A New Edit Distance Method for Finding Similarity in DNA Sequence

Patsaraporn Somboonsak, Mud-Armeen Munlin

Abstract—The P-Bigram method is a string comparison method base on an internal two characters-based similarity measure. The edit distance between two strings is the minimal number of elementary editing operations required to transform one string into the other. The elementary editing operations include deletion, insertion, substitution two characters. In this paper, we address the P-Bigram method to solve the similarity problem in DNA sequence. This method provided an efficient algorithm that locates all minimum operation in a string. We have been implemented algorithm and found that our program calculated that smaller distance than one string. We develop P-Bigram edit distance and show that edit distance or the similarity and implementation using dynamic programming. The performance of the proposed approach is evaluated using number edit and percentage similarity measures.

Keywords—Edit distance, String Matching, String Similarity

I. INTRODUCTION

The edit distance is a common similarity measure between two strings. It is defined as the minimum number of insertions, deletions or substitutions of single terminal needed to transform other of the strings into the other one. This distance is a key importance in several fields, such as Bioinformatics, Text processing consequently computational problems. Given a string $S_1 = [a_1 a_2 a_3… a_n]$ and $S_2 = [b_1 b_2 b_3… b_m]$ as the minimal cost of transforming $S_1$ into $S_2$ using the three operations insert, delete, and substitution, where only unit cost operations are considered in edit distance. The cost of elementary editing operations is given by some scoring function which induces a metrical on strings. The similarity of two strings is the minimum number of edit distance. DNA sequence can be seen as a pair of reverse complementary strings. The similarity relation on nucleotides (A T C G) means that A is complementary to T and C is complementary to G. In this paper, we develop two-letter edit distance. We compute the edit distance and the cost of operation with dynamic programming. The new algorithm is to find all minimal distance in a string.

II. RELATED WORK

The early works on finding similarity in strings deal with edit distance. The edit distance models were studied in two contexts, for string matching and for sequence similarity. A lot of works have been on string matching (see [6-11]).

We will focus on edit distance techniques as our main goal here is to concentrate on similarity. We decided to use the most common measure: the Levenshtein edit distance[4]. In the text is typically assumed to be random, i.e., each character is chosen uniformly and independently from the alphabet. A sequence alignment is a way of arranging the sequence of DNA, RNA or protein to identify regions of similarity that may be a consequence of functional, structural or evolutionary relationships between the sequences[2]. The first application of the edit distance algorithm for protein sequence was studied by Needleman-Wunsch[1]. It is commonly used in bioinformatics to align protein or nucleotide sequences. To find the alignment with the highest score, a two-dimensional array. The algorithm is an example of dynamic programming matrix. There is one column for each character in sequence A, and one row for each character in sequence B. This algorithm progresses, the $F_{ij}$ will be assigned to be the optimal score for alignment of the first $i = 0,\ldots,n$ characters in A and the first $j = 0,\ldots,m$ characters in B. This algorithm works in the same way regardless of length or complexity of sequences. The simplest and most common scoring function is the Levenshtein distance[4] which assigns a uniform score of string for every operation. Determining the edit distance between a pair of string is a fundamental problem in computer science in general, and in combinatorial pattern matching in particular, with applications ranging from database indexing and word processing, to bioinformatics[3][5]. The Levenshtein distance[4] between two strings is the minimum number of editing steps that convert one string into another. Given two strings $A[1\ldots,m]$ and $B[1\ldots,n]$, one can calculate their edit distance by dynamic programming. We refer to matrix $D[0..m, 0..n]$ as the edit distance table of string A and B. Initially, $D[i,j] = i$ for $0 \leq i \leq m$ and $D[0,j] = j$ for $1 \leq j \leq n$. Then the cell $D[i,j]$, where $i,j > 0$, stores the edit distance of string $A[1\ldots i]$ and $B[1\ldots j]$. We also say that the cells $D[i,j]$, where $j - i = d$ are on diagonal d of $D[1\ldots 15]$. We introduce some important properties of the edit distance table that are constantly used in later discussion. The Longest common subsequence(LCS)[14] problem is to find the maximum possible length of a common subsequence of two strings, ‘a’ of length [a] and b of length [b]. The sequence similarity analysis is the Longest Common Subsequence(LCS) problem, where we eliminate the operation of insertions deletions and substitutions[14]. Given strings S and T of lengths n and m, respectively, over an alphabet $\Sigma$, determine the lengths of the longest subsequence that is common to both s and t. Here, S subsequence of $S = s_1 s_2\ldots s_l$ is a string of the form $s_{i_1} s_{i_2}\ldots s_{i_k}$ where each $i_j$ is between 1 and n and $1 \leq i_1 < i_2 < \ldots < i_k \leq n$. For example, if $\Sigma = \{A, T, C, G\}$, $S = AGCGA$ and $T = CAGATAGAG$, then CGA is a subsequence of length 3 of both S and T. However, it is not the longest common...
subsequence of S and T, since the string AGGA is also a common
subsequence of length 4 of both S and T. Since these
two strings do not have a common subsequence of length of
the longest common subsequence of S and T is
4[16].Bioinformatics have developed several means to
characterize the similarity between genetic sequences. One
intuitively appealing measure is the edit distance. The edit
distance was originally proposed by Levnshtein[4]. One of the
major problems is that of edit distance. To compare two
strings, a common technique is so called string edit distance,
as a measure of their dissimilarity. Edit distance is equal to
the minimum number of editing operations required to transform
one sequence into the other. The three basic editing operations
are insertion, deletion, and substitution[11]. The operations to
transform one string into another string using the basic character
wise operation delete, insert and substitution. If each
operation has cost of 1, then edit distance is the number of
operations. The minimal number of these operations is called
edit distance or Levenshtein[4] distance. Each operation has
an associated cost, which is a function of characters involved
in operation. The cost of transformation is the sum of cost the
individual operations. Dynamic programming (DP) is a
method used for optimizing a multistage process and that is
particularly applicable to problems requiring a sequence of
interrelated decisions. Each decision transforms the current
situation into a new situation. A sequence of decisions, which
in turn yields a sequence of situations, is sought that
minimizes some measure of value. The value of a sequence of
decisions is generally equal to the sum of the values of the
individual decisions and situations in the sequence. It is a
“solution seeking” concept which replaces a problem of n
decision variable. Such an approach allows analysts to make
decisions stage by stage, until the final result is obtained. We
use the following formulation[13].Let S = \{s_1, s_2, \ldots, s_n\} be the
source sequence, T = \{t_1, t_2, \ldots, t_m\} the target sequence, and d_{i,j}
the distance between the subsequence \{s_1, s_2, \ldots, s_i\} and
\{t_1, t_2, \ldots, t_j\};

Then for 1 \leq i \leq m, 1 \leq j \leq n,

\begin{align*}
d_{0,0} &= 0 \\
d_{i,0} &= d_{i-1,0} + c(S_i,\emptyset) \\
d_{i,j} &= d_{i-1,j} + c(\emptyset, t_j)
\end{align*}

and

\begin{align*}
d_{i,j} &= \min \left\{ \\
&\quad d_{i-1,j} + c(S_i,\emptyset) \\
&\quad d_{i,j-1} + c(\emptyset, t_j) \\
&\quad d_{i-1,j-1} + c(S_i, t_j) \right\}
\end{align*}

where c(S_i,\emptyset) is the cost of deleting S_i, c(\emptyset, t_j) is the cost of
inserting t_j, and c(S_n, \emptyset) is the cost of substituting t_j for S_i. Theedit distance between S and T is simply d_{m,n}.

III. MATERIALS AND METHODS

In this section we describe the algorithm for finding P-
Bigram similarity new edit distance. We first describe P-
Bigram Distance, following which we describe reduced
operation to achieve a time and space efficient algorithm. The
reduce use similar idea to those in Levenshtein[4] distance.

This P-Bigram edit distance is an importance of the edit
distance for two characters which is also used for measuring
the similarity.

A. P-Bigram Edit Distance

We define a P-Bigram Edit Distance as a character that is
present in two nucleotide sequences. So, we estimate the
minimum length for which a maximal match is significant,
according to the length of the two compares sequences. Let S
= \{s_1, s_2, s_3, \ldots, s_n\} be a set of n characters and T = \{t_1, t_2, t_3, \ldots, t_m\}
in a text of m characters which are strings of nucleotide
sequence characters of length i and j over the nucleotide
alphabet S = \{A, C, G, T\}.

\textbf{Definition 1.} A P-Bigram Edit distance solution for
computing the edit distance between a pair of string S =
\{s_1, s_2, s_3, \ldots, s_n\} and T = \{t_1, t_2, \ldots, t_m\} involves filling in an (n+1) X
(m+1) table P, with P[i,j] sorting the edit distance between
\{s_1, s_2, \ldots, s_i\} and \{t_1, t_2, \ldots, t_j\}. In addition let |S| and |T|
denote the length of string S and T. We consider the Levenshtein
distance. The computation is done according to the base-case
rules given by P[0,0] = 0, P[i,0] = T[i-1,0] + cost of deleting
s_i and P[0,j] = P[0,j-1] +1 cost of inserting t_j, and according to the
following dynamic programming step:

\begin{align*}
P[i,j] &= \min \left\{ \\
&\quad P[i-1,j] + \text{the cost (1) of delete } s_i \\
&\quad P[i,j-1] + \text{the cost (1) of insert } t_j \\
&\quad P[i-1,j-1] + \text{the cost (1) of substitute } s_i \text{ with } t_j \right\}
\end{align*}

\textbf{Definition 2.} For a string S and value of \mathbb{E}, let P = \mathbb{E} [S] and
string T and value of \mathbb{E}, let P = \mathbb{E} [T]. For example, the two-
letter alphabet \{s, t\}, if S = ATTCCGCTCAAG and T =
ATTGGTATCAAGGA. The first scans and match two-letter
sequence from left to right then a pair of string S is\{ AT,
TC, CG, GA \} and a pair of string T is \{ AT, TG,
GT, TC, CA, AG \}. The P-Bigram Edit Distance between
two characters with two strings. Given two strings S, T a
standard technique for computing P-Bigram (S, T) is
the dynamic programming method, where we compute the DP
matrix of size (|S|+1) by (|T|+1) for which DP[i,j] = P-Bigram
(S[1..i], T[1..j]) for 1 \leq i \leq |S| and 1 \leq j \leq |T|.

The compute matrix is the following:

\begin{align*}
\text{DP}[i,j] &= \begin{cases} \\
0 & \text{if } i = 0 \text{ or } j = 0 \\
\max \{ \text{DP}[i-1,j], \text{DP}[i,j-1] \} & \text{if } i,j > 0 \text{ and } S[i] \neq T[j] \\
\text{DP}[i-1,j-1] + 1 & \text{if } i,j > 0 \text{ and } S[i] = T[j] \\
\end{cases}
\end{align*}

Therefore, to compute P-Bigram = DP S and T of length |S|
and |T| with a time complexity of O(|S| X |T|) and space
complexity of O(min(|S| X |T|)).
P-Bigram Edit Distance

1. EditDistance S(0...s-1) T(0...t-1)
2. int m[i,j] = 0
3. for i = 0 to n
4. do m[i,o] = i
5. i = i + 1
6. for j = 0 to m
7. do m[o,j] = j
8. j = j + 1
9. for i = 0 to n; i ++
10. for j = 0 to m; j ++
11. if s[i,i-1] = t[j,j-1] then
12. cost = 0
13. m[i,j] = m[i,j] +1
14. opt = delete(i,n) // opt operation of i and j
15. else
16. if s[i,i-1] ≠ t[j,j-1] then
17. cost = 1
18. m[i,j] = m[i,j] +1
19. opt = insert(n,i,j) // opt operation of i and j
20. else
21. if s[i,i-1] ≠ t[j,j-1] then
22. cost = 1
23. m[i,j] = m[i,j] +1
24. opt = substitution(n,i,j) // opt operation of i and j
25. for i = 1 to |S|
26. do for j = 1 to |T|
27. do m[i,j] = min{m[i-1,j-1], m[i-1,j] + 1, m[i,j-1] + 1}
28. return m[n,m]

P-Bigram algorithm compute the edit distance. Suppose we wish to calculate the edit distance between the strings S = s_1,s_2,...,s_n and T = t_1,t_2,...,t_m.

1. We begin by forming an (n + 1) X (m + 1) matrix P initially containing all zeros, that is P(i,j) = 0 for i = 0,1,2,...,n and for j = 0,1,2,...,m.
2. Assign values P_0j = j, j = 0,1,2,...,m and P_0i = i, i = 0,1,2,...,n.
3. Starting from the second top row and going from left to right, we fill in the values P(i,j) according to the following step:
   - Insertion = P(i-1,j-1) + cost
   - Deletion = P(i-1,j) + 1
   - Substitution = P(i,j-1) + 1
   \{ Substitution = min(substitution, deletion, insertion) \}

Where cost = 0 if s_i = t_j and cost = 0 if s_i ≠ t_j.
4. After computing a row, move to the row below, until the bottom row is reached.
5. The value P_nm is the edit distance between the string S and T.

Example Here is an example of computing the P-Bigram edit distance of two characters in two strings.

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>AT</th>
<th>TG</th>
<th>GT</th>
<th>TC</th>
<th>CA</th>
<th>AG</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>AT</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>TC</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>CG</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>GT</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>TC</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>CA</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>AG</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Fig. 1 P-Bigram edit distance of S_1 = ATTCCGTCAG and T_1 = ATTGGTTCCAAGGA.

The values of the above table have been obtained with the following unitary costs(Fig. 1, Fig. 2) Sub(s, t) = 1 if s ≠ t and Sub(s,t) = 0, Insert(t) = Delete(t) for s,t ∈ P.

A score (instead of a cost) is associated with each elementary edit operation. For s,t ∈ P:
- Sub(s,t) denotes the score of substituting the character t for the character s.
- Del(t) denotes the score of deleting the character t,
- Ins(t) denotes the score of inserting the character t.

Fig. 2 The DP matrix for P-Bigram(S,T), where S = ATTCCGTCAG and T = ATTGGTTCCAAGGA

The main idea behind the P-Bigram edit distance described above is that each entry P(i,j) corresponds to the minimal number of editing operations required to transform the substring S_i = s_1,s_2,...,s_i into the substring T_j = t_1,t_2,...,t_j. Initially, an empty string to transformed into a string of k characters by using exactly k additions (Step2). Explanation of Step3:
- If we can transform S_i into T_j in P(i,j-1) operations, then we can transform S_i into T_j in insertion = P(i,j-1) + 1 operations by simply adding the characters s_i to T_j-1.
- If we can transform S_i into T_j in P(i-1,j) operations, then we can transform S_i into T_j in deletion = P(i-1,j) + 1 operations by simply deleting the characters s_i from to S_i.
- If we can transform S_i into T_j in P(i-1,j-1) operations, then we can transform S_i into T_j in substitution = P(i-1,j) + 1 cost operations by simply substitution the characters s_i with T_j if they are different (cost = 1).
The minimal number of operations required to transform $T_i$ into $S_j$ is the minimum of the three quantities: $P_{ij} = \min(\text{substitution}, \text{deletion}, \text{insertion})$.

IV. EVALUATION AND RESULTS

In this section, we want to evaluate a similarity measure, so our evaluation will focus on edit distance a set of source nucleotide to a set of target nucleotide. We suppose that the size of the target set is a part of nucleotide to contain possible other similar nucleotides. The P-Bigram Edit Distance task consists of comparing two characters with two strings that contain nucleotide virus in order to decide whether two strings refer to the same entity. A data set used for testing the nucleotide virus dynamic programming techniques is usually represented as two sets of strings and subset of their Cartesian product that defines valid matches. We measured percentage similarity and edit of the matched characters in the string. It follows from definitions that the following (Table 1).

<table>
<thead>
<tr>
<th>Table I</th>
<th>NOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim</td>
<td>Similarity between S and T is related to their commonality</td>
</tr>
<tr>
<td>sim_{edit}(x,y)</td>
<td>Similarity of edit</td>
</tr>
<tr>
<td>editDist(x,y)</td>
<td>Minimum number of character (insertion, deletion, substitution)</td>
</tr>
</tbody>
</table>

Several metrics to evaluate effectiveness of character identification techniques have been proposed, combining such criteria. The problem of DNA similar measure is a different measures how many match is identified in relation to the total number of edit. The Results given in Table1 are the percent similarity and number of edit[12][17].

$$\text{sim}_{edit}(x,y) = \frac{1}{1 + \text{editDist}(x,y)}$$

Where editDist(x,y) is the minimum number of data(DNA) insertion deletion and substitution operations needs to transform one string to the other.

Example Here is an example of computing the P-Bigram similarity of two characters in two strings(ATTCGGTCAAG, ATGGTCCAAGGA).

$$\text{sim}_{edit}(s,t) = \frac{1}{1 + 3 (s,t)}$$

= 0.25

Similarity = 25.00%

Our training set is an export of the NCBI(National Center for Biotechnology Information) data. To simplify the evaluation, we had to set a threshold to decide if a nucleotide is a small edit operation. Using our trained similarity measure we computed the similarity between two characters in two strings and look for minimum edit distance (similarity) that was able to reduce edit operations. Our evaluation concentrates on trying to match two characters nucleotide. We summarize the results using edit and similarity in Table1, Table II, Table III, Figure 4, Figure 5.

<table>
<thead>
<tr>
<th>Table II</th>
<th>NUMBER EDIT FOR PAIR OF DNA SEQUENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Destination</td>
</tr>
<tr>
<td>LD</td>
<td>LCS</td>
</tr>
<tr>
<td>ND</td>
<td>PB</td>
</tr>
<tr>
<td>Edit</td>
<td>Edit</td>
</tr>
<tr>
<td>Edit</td>
<td>Edit</td>
</tr>
<tr>
<td>ATTCCG</td>
<td>ATGGTT</td>
</tr>
<tr>
<td>GTAAG</td>
<td>CCAAGGA</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>GAATTC</td>
<td>ATGGTTTC</td>
</tr>
<tr>
<td>AGTAA</td>
<td>CCAAGGA</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>GCATCG</td>
<td>ATCTCG</td>
</tr>
<tr>
<td>GTAAT</td>
<td>GACG</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>GCCCTA</td>
<td>GCGCAA</td>
</tr>
<tr>
<td>GCG</td>
<td>TG</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>TGATCG</td>
<td>CTGATCG</td>
</tr>
<tr>
<td>ATC</td>
<td>ATC</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

By using the edit distance we find the possible edit of sequence. Now to edit distance in many algorithms we used to compare edit. A better approach is to edit in such a way which minimum edit.

<table>
<thead>
<tr>
<th>Table III</th>
<th>SIMILARITY FOR PAIR OF DNA CHARACTERS IN MANY ALGORITHM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Destination</td>
</tr>
<tr>
<td>LD</td>
<td>LCS</td>
</tr>
<tr>
<td>ND</td>
<td>PB</td>
</tr>
<tr>
<td>Similary</td>
<td>Similary</td>
</tr>
<tr>
<td>Similary</td>
<td>Similary</td>
</tr>
<tr>
<td>ATTCGG</td>
<td>ATGGTT</td>
</tr>
<tr>
<td>GTAAG</td>
<td>CCAAGGA</td>
</tr>
<tr>
<td>14.29%</td>
<td>9.09%</td>
</tr>
<tr>
<td>12.50%</td>
<td>25.00%</td>
</tr>
<tr>
<td>GAATTC</td>
<td>ATGGTTTC</td>
</tr>
<tr>
<td>AGTAA</td>
<td>CCAAGGA</td>
</tr>
<tr>
<td>16.67%</td>
<td>14.29%</td>
</tr>
<tr>
<td>11.11%</td>
<td>20.00%</td>
</tr>
<tr>
<td>GCATCG</td>
<td>ATCTCG</td>
</tr>
<tr>
<td>GTAAT</td>
<td>GACG</td>
</tr>
<tr>
<td>12.50%</td>
<td>14.29%</td>
</tr>
<tr>
<td>14.28%</td>
<td>16.67%</td>
</tr>
<tr>
<td>GCCCTA</td>
<td>GCGCAA</td>
</tr>
<tr>
<td>GCG</td>
<td>TG</td>
</tr>
<tr>
<td>20.00%</td>
<td>16.67%</td>
</tr>
<tr>
<td>11.11%</td>
<td>25.00%</td>
</tr>
<tr>
<td>TGATCG</td>
<td>CTGATCG</td>
</tr>
<tr>
<td>ATC</td>
<td>ATC</td>
</tr>
<tr>
<td>50.00%</td>
<td>10.00%</td>
</tr>
<tr>
<td>12.50%</td>
<td>50.00%</td>
</tr>
</tbody>
</table>

Fig. 4 Number Edit

Fig. 4 Show the comparisons of different algorithms with the sim_{edit} metrics. The current metrics gives good performance in reducing the number of editions compared with other popular methods.
Fig. 5 Comparison of different algorithms with percentage similarity

V. CONCLUSION

We proposed an edit distance of the P-Bigram method integrating the dynamic programming concept to compared similarity. That edit distance implements a heuristic of giving greater importance in the combined measure to the pairs of strings whose similarity is higher in comparison with the similarity of other pairs. The proposed method was tested on 5 dna-matching datasets with representative two characters-based string measures: Levenshtein distance. The result showed that the performance of the P-Bigram method can be improved. The result of such process can then compute similarities between two characters two nucleotide with quite high accuracy. However there are several improvements we will try to address in the future.

ACKNOWLEDGMENT

This works was financially supported by Faculty of Information Science and Technology Mahanakorn University of Technology.

REFERENCES


Mud-Armeen Munlin holds a Ph.D. degree in Computer Science from The University of Leeds, UK in 1995. Assistant Professor worked as associate Dean for Academics Faculty of Information Science and Technology Mahanakorn University of Technology Thailand. His specialized in solid modeling, virtual reality, medical application, internet and mobile applications.

Patsaraporn Somboonsak received the Master degree in Management Information Technology from Walailak University Thailand in 2006. Currently she is a Ph.D. student at Faculty of Information Science and Technology Mahanakorn University of Technology Thailand. Her interesting research are bioinformatics and computational for health medicine.