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

Copyright 2004 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
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 $O(m^2)$.

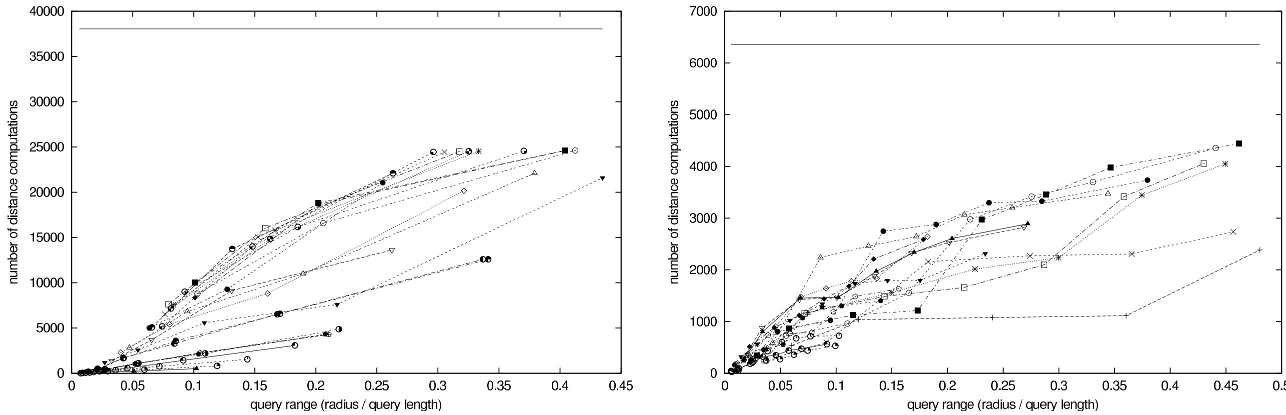

Figure 1: Human (left) and a yeast (right) proteomes indexed using a standard binary VP-Tree. Each line indicates the query results (in number of comparisons) for a query sequence (picked from the data set itself) with increasing query radius (as a percentage of the query length itself). Note that the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