 (2917)
KAS	1846 (2401)	1613 (2347)	1759 (2431)	1362 (2236)	1583 (2834)	1857 (2811)

of the MBRs are inspected, and 0.75 accuracy after 15-24 % of the MBRs are inspected. Although there is no clear winner among these three techniques, KAS finds high scoring results slightly faster on the average. FS and FS-prune overlap for mutation rates greater than 10 %.

Evaluation of pruning rate: Unlike FS-prune, both FS and KAS search the entire database to ensure that there is no remaining match better than the reported ones. FS-prune reduces total search time by pruning low scoring regions of the database. In the experiments in Table 3, we achieved 38.9 %, 17.3 %, and 12.1 % pruning for mutation rates 5 %, 10 %, and 20 % respectively. For larger mutation rates, the pruning went down to zero.

For 1, 2, 4, and 8-NN, FS-prune eliminated 38.9 %, 38.9 %, 37.5, and 17.5 % of the database respectively. The amount of database pruned reduces slowly as the number of NN increases. However, even when $k = 8$, FS-prune is 21 % faster than both FS and KAS.

Evaluation of confidence estimates Figure 2 shows how the confidence of FS changes over iterations for the experiment in Figure 1. FS achieves 90% confidence after performing only 30% of the iterations. Another important point is that the confidence of FS corresponds closely to its accuracy in Figure 1. That is, FS can accurately report the user the quality of the partial results with respect to the final results before all the final alignments are found. This experimentally substantiates the theoretical developments of Sections 4.2 and 4.3. This is important, because the user can estimate the score of the best matches of the final alignment at intermediate steps by inspecting the partial results and the confidence estimates.

Global alignment results: In this experiment, we evaluate the quality and performance of FS, FS-prune, and KAS for global alignment of protein sequences. We use the standard dynamic programming method with BLOSUM62 score matrix in this experiment.

Figure 3 plots the average score found during various iterations for the query set $|q| = 256$ with 20 % modification. We do not plot the results for other query lengths and mutation rates since they have similar behavior. The experiments show that FS is slightly better than KAS. In this experiment, FS achieved 75 % accuracy after only 15 % of the data is processed. On the other hand KAS processed 24 % of the dataset to achieve the same quality. However, KAS catches up with FS as accuracy increases to 90 %. In this case FS searches 36 % of the data while KAS searches 37 %.

6 Discussion

In this paper, we considered progressive subsequence searching for biological sequences. We defined a new statistical model, based on frequency vectors, for the analysis of the distance distribution between a query and a set of sequences based on their frequency vectors. We presented formulas to compute this distribution in $O(\gamma)$ time, where γ is the alphabet size.

We proposed two novel progressive subsequence search techniques called *KAS* and *FS*. KAS splits the database into overlapping blocks and computes the average frequency of all the letters in each block. It employs Karlin-Altschul statistics to predict the maximal local alignment score for each block, and ranks the blocks based on this prediction. FS organizes the database as a set of MBRs with the help of the MRS index structure. The MBRs of the index structure are reordered based on the order statistics of their representative frequency vectors at various resolutions. These MBRs are then searched iteratively using any traditional sequence search tool. The partial results are reported at each iteration along with their confidence estimates. We also reduced the total run time of FS by pruning the unpromising MBRs. This technique is called *FS-prune*.

Our experimental results showed that the proposed techniques achieve high accuracies quickly. In our experiments, our techniques achieved 75 % accuracy within the first 2.5-31 % of the iterations and 90 % accuracy within the first 12-37 % of the iterations. In our experiments, KAS performed slightly better for the local alignment queries and FS performed better for the global alignment queries. Our pruning strategy eliminated up to 38 % of the database in these experiments. The confidence value computed by FS corresponded closely to its accuracy.

References

[AG96] S.F. Altschul and W. Gish. Local alignment statistics. *Methods in Enzymology*, pages 460–480, 1996.

[AGM$^+$90] S. Altschul, W. Gish, W. Miller, E. W. Meyers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[AW94] R. Arratia and M.S. Waterman. A phase transition for the score in matching random sequences allowing deletions. *The Annals of Applied Probability*, 4(1):200–225, 1994.

[Cas88] E. Castillo. *Extreme Value Theory in Engineering (Statistical Modeling and Decision Science Series)*. Academic Press, New York, 1988.

[Ham87] R. Hamming. *Numerical Methods for Scientists and Engineers*. Dover Pubns, 2 edition, April 1987.

[KA90] S. Karlin and S.F. Altschul. Methods for Assessing the Statistical Significance of Molecular Sequence Features by Using General Scoring Schemes. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 87(6):2264–2268, 1990.

[KA93] S. Karlin and S.F. Altschul. Applications and Statistics for Multiple High-Scoring Segments in Molecular Sequences. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 90(12):5873–5877, 1993.

[KLS04] T. Kahveci, V. Ljosa, and A.K. Singh. Speeding up whole-genome alignment by indexing frequency vectors. *Bioinformatics*, 2004. to appear.

[KS01] T. Kahveci and A. Singh. An Efficient Index Structure for String Databases. In *International Conference on Very Large Databases (VLDB)*, pages 351–360, Roma, Italy, September 2001.

[MPK03] C. Meek, J. M. Patel, and S. Kasetty. OASIS: An Online and Accurate Technique for Local-alignment Searches on Biological Sequences. In *International Conference on Very Large Databases (VLDB)*, 2003.

[NW70] S. B. Needleman and C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48:443–53, 1970.

[SW81] T.F. Smith and M.S. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, March 1981.

Novel Approaches to Biomolecular Sequence Indexing *

Emre Karakoc
Simon Fraser University
ekarakoc@cs.sfu.ca

Z. Meral Ozsoyoglu
Case Western Reserve University
ozsoy@eecs.cwru.edu

S. Cenk Sahinalp
Simon Fraser University
cenk@cs.sfu.ca

Murat Tasan
Case Western Reserve University
tasan@eecs.cwru.edu

Xiang Zhang
Simon Fraser University
xzhangi@cs.sfu.ca

Abstract

In many biomolecular database applications involving string/sequence data, it is common to have similarity search in the form of near neighbor queries or nearest neighbor queries. The similarity between strings/sequences are typically measured in terms of the least costly set of allowed edit operations that transform one string/sequence to another. In this survey, we briefly describe some of the recent developments in biomolecular sequence indexing methods that allow efficient similarity search. Our focus here is on global similarity measures that compare sequences in full; such measures are important for comparing protein sequences and smaller biomolecules. Examples include character and block edit distances and their weighted variants. Two major approaches are summarized here: distance based indexing and embeddings of general sequence similarity measures to Hamming distance, for which efficient indexing methods are available.

1 Introduction

The advent of efficient DNA sequencing techniques have lead to exponential growth in biomolecular sequence data. With data growth levels surpassing Moore's law, it has become essential to develop highly efficient data structures and indexing tools for string/sequence similarity search [NCBI].

Efficient similarity search is key to handling/processing massive biomolecular sequence data as sequence similarity often implies functional and evolutionary relationship. Similarity between sequences are usually defined in terms of the distance function in use. In this survey we focus on *global* similarity measures between sequences/strings. The best studied global distance measures are *character edit distance* [Lev66] and *block edit distance* [CPSV00, MS00] (also known as the transformation distance [VDR99]), as well as their weighted variants.

Given a distance function, one can search for sequences similar to a query sequence in the form of two commonly used query types: (i) k-nearest neighbor queries ask for the k "most" similar sequences (i.e. k

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Names of the authors are listed alphabetically sequences with the smallest distance) to a query sequence; (ii) range queries ask for all sequences that have "sufficient" similarity (i.e. the ones which have distance at most some user defined ℓ) to the query sequence.

In this survey we cover two recently developed indexing strategies that enable efficient sequence similarity search.

The first strategy is the use of *distance based indexing* methods [STMO03]. Most general indexing methods perform similarity search by iteratively pruning subsets of potential answers via (i) partitioning the data set (into overlapping or non-overlapping) subsets in the preprocessing stage, and (ii) checking out to which partition(s) the query belongs during the pruning stage. A distance based indexing method performs partitioning and pruning based (only) on distances between the data items. No other information about the data items are used.

Distance based indexing methods were originally designed for arbitrary metric distances. We describe how these methods could be used for string/sequence distance measures, provided that they form a metric or an *almost metric*. (Many sequence distances including the character edit distance and the block edit distance form metrics whereas others such as the *compression distance* and many weighted versions of the character edit distance are almost metrics.)

In this survey we focus on the Vantage Point (VP) tree for illustrating how distance based indexing methods can be used for our purposes. In order to analyze the performance of VP trees we describe a data model based on distribution of distances between sequence pairs. With the help of this model we describe how to modify/tune VP trees to obtain the best performance guarantees on sequence/string data of interest, while providing tradeoffs between search time and space.

Although distance based indexing methods perform well for many data sets of practical importance, they suffer from the "curse of dimensionality"; i.e. in the worst case, they either have query time proportional to the data set size or preprocessing time exponential with the data set size. In fact the only known data structures that provide *desirable* worst case performance (preprocessing time polynomial with the data set size, query time polynomial with the query size) work only for *Hamming distance* [KOR98, IM98] (or some of its weighted variants such as the L_1 distance). Although these data structures do not guarantee exact answers to queries, they provide good approximate answers; e.g. in nearest neighbor search, they guarantee to return a data item whose distance to the query item is within $1 + \epsilon$ factor of the distance between the query and its nearest neighbor.

A new approach to sequence similarity search under several measures of interest such as the block edit distance is to embed the distance into the Hamming distance. These embeddings are distance preserving; i.e. they map each sequence to a binary vector such that the Hamming distance between any two such binary vectors approximate the block edit distance between the original sequences. After the embedding is performed one can use any one of the efficient indexing techniques for the Hamming distance. The second half of our survey is thus dedicated to embedding distances of interest to Hamming distance. We summarize positive results as well as some lower bounds that imply certain limitations of this general approach, especially in the context of character edit distances.

Notation. Throughout the paper s, q, r, t denote strings, i.e. contiguous character sequences from an arbitrary alphabet (denoting nucleotides in a DNA/RNA molecule or amino acids in a protein); $s[i]$ denotes the i^{th} character of string s and $s[i : j]$ the substring between the i^{th} and j^{th} characters of s. The length of the string s is denoted by $|s|$.

2 Commonly Used Similarity Measures between Sequences/Strings

Similarity between a pair strings s and r are typically measured via edit operations that *transform* one string into the other. Possible edit operations that involve single characters are *character insertion*,*character deletion* and *character replacement*. One may also consider block (i.e. substring) edit operations such as *block copying*, *block deletion*, and *block relocation*. Each of these edit operations may have a specific cost; thus given a transformation from r to s, a *distance* $d(r \rightarrow s)$ from r to s can be defined as the minimum total cost of edit operations to transform r to s. Such a distance is not necessarily symmetric, i.e., there may be strings s and r for which $d(r \rightarrow s) \neq d(s \rightarrow r)$.

Possibly the simplest distance measure between two (equal length) strings s and r is the Hamming distance $H(s, r)$ which is defined as is the number locations i such that $s[i] \neq r[i]$. A more common string similarity measure is the character edit distance, which is also referred as the Levenshtein edit distance or simply the edit distance [Lev66]. It can be defined as the minimum number of single character insertions, deletions and replacements needed to transform one string into another. Weighted versions of the character edit distance assign costs to specific operations to specific characters and these measures try to find the minimum cost operations for transforming one string to the other one (as in the case of PAM and BLOSSUM substitution tables [DSO78, HH92]).

More recently block edit operations and distances based on these operations have received considerable attention especially in the context of evolutionary analysis of world languages and mitochondrial DNA sequences of various species (see, for example, the review in Nature [Ball02]). Given two strings, their *block edit distance* [MS00] (a.k.a. *transformation distance* [VDR99]) is defined to be the minimum number of block relocations copies and deletions as well as single character insertions, deletions and replacements to transform one string to another.

Because of its generality, the block edit distance provides a lower bound to any distance based on character and block edits; however it is NP-hard to compute. A more limited distance based on block edits is the compression distance (which received recent attention [LBXKKZ01, BCL02, LCLMV03]) that can be defined in terms of the total number of phrases returned by the Lempel-Ziv-77 data compression method when compressing each one of the strings while using the other as a static dictionary. It was shown recently that the compression distance tightly approximates the block edit distance [EMS03, STMO03]. By the use of suffix trees the compression distance as defined here can be computed in time linear with the total lengths of the strings [RPE81].

3 Sequence Similarity Search via Distance Based Indexing

The general sequence similarity search problem can be defined as follows. Given a set of sequences $X = \{x_1, ..., x_n\}$, a distance function d, a search radius R, and a query sequence q, (1) retrieve all sequences that are within distance R to the query sequence - this is called near neighbor search, or (2) retrieve the sequence that has the smallest distance to the query sequence - this is called nearest neighbor search. One can also ask for the k-nearest neighbors of q for $k > 1$.

A recent approach to the sequence similarity search problem is through the use of distance based indexing methods. In distance based indexing, pairwise distances are used to iteratively partition the space into smaller subspaces; search is performed through pruning potential answers by limiting search into subspaces that progressively get smaller [Uhlmann91, Yianilos93, BO97, CPZ97]. Distance based indexing methods have been described for arbitrary metric distances d; i.e. those distances that are symmetric ($d(x, y) = d(y, x)$), reflexive ($d(x, y) \geq 0$ and $d(x, y) = 0$ iff $x = y$) and satisfy the triangle inequality ($d(x, y) + d(y, z) \geq d(x, z)$).

In this survey we illustrate the use of distance based indexing through one specific method, the Vantage Point (VP) trees [Uhlmann91]. In its standard form, a vantage point tree is a binary tree that recursively partitions a data set into two subsets. Each internal node is of the form $(x_v, M, Rptr, Lptr)$ where x_v is the (possibly randomly selected) vantage point, M is the median distance among the distances of all the points(from x_v) indexed below that node, and $Rptr$ and $Lptr$ are pointers to the right and left branches. Left branch of the node indexes the points whose distances from x_v are at most M, and right branch of the node indexes the points whose distances from x_v are at least M. Leaf nodes simply consist single data points.

Given a non-empty set $X = \{x_1, ..., x_n\}$ and a metric distance $d(x_i, x_j)$, a binary VP tree can be constructed as follows. Let x_v be an arbitrary element from X. Also let M be the median of $\{d(x_i, x_v) | \forall x_i \in X\}$; let $X_l = \{x_i | d(x_i, x_v) \leq M, x_i \in X, x_i \neq X_v\}$ and $X_r = \{x_i | d(x_i, x_v) \geq M, x_i \in X\}$. Recursively create VP tree on X_l and on X_r as the left and right branches of the root. This construction can be done by performing $O(nlogn)$ pairwise distance computations.

For a given query item q, the set of data items that are within distance R of q are found using the following search routine. If x_v is the single data point in X and $d(q, x_v) \leq R$, then x_v is in the answer set. Else, if $d(q, x_v) + R \geq M$, then recursively search the right branch. If $d(q, x_v) - R \leq M$, then recursively search the left branch. If both conditions apply, both branches must be searched. The correctness of this routine follows from the triangle inequality satisfied by d.

Although both character edit distance and block edit distance are metric distances, their weighted variants and the compression distance are not. Fortunately, these measures have been shown to be *almost metrics* (or near metrics) [STMO03]. A distance function f is an almost metric for space S if it is symmetric, reflexive and satisfies the triangular inequality within a constant factor K; i.e. for all $s, r, q \in S, f(s, r) \leq K \cdot [f(s, q) + f(q, r)]$.

It is possible to use VP trees for almost metrics via the following update on the search strategy [STMO03]. Let d be an almost metric distance which satisfies the triangular inequality within a factor of K. Now let q be the query item, R be the query range, x_v be the vantage point accessed during the search, and M be the median distance value for x_v. If $d(x_v, q)+R < M/K$ then we can prune the right branch. Also, if $d(x_v, q)/K-R > M$ then we can prune the left branch. If neither of the conditions are satisfied, both branches must be searched.

3.1 Modifying VP-Trees for Specific Data Distributions

It is easy to verify that the worst case performance of VP tree search could be comparable to the brute force search. In fact, it has been demonstrated for high dimensional spaces that when the data points are distributed uniformly over search space, the performance of *any* indexing method becomes comparable to brute force search [BK98]. For many data sets of practical importance, however, VP trees seem to work quite well. In an attempt to understand the conditions under which VP trees work efficiently, we focus on specific distributions of pairwise distances which are common to genomic and protein sequences. We then describe several modifications to the VP trees which have provably good expected performance under such distributions [STMO03].

In the analysis below the following is assumed: (1) the distribution of the distances between a “typical” data point to other points in the data set is similar to the overall pairwise distance distribution, (2) the distribution of the query points in the input space is similar to that of the data points. Under these assumptions we will consider two types of distributions, (i) *exponential* and (ii) *power-law*, and analyze the performance of the modified VP trees for nearest neighbor queries.

Nearest neighbor search in exponentially distributed data. Let a data set D contain m strings. Given a typical query point q, we say that $f_q(R)$, the number of points observed at distance $\leq R$ is exponentially distributed if $f_q(R) \sim k \cdot c^R$ for some c and k.

Denote by $nn_h(q)$ the h-nearest neighbor of the query point q. By definition, $f_q(d(q, nn_1(q))) \sim 1$ and thus $d(q, nn_1(q)) \sim log_c 1/k$. Thus, when one is searching for the nearest neighbor of a query point q, one is looking for retrieving all the data points whose distance to q is $d(q, nn_1(q)) \sim log_c 1/k$ - this is by assumption (1) above.

Let p be the topmost vantage point in the VP tree built for D. It is possible to compute the distance between p and its m/l'th nearest neighbor for some constant l: $f_p(d(p, nn_{m/l}(p))) \sim m/l$ and thus $d(p, nn_{m/l}(p)) \sim log_c m/kl$. The number of points that are within distance $d(p, nn_{m/l}(p)) + d(p, nn_1(p))$ from p are thus
$f_p(d(p, nn_{m/l}(p)) + d(p, nn_1(p))) \sim k \cdot c^{log_c 1/k} \cdot c^{log_c m/kl} \sim m/k \cdot l$.

The VP tree is constructed so that each time a vantage point p for a subset is determined, it partitions the data set into two: (1) inner partition include the nearest m/kl points to p and (2) outer partition includes the remaining points. In the standard VP tree, the cardinality of the inner and the outer partitions are equal (they are separated by the median point) which implies $1/kl = 1/2$.

Search for the nearest neighbor of a query item q will be performed as follows. For each vantage point p encountered one of the following cases will apply.
(i) If $d(p,q) \leq d(p, nn_{m/l}(p)) \sim log_c m/kl$ then the outer partition will be eliminated and the search will be iteratively performed on the inner partition. According to assumption (2) above, the probability of this case is $1/l$.
(ii) If $d(p,q) > log_c m/k^3 l$ then the inner partition will be eliminated and the search will be performed iteratively on the outer partition. The probability of this case is $1 - 1/lk^2$.
(iii) Otherwise both the inner and outer partitions will be searched.

The probability of case (ii) is non-zero only if $k > 1/2$ which is very atypical (see the experimental results in the next section). Thus we will ignore case (ii)and obtain a recursion for query time focusing on cases (i) and (iii) only; i.e. we will assume that outer partition is often "thin" and will offer little additional pruning at the lower levels of the search tree. Let $T(m)$ be the nearest neighbor search time for q among m data points.
Then $T(m) \leq 1 + 2k \cdot T(m/2) + (1-2k) \cdot 2 \cdot T(m/2)$.
This recursion has a solution at $T(m) \leq m^{log 2 - k/2}$.

Although the above analysis reveals that the worse case does not improve brute force search, it is possible to improve the performance by the following *modification* to the standard VP tree construction. In this updated VP tree, rather than having a single vantage point at a given node, we have multiple vantage points. When one visits a node during search, if the first vantage point fails to satisfy case (i) another vantage point may be considered. If the number of vantage points at each node is set to $j \cdot l$ (where j is a constant) the running time of a query $T(m)$ will be $T(m) = O(2/k \cdot m^{log 1 + 1/e^j})$. This will be much faster than the brute force search if j is chosen to be sufficiently large.

The increase in the number of vantage points per node clearly increases the space complexity of the resulting data structure. For $j = 1$, there will be $2/k$ vantage points in level 1; in level i there will be $(2/k)^i$ vantage points. Because the number of levels is $\log m$, the overall space complexity becomes $O((2/k)^{\log m}) = O(m^{1-\log k})$; this is a small polynomial of m for data sets encountered in relevant applications.

Nearest neighbor search under power-law distance distribution. Given a query item q we say that the number of data items observed at distance $\leq R$ have power-law distribution if $f_q(R) \sim k \cdot R^c$. By definition, $f_q(d(q, nn_1(q))) \sim 1$ and thus $d(q, nn_1(q)) \sim (1/k)^{1/c}$. Similarly given a vantage point p, it is possible to compute the distance between p and its m/l'th nearest neighbor for some constant l: $f_p(d(p, nn_{m/l}(p))) \sim m/l$ and thus $d(p, nn_{m/l}(p)) \sim (m/lk)^{1/c}$.
It is easy to verify that the number of points that are within distance $d(p, nn_{m/l}(p)) + d(p, nn_1(p))$ from p is approximately m/l. In the standard VP tree, the cardinality of the inner and outer partitions are equal and thus $l = 2$. We can write the recurrence relation for the nearest neighbor search time $T(m)$ as $T(m) = 1 + 3/2 \cdot T(m/2)$ which has a solution at $T(m) = O(m^{0.58})$.

Although the above analysis reveals that the worst case running time for the nearest neighbor search is better than the brute force search, it is possible to improve the performance by a modification similar to that applied to the exponential distribution: i.e. there will be as many vantage points at each node as allowed by the space complexity.

Let the number of vantage points at each node be j; the reader can verify that the running time of a query $T(m)$ will be $T(m) = O(2 \cdot m^{log(1+1/2^j)})$. There are j vantage points for each node and the number of the levels in the VP tree is $O(\log m)$; thus the space complexity of the modified VP tree will be $O(j^{\log m}) = O(m^{\log j})$. This modification will have much better search performance if j is chosen to be sufficiently large. For example for $j = 4$ one can achieve a search time of $O(m^{1/11})$, which will be a very small figure for all practical data sets; the space complexity will be only $O(m^2)$.

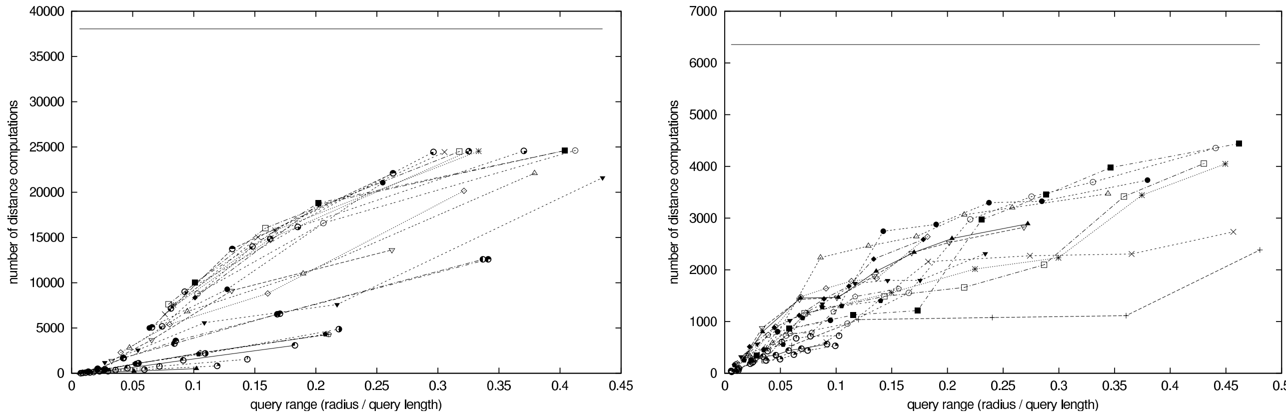

Figure 1: Human (left) and a yeast (right) proteomes indexed using a standard binary VP-Tree. Each line indicates the query results (in number of comparisons) for a query sequence (picked from the data set itself) with increasing query radius (as a percentage of the query length itself). Note that the search radii will typically be within 25% of the query length.

3.2 Some Experimental Results

We report on two set of experimental results on protein sequences. The first data set involves all active and potential proteins derived from the complete human genome sequence. The second data set involves all proteins from the yeast (S.Cerevisiae) genome. Both data sets are exponentially distributed under the character edit distance (typical values are $k < 1/4$ and $c \sim 2^{1/400}$) [STMO03]. The pruning results of standard VP tree searches with varying search radii (together with the "brute-force" search) are demonstrated in Figure 1. For most query sequences, as the radius for near neighbor search increases, the number of distance computations (and thus the running time) increases linearly.

4 Sequence Similarity Search via Embeddings

As demonstrated above, distance based indexing methods usually have good performance for practical string data sets; however, in the worst case they have suffer from the curse of dimensionality. In fact for no distance measure that allows non-trivial edit operations is known to lead to an efficient data structure that provides a desirable worst case performance guarantee; i.e. for all distance measures, all known data structures require either preprocessing time exponential with the number of data items or a query time comparable to brute force search.

For the case of Hamming distance, however, there are a number of data structures that provide desirable worst case performance guarantees for nearest neighbor search [KOR98, IM98]. (The guarantees are valid only if a small $(1 + \epsilon)$ factor of approximation can be tolerated in the answers provided; i.e. the answer to a nearest neighbor query will not be exact but will be within $(1 + \epsilon)$ factor of the distance between the query and its true nearest neighbor.) Such data structures work by dimensionality reduction, space partitioning and bucketing. Unfortunately none of these methods seems to work with distance functions involving non-trivial edit operations. However it may still be possible to utilize an efficient data structure that work under Hamming Distance for other distance functions, provided such distances could be "embedded" into the Hamming distance with small distortion. In this section, we will show that such embeddings exist, in particular from the block edit distance [CPSV00, MS00].

The embedding of the block edit distance into the Hamming distance involves hierarchically parsing a given string into 'core' substrings [CPSV00, MS00]. Given an alphabet, the complete list of core substrings of varying size is known. To embed a string into a binary vector, one needs to prepare a bit vector whose length is the total number of possible cores of relevant size. The i^{th} bit of this vector is set to 1 only if the lexicographically i^{th} largest core (for the specific alphabet) is present in the string.

The computation of core substrings is performed through the use of *Locally Consistent Parsing* (LCP), first described for optimal parallel construction of a suffix tree [SV94]. A symmetric variant of LCP that allows block rotations was later described in [MS00]. LCP uses the local composition of a string for partitioning it into (possibly overlapping) core substrings of roughly equal size. Each core substring can be replaced by a fingerprint to have a shorter representation of the string. On this short representation, LCP can be applied iteratively until it is shrunk to a constant size. Because the core substrings are extracted independent of their location in the string, the core substring composition of a long block does not change even if it is moved within the string.

Suppose that the embedding of two strings s and r are the binary vectors $T(s)$ and $T(r)$ respectively. Because the core substring composition of a string is mostly preserved after a block operation, $T(s)$ and $T(r)$ guarantee that their Hamming distance is a $O(\log l \log^* l)$ approximation of the block edit distance between s and r ($l = |s| + |r|$). Although the size of a vector $T()$ will be $O(2^l)$, there will be at most l nonzero entries. Such a vector can be represented by using $O(l^2)$ bits.

The nature of SNN problem depends on the distance function used for determining the similarity between two strings. Although block edit distance can be embedded into the Hamming Distance quite efficiently, no such embedding is known for the character edit distance or any of its variants. In fact, a recent result [ADGIR03] demonstrates that an embedding from character edit distance to Hamming distance can not be achieved with an approximation factor better than $3/2$. Other limitations of the embedding approach is described in [SU04].

5 Conclusions

The recent increase in the amount of sequence data in biomolecular databases bring many challenges to the sequence similarity search problem. Here we survey two novel approaches for performing global sequence similarity search: (i) distance based indexing and (ii) similarity search via embeddings. The first approach is quite a general one applicable to all distance measures that form a metric or an almost metric. The performance is, however, dependent on the specific pairwise distribution observed in the data set. In fact, the worst case performance of this approach could be comparable to the brute force search.

For Hamming distance and a number of its variants that do not allow any non-trivial edit operations, a number of data structures with polynomial worst case performance guarantees have been recently developed. The second approach surveyed here aims to embed an arbitrary distance measure to Hamming distance via the use of a distance preserving transformation. One such embedding for Block Edit distance with relatively small distortion is summarized in this survey. A major open problem is whether such embeddings could be obtained for character edit distances.

References

[ADGIR03] A. Andoni, M. Deza, A. Gupta, P. Indyk, S. Raskhodnikova. Lower Bounds for Embedding Edit Distance into Normed Spaces. In *Symposium on Discrete Algorithms*, pages 523-526, 2003.

[Ball02] Philip Ball. Algorithm makes tongue tree. *Nature*, Science update, Jan 22, 2002.

[BCL02] D. Benedetto, E. Caglioti, V. Lorento. Language Trees and Zipping. *Physical Review Letters*, 88(4), Jan 2002.

[BK98] S. Berchtold, D.A. Keim. High-dimensional Index Structure. In *Proc. ACM SIGMOD*, page 501, 1998.

[BO97] T. Bozkaya and M. Ozsoyoglu. Distance-Based Indexing for High-Dimensional Metric Spaces. *Proc. SIGMOD*, pages 357–368, 1997.

[Brin95] S. Brin. Near Neighbor Search in Large Metric Spaces. In *Proc. VLDB*, pages 574–584, 1995.

[BK73] W. A. Burkhard, and R. M. Keller. Some Approaches to Best-Match File Searching. *Communications of the ACM*, 16(4), pages 230-236, April 1973.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-Trees: An efficient access method for similarity search in metric space. In *Proc. VLDB*, pages 426–435, 1997.

[CPSV00] Graham Cormode, Mike Paterson, Suleyman Cenk Sahinalp, Uzi Vishkin. Communication complexity of document exchange. In *Symposium on Discrete Algorithms*, pages 197-206, 2000.

[DSO78] M. Dayhoff, R. Schwartz, B. Orcutt. A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*, Volume 5, pages 345–352, 1978.

[EMS03] Funda Ergun, S. Muthukrishnan, Suleyman Cenk Sahinalp. Comparing Sequences with Segment Rearrangements. In *Foundations of Software Technology and Theoretical Computer Science*, pages 183-194, 2003.

[HH92] S. Henikoff, J.G. Henikoff. Amino acid substitution matrices from protein blocks. In *Proceedings of the National Academy of Sciences*, Volume 89, pages 10915–10919, 1992.

[IM98] P.Indyk, R.Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proc. ACM -SIAM Symposium on Theory of Computing*, pages 604–613, 1998.

[KOR98] E. Kushilevitz, R. Ostrovsky, Y.Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proc. ACM -SIAM Symposium on Theory of Computing*, pages 614–623, 1998.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletins, insertions and reversals. *Cybernetics and Control Theory*, 10(8): pages 707–710, 1966.

[LBXKKZ01] M.Li, J. H. Badger, C. Xin, S. Kwong, P. Kearney, H. Zhang. An information based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17, 2001.

[LCLMV03] M. Li, X. Chen, X. Li, B. Ma, P. Vitanyi. The similarity Metric. *Proc. ACM-SIAM SODA*, Baltimore MD, 2003.

[MS00] S. Muthukrishnan and S. C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. *Proc. ACM Symposium on Theory of Computing*, 2000.

[NCBI] NCBI Genbank Statistics. http://www.ncbi.nih.gov/Genbank/genbankstats.html.

[RPE81] M. Rodeh, V. R. Pratt, S. Even. Linear Algorithm for Data Compression via String Matching. *JACM*, 28(1): pages 16–24, 1981.

[STMO03] S.Cenk Sahinalp, Murat Tasan, Jai Macker, Z.Meral Ozsoyoglu. Distance-Based Indexing for String Proximity Search. In *IEEE Data Engineering Conference*, 2003.

[SU04] Suleyman Cenk Sahinalp, Andrey Utis. Hardness of String Similarity Search and Other Indexing Problems. In *International Colloquium on Automata, Languages and Programming*, pages 1080-1098, 2004.

[SV96] S.Cenk Sahinalp, U. Vishkin. Approximate and Dynamic Matching of Patterns Using A Labeling Paradigm. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 320–328, 1996.

[SV94] S.Cenk Sahinalp, U. Vishkin. Symmetry breaking for suffix tree construction. In *ACM Symposium on Theory of Computing*, pages 300-309, 1994.

[Uhlmann91] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *IPL*, (4): pages 175–179, 1991.

[VDR99] J. S. Varre, J. P. Delahaye, and E. Rivals. The Transformation Distance: A Dissimilarity Measure BVased on Movements of Segments. In *Bioinformatics*, 15:3, pages 194–202, 1999.

[Yianilos93] P. N. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.

The 21st International Conference on

Data Engineering (ICDE 2005)

April 3-9, 2005
National Center of Sciences, Tokyo, Japan
Sponsored by
The IEEE Computer Society
The Database Society of Japan (DBSJ)
Information Processing Society of Japan (IPSJ)
The Institute of Electronics, Information and Communication Engineers (IEICE)

http://icde2005.is.tsukuba.ac.jp/ http://icde2005.naist.jp/ (mirror)

Welcome to ICDE 2005 Tokyo

Data Engineering deals with the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments. The 21st International Conference on Data Engineering (ICDE 2005) provides a premier forum for:

- sharing research solutions to problems of today's information society
- exposing practicing engineers to evolving research, tools, and practices and providing them with an early opportunity to evaluate these
- raising awareness in the research community of the problems of practical applications of data engineering
- promoting the exchange of data engineering technologies and experience among researchers and practicing engineers
- identifying new issues and directions for future research and development work.

The conference will be held in Tokyo, the capital city of Japan. Tokyo is the most populous metropolitan area in the world. Yet despite its complex urban landscape and impressive architecture, this city abounds with parks and gardens. Particularly in the spring season, blooming *cherry blossoms* will welcome you to Tokyo.

Advanced Technology Seminars

- XQuery Midflight: Emerging Database-Oriented Paradigms and a Classification of Research Advances (I. Manolescu and Y. Papakonstantinou)
- Data Stream Query Processing (N. Koudas and D. Srivastava)
- Web Service Coordination and Emerging Standards (F. Casati and G. Alonso)
- Online Mining Data Streams: Problems, Applications, Techniques and Progress (H. Wang, J. Pei, and P. Yu)
- Rank-Aware Query Processing and Optimization (I. Ilyas and W. Aref)
- Data Mining Techniques for Microarray Datasets (L. Liu, J. Yang, and A. Tung)

Workshops

- International Workshop on Ubiquitous Data Management (UDM 2005), Apr. 4, 2005. Deadline: Nov. 15, 2004.
- International Workshop on Biomedical Data Engineering (BMDE 2005), Apr. 3-4, 2005. Deadline: Nov. 15, 2004.
- International Workshop on Challenges in Web Information Retrieval and Integration (WIRI 2005), Apr. 8-9, 2005. Deadline: Nov. 15, 2004.
- International Workshop on Privacy Data Management (PDM 2005), Apr. 8-9, 2005. Deadline: Nov. 17, 2004.
- International Workshop on Autonomic Database Systems, Apr. 8-9, 2005. Deadline: Nov. 15, 2004.
- International Workshop on Realworld Multimedia Corpora in Mobile Environment (RWCinME 2005), Apr. 8-9, 2005. Deadline: Nov. 15, 2004.
- 2nd International Workshop on XML Schema and Data Management (XSDM 2005), Apr. 8-9, 2005. Abstract Deadline: Oct. 15, 2004. Paper Deadline: Oct. 22, 2004.
- International Workshop on Data Engineering Issues in E-Commerce (DEEC 2005), Apr. 9, 2005. Deadline: Nov. 22, 2004.
- International Workshop on Managing Data for Emerging Multimedia Applications (EMMA 2005), Apr. 8-9, 2005. Deadline: Nov. 15, 2004.

Keynote Speakers (other speaker to be announced)

- Pat Selinger (IBM, USA)
- Mike Stonebraker (MIT, USA)

Conference Officers

Honorary Chair:	The Late Yahiko Kambayashi (Kyoto University, Japan)
Organizing Committee Chair:	Yoshifumi Masunaga (Ochanomizu University, Japan)
General Chairs:	Rakesh Agrawal (IBM Almaden Research Center, USA)
	Masaru Kitsuregawa (University of Tokyo, Japan)
Program Chairs:	Karl Aberer (EPF Lausanne, Switzerland)
	Michael Franklin (UC Berkeley, USA)
	Shojiro Nishio (Osaka University, Japan)
PC Area Chairs:	Anastassia Ailamaki (Carnegie Mellon University, USA)
	Gustavo Alonso (ETH Zurich, Switzerland)
	Phillip Gibbons (Intel Research, USA)
	Takahiro Hara (Osaka University, Japan)
	Jayant R. Haritsa (IISc Bangalore, India)
	Alfons Kemper (University of Passau, Germany)
	Sharad Mehrotra (UC Irvine, USA)
	Wolfgang Nejdl (University of Hannover, Germany)
	Jignesh M. Patel (University of Michigan, USA)
	Evaggelia Pitoura (University of Ioannina, Greece)
	Jayavel Shanmugasundaram (Cornell University, USA)
	Kyuseok Shim (Seoul National University, Korea)
	Kian-Lee Tan (National University of Singapore, Singapore)
Executive Committee Chair:	Jun Adachi (National Institute of Informatics, Japan)
Panel Chairs:	Umeshwar Dayal (HP Labs, USA)
	Hongjun Lu (HKUST, China)
	Hans-Jörg Schek (UMIT, Austria/ETH Zurich, Switzerland)
Seminar Chairs:	Michael J. Carey (BEA Systems, USA)
	Stefano Ceri (Politecnico di Milano, Italy)
	Kyu-Young Whang (KAIST, Korea)
Demo Chairs:	Daniel Keim (University of Konstanz, Germany)
	Ling Liu (Georgia Institute of Technology, USA)
	Xiaofang Zhou (University of Queensland, Australia)
Industrial Chairs:	Anand Deshpande (Persistent Systems, India)
	Anant Jhingran (IBM Silicon Valley Lab, USA)
	Yasushi Kiyoki (Keio University, Japan)
	Eric Simon (Medience, France)
Local Arrangement Chair:	Haruo Yokota (Tokyo Institute of Technology, Japan)
Workshop Chair:	Masatoshi Yoshikawa (Nagoya University, Japan)
Proceedings Chair:	Motomichi Toyama (Keio University, Japan)
Publicity Chair:	Hiroyuki Kitagawa (University of Tsukuba, Japan)
DBSJ Liaison:	Hiroshi Ishikawa (Tokyo Metropolitan University, Japan)
CCF-DBS Liaison:	Xiaofeng Meng (Renmin University, China)
KISS SIGDB Liaison:	Chin-Wan Chung (KAIST, Korea)
DBIndia Liaison:	Krithi Ramamritham (IIT Bombay, India)
Australian DB Liaison:	James Bailey (University of Melbourne, Australia)
Treasurer:	Miyuki Nakano (University of Tokyo, Japan)

Related Conferences and Workshops

- 15th International Workshop on Research Issues on Data Engineering: Stream Data Mining and Applications (RIDE-SDMA), Apr. 3-4, Tokyo.
- The Seventh Asia Pacific Web Conference (APWeb 2005), Mar. 29-Apr. 1, 2005, Shanghai, China. http://apweb05.csm.vu.edu.au/
- 10th International Conference on Database Systems for Advanced Applications (DASFAA 2005), Apr. 18-20, Beijing, China. http://dasfaa05.cs.tsinghua.edu.cn/
- 4th International Workshop on Databases in Networked Information Systems (DNIS 2005), Mar. 28-30, Aizu, Japan. http://www.u-aizu.ac.jp/labs/sw-db/DNIS2005.html

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398