Distance Based Indexing and Similarity Search for Sequences

Z. Meral Ozsoyoglu
Computer Science, EECS Dept.
Case Western Reserve University
Cleveland, OH 44106

This talk is based on

Distance Based Indexing for String Proximity Search, *by C. Sahinalp, M. Tasan, J. Macker, Z.M. Ozsoyoglu,* ICDE 2003*, March 2003.*

Indexing for large Metric Spaces for Similarity Search Queries*, by T. Bozkaya, and Z.M. Ozsoyoglu,* ACM TODS, 1999*, Vol. 24, No. 3, pp. 361-404.*

Distance Based Indexing for High Dimensional Metric Spaces*, by T. Bozkaya and Z.M. Ozsoyoglu,* ACM SIGMOD 1997*, pp. 357-368.*

Overview

Goal: Efficient Tools for *Near Neighbor Queries for Sequences* using *character and block edit distances*

Approach: Use Distance Based Index Structures

Problem: common distance functions *not* metric

Contributions:

- compression distance and weighted edit distance are *almost metrics,*
- Modify distance based index structures (VP, MVP-trees) for almost metrics,
- Exploit the data distribution to improve performance
- Experimental results with synthetic and real data

Outline

- Sequence Similarity Measures
  - character and block edit distances--metric
  - weighted character edit distance, and compression distance—not metric *but almost metric*.
- Distance Based Index Structures
  - VP and MVP-trees
  - Using almost metric distance functions
- Performance and Experimental Results

Similarity Search for Sequences

Many Applications:

- Genetics
- Image recognition
- Video Compression
- Time series analysis/data mining
- Information retrieval
- etc.

Computational Genomics: Sequence Similarity among DNA Sequences

- Indicates evolutionary relationship →functional similarity
- Identifying mechanisms for genome rearrangements (duplications), key to understanding causes of a number of genetic diseases
- Multiple sequence alignment, evolutionary tree construction
- Assembly of shotgun sequenced genome

Key Issues

- Distance function
  - captures the notion of similarity
  - efficiently computable
  - enables efficient indexing methods
- Efficient access structure/indexing technique

Distance Functions: Character Edit Distances

Described as min. total cost of character edit operations (insertion, deletion, replacement) to transform one string to another.

- Levenshtein edit distance
- Weighted versions [Needleman, Wunsch 1971, Smith Waterman, 1981], PAM and Blossum scoring matrices

Quadratic time complexity, dynamic programming,

$|S|=n$, $|R|=m$, $S \rightarrow R$ is $O(n.m)$

Block Edit distances

- Minimum no. of block edit operations *(block copying, block deletion, block relocation),* and single character operations to transform one string to another (transformation distance)
- NP-hard

A restricted version: compression distance [Ming Li, et al, 2001,2003, Benedetto, et. al. 2001]

- Evolutionary relationship of sequences,
- Authorship of works of literature

Compression Transformation

Given two strings R, S, transform R to S: R→S

start with R'=R,

iteratively copy any substring of R' or insert any single character to its right end, and

final operation: delete original version of R from the left end of R'

Compression Distance

Given a transformation $R \rightarrow S$,

distance $d(R \rightarrow S)$= minimum total cost of edit operations to transform R to S.

Symmetric distance

$$d(R,S) = (d(R \rightarrow S) + d(S \rightarrow R))/2$$

Compression distance $c(R,S)$ is the symmetric distance corresponding to compression transformation $c(R \rightarrow S)$ from R to S.

Compression Transformation

Given two strings R, S, transform R to S: R→S

start with R'=R,

iteratively copy any substring of R' or insert any single character to its right end, and

final operation: delete original version of R from the left end of R'

Greedy Compression Transformation

Given two strings R, S, transform R to S: R→S

start with R'=R,

iteratively copy any substring of R' or insert any single character to its right end, and

final operation: delete original version of R from the left end of R'

Replace with

longest prefix of (remaining) S which appears as a substring of R'

Compression Distance: Greedy version

R,S two sequences,

gc(R,S):symmetric distance associated with greedy compression transformation,

c(R,S): compression distance.

Lemma: gc(R,S)=c(R,S).

(*greedy*) Compression distance - Example

Given R=A,C,T,A,G,T,A,T, and S=A,G,T,C,T,A,A,T

Compute c(R,S).

R'=R=A,C,T,A,G,T,A,T

*longest prefix of S which is a substring of R' is S'[4:6]=*A,G,T.

R'=A,C,T,A,G,T,A,T,A,G,T

similarly, C,T,A is a prefix in (rest of) S which is a substring in R'.

R'=A,C,T,A,G,T,A,T,A,G,T,C,T,A

*finally, the rest of S exists in R' as S'[7:8]=*A,T

R'= A,C,T,A,G,T,A,T,A,G,T,C,T,A, A,T

deleting the original R as a prefix as a prefix of R', leaves

R'=S=A,G,T,C,T,A,A,T

This implies c(R→S)=4.

Similarly, c(S→R)=5.

Thus, c(R,S)=c(S,R)=9/2.

Compression distance is efficiently computable

Lemma: The compression distance c(R,S) between two strings R and S can be computed in O(|R|+|S|).

The lemma follows from on-line suffix tree construction [Rodeh, et. al, JACM, 1981]

Metric distance function

A metric space is a set X with a distance function d: $X^2 \rightarrow R$ such that for all x,y,z in X,

- d(x,y)=d(y,x). *(symmetry)*
- d(x,y) ≥ 0 and d(x,y)=0 iff x=y. (positivity)
- d(x,y)+d(y,z) ≥ d(x,z) *(triangular inequality)*

d(.,.) is a metric distance function.

Near neighbor query

Given a set of data objects $X=\{x_1,\ldots,x_n\}$,

a metric distance function d(.,.),

a constant r, and

a query point q,

retrieve all data objects that are within distance r to the query point.

$\{x_i \mid x_i \in X \text{ and } d(x_i,q) \leq r\}$

Problem: Weighted character edit distance and Compression distance are *not* metric (does not satisfy the triangular inequality!)

Character edit distance and Transformation (block edit) distance are both metric distances.

Solution: Notion of Almost Metric, i.e., reflexive, symmetric, and satisfy the triangular identity by a constant factor.

Example

R=v,w,x,y,z

Q=z,y,x,w,v

S=w,v,x,w,y,x,z,y,y,x,w,x,w,v,z,y,x,z,y,x,w,

y,x,w,v, z,y,x,w,v.

c(R→Q)=6=c(Q→R) c(R,Q)=6

c(Q→S)=11 c(S→Q)=2 c(Q,S)=13/2=6.5

c(R→S)=31 c(S→R)=6 c(R,S)=37/2=18.5

c(R,S) not ≤ c(R,Q)+c(Q,S) *not metric*

(*Greedy*) Compression Transformation

Given two strings R, S, transform R to S: R→S

start with R'=R,

iteratively copy longest prefix of (remaining) S which appears as a substring of R' or insert any single character to its right end, and

final operation: delete original version of R from the left end of R'

Relaxed Compression Transformation

Given two strings R, S, transform R to S: R→S

start with R'=R,

iteratively copy longest prefix of (remaining) S which appears as a substring of R' or insert any single character to its right end, and

final operation: delete original version of R from the left end of R'

allow

deletion of *any prefix of R'* as the final operation.

Relaxed Compression Distance

Lemma: Relaxed compression distance rc(R,S) is a metric.

Unfortunately, relaxation makes the distance much harder to compute.

Lemma: Computing rc(R,S) is NP-hard.

"Almost" Metric

A distance function f is *almost metric* for space P, if it is

- *symmetric*,
- *reflexive*, and
- *satisfies triangular inequality within a constant factor k.*

for all s, r, q in P, $f(s, r) \leq k. [f(s, q)+f(q, r)]$

Compression Distance is Almost Metric

Theorem: $rc(R,S) \leq c(R,S) \leq 3.rc(R,S)$

which implies,

$$c(r,s) \leq 3\,[c(r,q) + c(q,s)]$$

Summary

Compression Distance is used to capture evolutionary relationships between sequences, including

- DNA, RNA sequences [Ming Li, et. al. Bioinformatics, 2001, Varre, et. al., Bioinformatics, 1999], as well as
- Text strings, in different world languages [Benedetto, Physical Rev. Letters, 2002].

It is efficiently computable—linear in size of the sequences.

We show that it is almost metric, so that we can use index structures designed for metric spaces with some modification.

Distance Based Indexing

Inherently different from Spatial Indexing, or Multidimensional indexing.

Here *only* the relative distances are used for index construction, space partitioning, and search, i.e. no spatial information on the data elements are utilized.

Examples: VP-Tree *[Burkhard, Keller, 1973, Uhlmann, 1991, Yianilos, 1993]*, GNAT *[Brin, 1995]*, MVP-Tree *[Bozkaya, Z.M.Ozsoyoglu, 1997, 1999]*, M-Tree *[Ciaccia, Patella, Zezula, 1997]*.

VP-Tree

Binary trees that recursively partition data space using

- *vantage points*, and
- distances of data points to vantage points.

Internal nodes: (x_v, M, R_{ptr}, L_{ptr}), where

M: median distance of among $d(x_{vp}, x_i)$ for all x_i in the space partitioned.

x_{vp}: Vantage point.

External nodes: References to data points are kept.

Near Neighbor Search in VP Trees

Given a query point q, a distance metric d(.,.) and a tolerance factor r,

Find all data points x such that $d(x,q) \leq r$.

1. If $d(q, x_v) \leq r$ then x_v is in the answer set.
2. If $d(q, x_v) - r \leq M$ then recursively search the left branch.
3. If $d(q,x_v) + r \geq M$ then recursively search the right branch.

Note that both branches can be searched if both search conditions are satisfied.

Distance Based Indexing for *Almost Metrics*

q: *query element*

r: *query range (i.e. tolerance)*

x_v: *vantage point*

M: *median distance value for M*

d(x,y): *almost metric distance function (satisfies triangular inequality) with constant 3.*

Then,

If $d(x_v,q)+r < M/3$ then no need to search right branch.

If $d(x_v,q)-3r > 3M$ then no need to search left branch.

Synthetic Data Generation

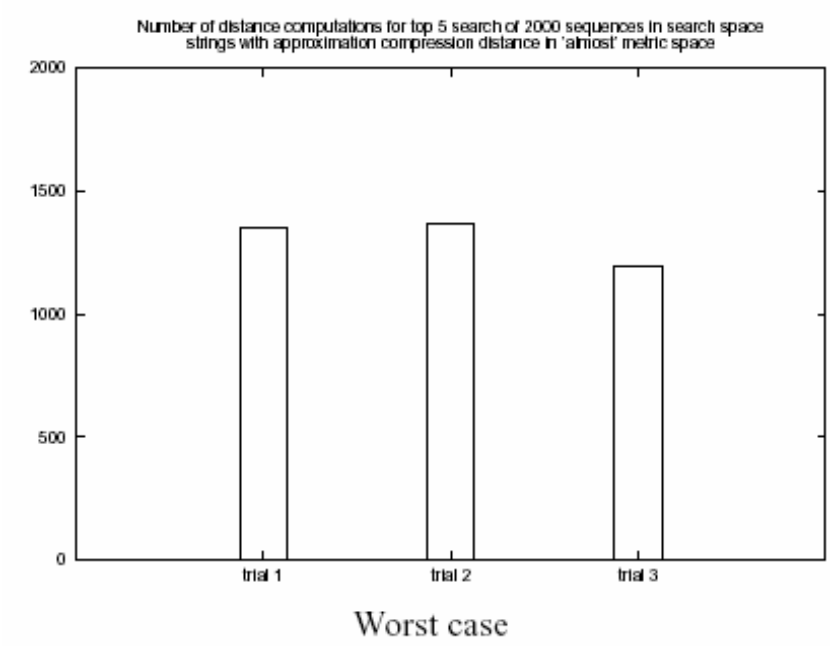
- started with a random “seed” sequence
- 5 neighbor sequences were created by randomly applying 5 block edit operations to the seed for each new sequence
- 50 moderately near neighbor sequences made same way but with 15 block edit ops
- 1945 far neighbors created by applying 45 block edit ops

Synthetic Data Querying

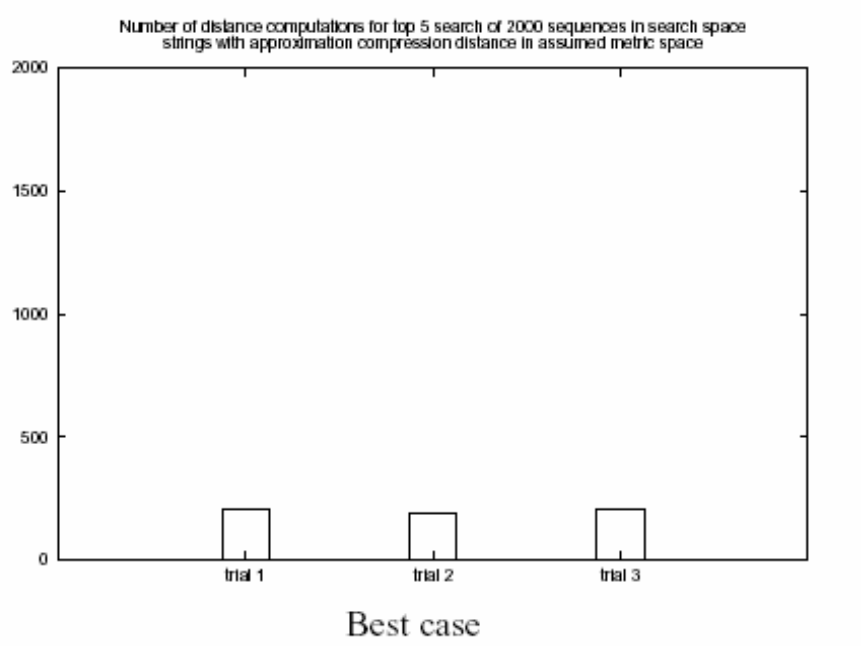
- 3 randomized indices built on the 2000 sequences derived from the seed
- performed search using seed as query with a range of 5 block edit operations (to retrieve the first 5 neighbors)
- each search done two ways: one assuming metricity, and one with almost metric at maximum factor (3)

Synthetic Data Querying Results

- brute-force would be 2000 comparisons
- ~37% pruning with almost metric search
- ~90% pruning with assumed metric search



Worst case



Best case

Protein Sequence Data 1

- a collection of proteins that are active in mammalian brains
- search results were inconsistent with predictions, so we took a closer look at the data
- we compared each sequence against each other sequence, giving a distance distribution

Protein Data

Polynomial distribution under both character edit distance and compression distance.

$f(r) = k.r^c$ for some k and c.

can be observed from log-log plots of f(r) against r

(log f(r) = log k + c. log r).

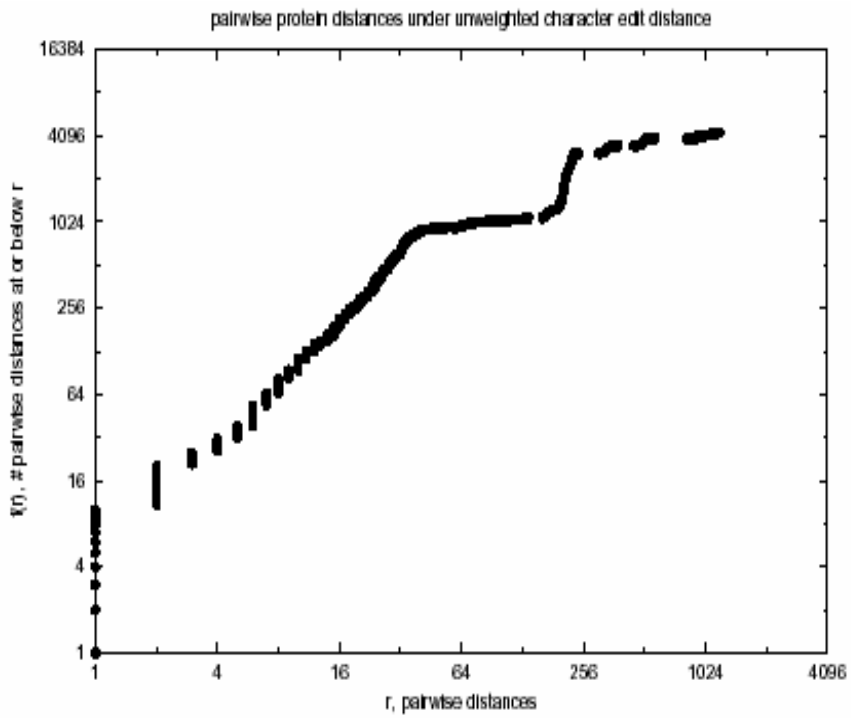
constants k and c vary only slightly for two distance metrics.

Data set:

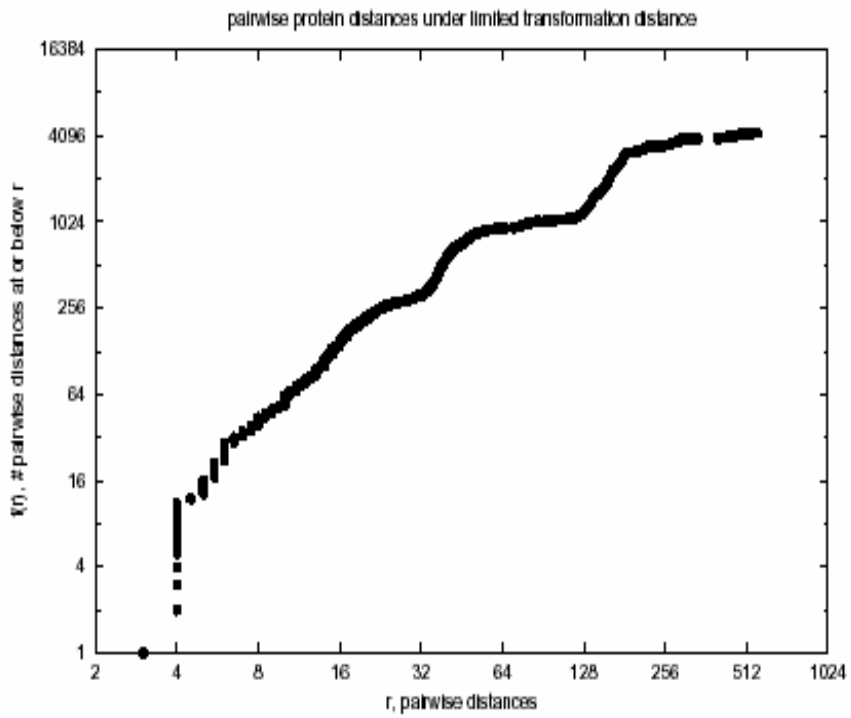
Complete set of protein strings that are active in the brain cells of Humans and other organisms from SwissProt (93 proteins)

Protein Sequence Data 1

- this set of proteins exhibits a polynomial distribution over in the input space



Character edit distance



Compression distance

Language Data Set

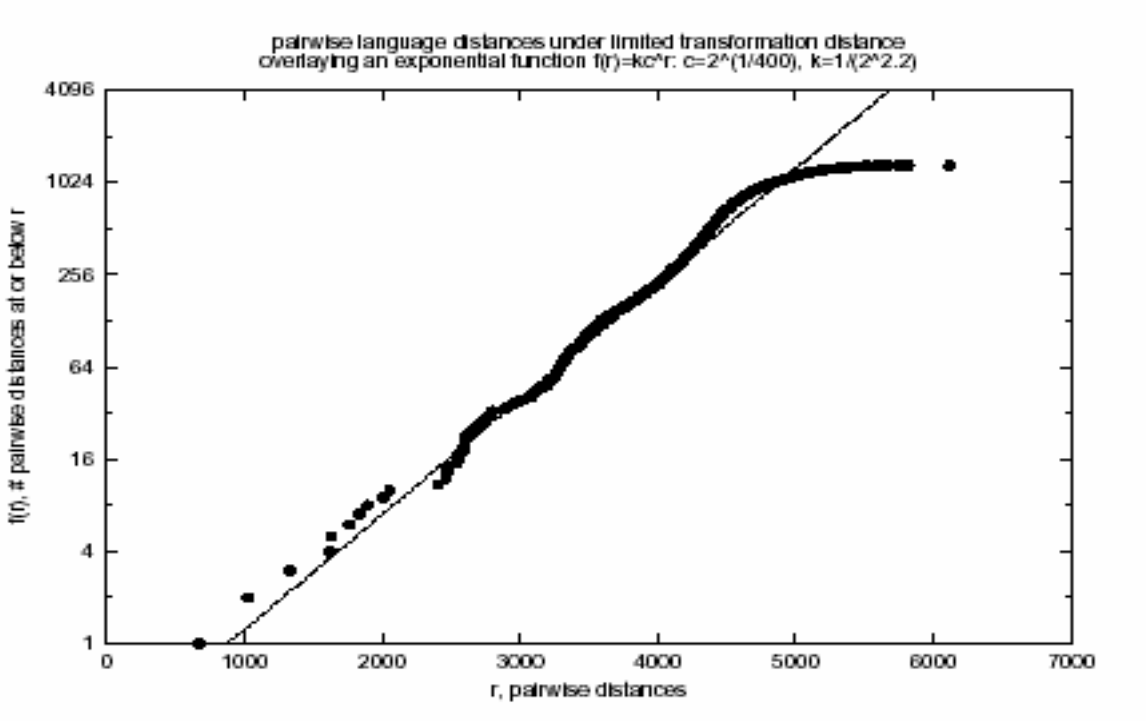
- to help figure out why the performance varied across different data sets, we also looked at a single text document (The Declaration of Human Rights) translated into 52 Eurasian languages

- languages follow an evolutionary path as well as genomes of species

- block edit distance used to compare each document

Linguistic Data Set

- this set of documents exhibits an exponential distribution over the input space



Linguistic Data

Linguistic data under compression distance:

Exponential distribution: $f(r)=k.c^r$ for some constants k and c, where

c is approximately $2^{1/400}$

k is approximately $2^{-2.2}$

log f(r) = *log* k + r. *log* c *(log f(r) is linear against r)*

Data set:

Declaration of Human rights is 52 Eurasian languages

[Benedetto et.al, 2002, Li, et.al, 2001]

What does this mean?

- by fitting a polynomial or exponential function to the distributions, we can more effectively tune and predict effectiveness of our index

- example: rather than making each subdivision have equal cardinality during construction, a data set may indicate that the split should not be even

Protein Data Set 2

- assuming a configuration of our index for a polynomial distribution, we performed queries on a large subset of human proteome sequences (~25000 sequences)
  - full pairwise distance distribution can take a long while to acquire: $^1/_2(25000^2)$ computations

- Levenshtein distance used to build and search through index

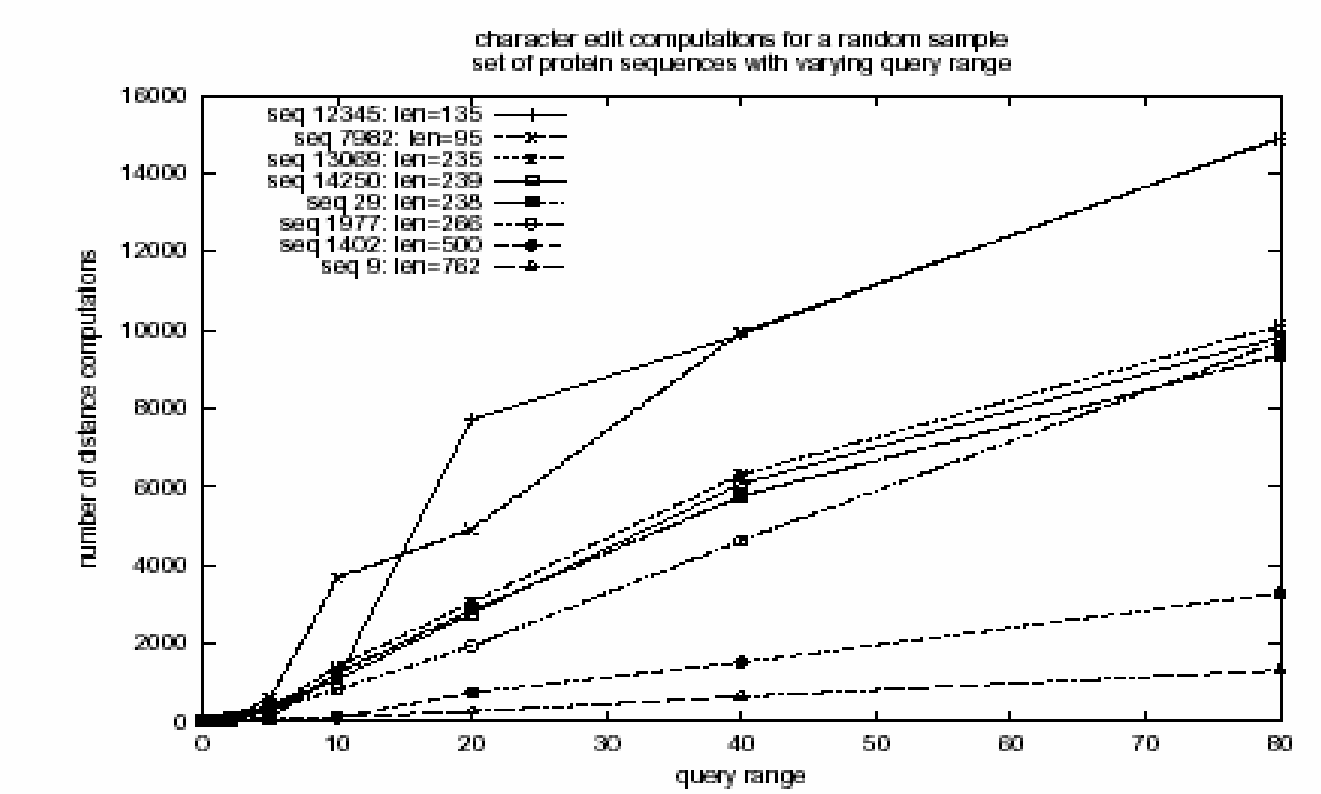
Protein Data Set 2

- query sequences were actual sequences in index (so those searches found “themselves” and any other neighbors within search radius)

- search results varied, but for reasonable search radii, all showed some significant pruning levels (brute-force method would entail ~25,000 distance computations)

Protein Data Set 2

- performance may be a function of length?
- average performance gives >50% pruning



Overall Experimental Results

- pruning results during search indicate very promising results

- since BLAST/FAST only work on specialized scoring methods, immediate use of index could be to perform quick searches using block edit distance approximation and other general distance functions

What now?

- make effective comparisons with BLAST/FAST
- analyze large data set distributions and customize indices accordingly to gain increased performance (experiments here were on very rudimentary version)
- figure out how to incorporate similarity scoring

how to compare with BLAST

- since BLAST is heuristic, last step is doing regular alignment (computationally equivalent to a distance computation)
- we can count how many good hits BLAST finds and runs this last step on, and use it as a baseline comparison... if the values are close, we have a competitive search tool

Similarity Score vs Distance

- basic example:
  - using BLOSUM62 scoring matrix
  - alignment 1: **S = AAAAA** score = 20
    T = AAAAA
  - alignment 2: **S = WWWWW** score = 55
    T = WWWWW
  - distance score of alignment 1 and 2 = 0
  - remember that a metric distance is needed for searching

Summary

We investigate applicability of Distance Based Indexing for Sequence Similarity Search.

- Show that compression distance (weighted character edit distance) is almost metric
- Modify VP-trees (other distance based index structures) to use with almost metrics
- Experimental results using synthetic data and real protein data

Conclusions & Future Work

- Indexing sequences by relative distances seems to be a promising search technique
- Many improvements can still be made, and comparisons with other search tools need to be done
- Resolution of similarity vs distance needs to be continually addressed as well to ensure data is biologically meaningful
- Explore Variations of VP tree which will perform better for data with different distance distributions,
- More experimental results.