Artificial Neural Network based String Matching Algorithms for Species Classification – A Preliminary Study and Experimental Results

Sathish Kumar S
Research Scholar
Dr MGR Educational and Research Institute University,
Chennai, Tamil Nadu, India

N. Duraipandian, M.E., Ph.D
Phd, Vice Principal
Velammal Engineering College
Ambattur – Redhills Road, Chennai,
Tamil Nadu, India

ABSTRACT
The preliminary research in the area of applications of neural networks and pattern matching algorithms in species classification is presented. Artificial neural networks for classification and different pattern matching algorithms for matching the given DNA patterns or strings with the existing DNA sequences available in the databases are specifically studied. A set of local searching algorithms were experimented for different test string lengths and their time complexity is tabulated. Conclusions and future directions are also presented.

Keywords
Alignment, ANN, DNA sequencing, Species classification, String matching

1. INTRODUCTION
Biological classification is the process by which scientists group living organisms. Organisms are classified based on how similar they are. Historically, similarity was determined by examining the physical characteristics of an organism but modern classification uses a variety of techniques including genetic analysis. [1]

In molecular biology, DNA sequences are the fundamental information for each species and a comparison between DNA sequences is an interesting and basic problem. There are various open databases available in different Countries to maintain the DNA sequences of already analyzed and clarified species. When the research laboratory or organizations landed in any of an unidentified species' DNA sequence, they need to compare that with the existing databases. From that they will find out suitable or nearer species which resembles the test data. This is called as classification or grouping. Mostly, the test DNA sequence may be a part of a gene of an organism and the available databases are very huge with billions of DNA sequence details.

Also DNA sequences can be handled as bit streams. Since a DNA sequence is constructed with four bases (A, C, T, and G). There are at least 26 billion base pairs (bp) representing the various genomes available in the server of the National Center for Biotechnology Information (NCBI). [2]

Gene identification is one of the most important tasks in the study of genomes. Methods such as clustering, data mining, gene identification etc were used in DNA analysis. The objective of these methods was to facilitate collaboration between researchers and bio-informaticians by presenting cutting edge research topics and methodologies. All the above studies aim to discover genes and their characteristic expression parameters, which help to discriminate between object classes. So far, great effort has been put into various methods of classification of genes for the purpose of DNA analysis. [22]

There are various kinds of string comparison tools available and they will provide approximate matching. However most of the tools are based on exact matching of the strings. But, the total number of sequences is rapidly increasing, efficient methods are needed. Today's requirement is a tool, not only for fast matching but also for efficient sequences storage. [10]

In this paper, we are comparing the existing algorithms for string matching as well as the tools available for the same. We try to introduce the concept of ANN into this, for speed up the process of string comparison. In section 2, we will discuss what data mining, Bioinformatics and neural networks are in brief. As well as we will discuss the application of them in biological databases particularly in species classification. In section 3, we will discuss about the use of neural network in datamining. In this Section we will discuss one simple algorithm (Back propagation algorithm) as well as one tool (which are using ANN as building block). In section 4, a brief about how data mining is used in Biological database with a simple block diagram is discussed. In section 5, we are discussing a whole lot of algorithms used for string searching. This Section contains the introduction of the algorithm along with their time complexity. Even the pseudo codes of most of the algorithms were provided. We classify the search algorithm in to two broad categories, one for local search and another for global search. In this section we are discussing some of the most famous local search algorithms. Section 6 is dedicated for the global search algorithms. In this Section we are discussing few ever green global algorithms along with their pseudo codes. Section 7 is dedicated for the comparison of the local search algorithms with some sample data as well as the graphical representation of that comparison. This will give the strength and weakness of every algorithm in different scenarios. Section 8 is discussing about the application of this species classification method in some other areas. Section 9 is discussing about the conclusion and future exploration areas of this field. Section 10 is reference section.

2 BASIC BUILDING BLOCKS
This section introduces the concept of bioinformatics and the applications of data mining, and neural networks in biological databases with simple block diagram.

2.1 Data Mining
Data mining is the process of discovering interesting patterns from massive amounts of data. In biological databases, the amount of data available is in trillions of bps. In Data mining,

as a process of knowledge discovery, the steps typically involves - data cleaning, data integration, data selection, data transformation, pattern discovery, pattern evaluation, and knowledge presentation [3]

So, in short, Data mining is the process of automating information discovery. It is the process of analyzing data from different perspectives, summarizing it into useful information, and finding different patterns. The following flow chart, figure 1 shows the simple stages of data mining even though the actual data mining needs minimum 7 building blocks.

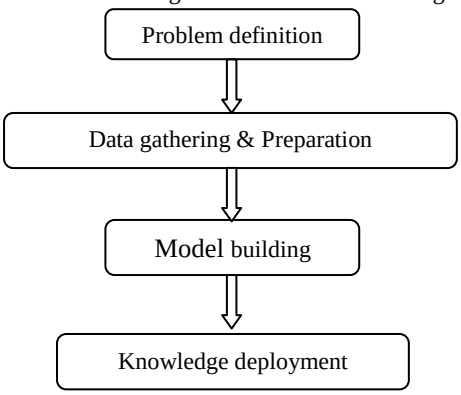

Figure 1: A simple flowchart representing the stages of simplified datamining

Our requirement in species classification is also involves the process of analyzing the data, summarizing the useful information and classify them into different groups

2.2 Bioinformatics

Bioinformatics is the field where computers are used to handle the biological information. It is an inter-disciplinary of mathematics, biology and computers. Bioinformatics refers to the use of computers for storing, comparing, retrieving, analyzing or predicting the composition or the structure of the biological information mostly available from research centers and research laboratories. [4] It is a large scale technique to conceptualize the biology in terms of basic molecules and apply informatics to understand them and organize them.

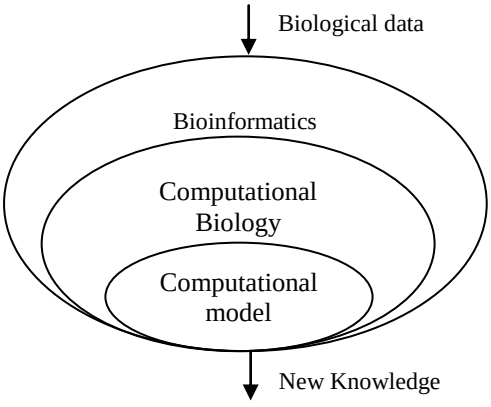

Figure 2: The block diagram of Bioinformatics

The biological database is broadly classified as structural biological database, sequential biological database and others. We are concentrating on sequential biological database as the DNA sequence is falling under this category. The following figure shows the categories falling under biological database.

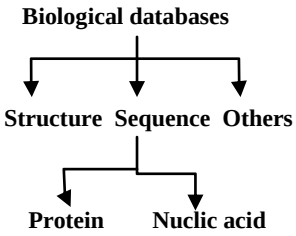

Figure 3: Broad classification of Biological Database

2.3 Neural Network

A machine that is designed to imitate the way human brain performs a particular task or function is called as neural network. To implement neural network in digital computer, either electronic components or software simulators will be used. To make neural network to perform useful computations, the process of learning is used. Based on the massive interconnections, the performance of a neural network will vary. The computing cells of neural networks were referred as neurons or processing units [5]

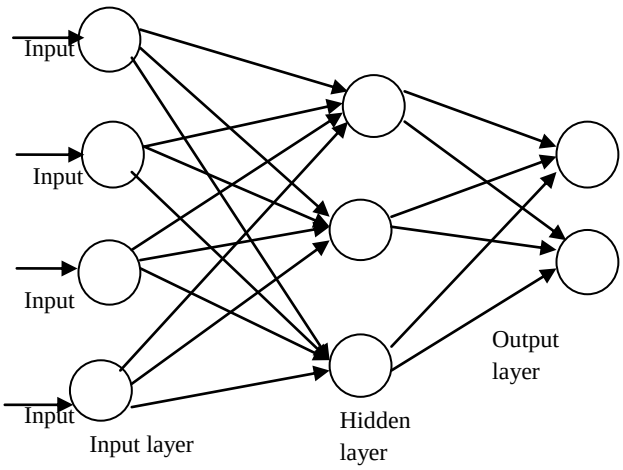

Figure 4: A typical neural network

The big Strengths of Neural Networks are, it is generalized and self organized. The neural network will work with inadequate and volatile knowledge base also. We can use incorrect or incomplete or noisy inputs to recall the information. This characteristic of the neural network is best suited for biological databases as most of them or noisy and incomplete. Even though the training phase takes reasonable time, the project development phase will take short time only. Even we prefer this characteristic is best suited for biological database.

3 NEURAL NETWORK AND DATA MINING

From the above discussions, we can find the clarity about the relationships of biological data, bioinformatics, data mining

and the neural networks. So to consolidate it, we need to do the following steps:

Develop a Neural Network Applications for our classification problem. Check that the neural network approach is appropriate and select an appropriate paradigm. Once that is done, start to give inputs by selecting proper inputs and facts. Once the inputs are in their proper format, start to train the network. Once the network is trained, test it with known data and check its perfectness. Now the neural network application is ready for run. Run the neural network for different unknown data sets and find out the similarities of the existing data and the test data. Table it. Run the same set of data for different string comparison algorithms and compare it our neural network's output. From the table we can conclude which is the best algorithm for the species classification.

3.1 Back Propagation Algorithm:

This is the common method of teaching artificial neural networks how to perform for a given task by sending the signals in the forward direction as well as the propagation of error in the backward direction. This method is used in the layered feed forward artificial neural networks. The concept is, the neurons are organized in layers and the signals are travelling in forward direction while the errors, if any, will be propagated in the backward direction. This type of back propagation is helpful for reducing the errors while the ANN is in the learning phase. Once the ANN finishes its learning, it will take the inputs and calculate the outputs more accurately. [7]

3.2 Example for Neural Network Tool

There are some standard neural network applications available in the market. For example, according to the website for NeuralWare (2005):

"NeuralWorks Predict® is an integrated, state-of-the-art tool for rapidly creating and deploying prediction and classification applications. Predict® combines neural network technology with genetic algorithms, statistics, and fuzzy logic to automatically find optimal or near-optimal solutions for a wide range of problems." [6]

The University of California at Irvine Machine Learning Repository had used the NeuralWorks Predict® software on data sets for variety of applications. The applications vary from USA Space Shuttle database to wine recognition databases. We can find out from this discussion that NeuralWorks Predict® can work with any sort of big databases. So we can even use this product for biological databases too.

4 PATTERN MINING FROM DNA SEQUENCE

The first and initial stage of the proposed technique mines the nucleotide pattern from the DNA sequence. At this stage, patterns formed by different combinations of nucleotides are mined using a novel mining algorithm. Let g be the DNA sequence, which is a combination of four nucleotides A, G, C and T. For instance, a sample DNA sequence is given as CGTCGTGGAA. From the sequence, the mining algorithm extracts different nucleotide patterns and their support. The algorithm is comprised of two stages, namely, pattern generation and support finding. In pattern generation, patterns with different length are generated whereas in support finding, support values for every generated pattern are determined from the DNA sequence. [8] The basic structure of the algorithm is given as a block diagram in Fig. 5.

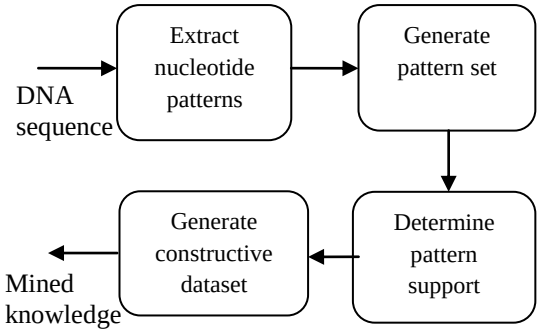

Figure 5: Block diagram of the pattern mining algorithm

5 ALIGNMENT METHODS

In this section we are trying to list out some of the most popular alignment algorithms, which were exist from the last century, their contribution in present day algorithms and their time complexity. If the sequences are very short or very similar, they can be aligned easily by hand. But most of the problems in biological databases require the alignment of extremely numerous sequences; they are very lengthy and highly different in similarity. In this case, we cannot align it by using manual methods. We need a high quality alignment algorithm for sequence alignment, which will run with very limited human intervention. This is called as Computational approach. The computational approach can be categorized broadly as: global alignment and local alignment.

Global alignment will take care of an end-to-end alignment of the sequences to be aligned. Calculating a global alignment is a form of global optimization that "forces" the alignment to span the entire length of all query sequences.

Local alignments identify regions of similarity within long sequences that are often widely different. Local alignments are often preferable, but can be more difficult to calculate because of the additional challenge of identifying the regions of similarity. A variety of computational algorithms have been applied to the sequence alignment problem, including slow but formally correct methods like dynamic programming, and efficient, heuristic algorithms or probabilistic methods that do not guarantee to find best matches designed for large-scale database search.[9]

5.1 Knuth–Morris–Pratt String Searching Algorithm (or KMP algorithm)

In this algorithm, we are searching for the occurrences of a test string "word" W within a main "text string" S. This algorithm employs the observation that when any mismatch occurs, the next match will be determined by the word itself by embedding sufficient information, thus bypassing the re-examination of previously matched characters. This algorithm is having two parts. The complexity of each algorithm is O(k) and O(n). So the overall complexity of Knuth-Morris-Pratt algorithm is O(n+k). The advantage of this algorithm is that the complexity is same and not depends on the size of W and S. [11]

5.1.1 The KMP Algorithm – Pseudo Code

```
algorithm kmp_search:
  input:
    an array of characters, S
    an array of characters, W
  output:
```

an integer (the zero-based position in S at which W is found)
define variables:
an integer, m ← 0
an integer, i ← 0
an array of integers, T
while m+i is less than the length of S, do:
if W[i] = S[m + i],
if i equals the (length of W)-1,
return m
let i ← i + 1
otherwise,
let m ← m + i - T[i],
if T[i] is greater than -1,
let i ← T[i]
else
let i ← 0
(if we reach here, we have searched all of S unsuccessfully)
return the length of S

5. 2. Boyer–Moore String Search Algorithm (or BM algorithm)

This algorithm is the standard benchmark for practical string search because of its efficiency in string searching. BMS algorithm will preprocesses the searching string (the pattern), but will not modify the string being searched in (the text). This algorithm is best suited for applications where the text does not continue to be same across multiple searches. The information gathered during the pre-process will be used to skip the sections of the text to lower the constant factor. This advantage is missing in many other algorithms. If the pattern length increases, this algorithm will run faster.

The Boyer-Moore algorithm has worst-case running time of O(nm) only if the pattern does not appear in the text.

5.2.1 The Boyer- Moore Algorithm – Pseudo Code

Suppose to alignment P and T, a substring t of T matches a suffix of P, but a mismatch occurs at the next comparison to the left. If it exists, the right-most copy t' of t in P such that t' is not a suffix of P and the character to the left of t' in P differs from the character to the left of t in P. Shift P to the right so that substring t' in P is below substring t in T. If t' does not exist, then shift the left end of P past the left end of t in T by the least amount so that a prefix of the shifted pattern matches a suffix of t in T. If no such shift is possible, then shift P by n places to the right. If an occurrence of P is found, then shift P by the least amount so that a proper prefix of the shifted P matches a suffix of the occurrence of P in T. If no such shift is possible, then shift P by n places; shift P past T. [11]

5.3. Rabin–Karp String Searching Algorithm

The RK string searching algorithm is a string searching algorithm that uses hashing to find any one of a set of pattern strings in a text. For text of length t and patterns of p and their combined length m. This algorithm's average and best case running time is O(n+m). It's worst-case time is O(nm). [23]

5.3.1 The RK Algorithm – Pseudo Code

Function rabin_karp
Let m is the length of the text string t
Let n is the length of the pattern string p
Let si denote the length-n contiguous substring of t beginning at offset i ≥ 0
use a hash function h to map each si to a good sized set of the first k nonnegative integers
compute h(p)
for every I, h(si) = h(p), check for a match as in the naive algorithm
If h(si) = h(p), stop the checking operation

5.4. The Aho–Corasick String Matching Algorithm

The AC string matching algorithm is dictionary-matching algorithm that locates elements of a finite set of strings (the "dictionary") within an input text. It matches all patterns simultaneously. Because all matches are found, there can be a quadratic number of matches if every substring matches. Basically, the algorithm constructs a finite state machine that resembles a prefix tree with additional links between the various internal nodes. These extra internal links allow fast transitions between failed pattern matches, to other branches of the prefix tree that share a common prefix. This allows the automaton to transition between pattern matches without the need for backtracking. [24]

The complexity of the algorithm depends on the length of the patterns plus the length of the searched text plus the number of output matches. This algorithm has asymptotic worst-time complexity O(n+m) in space O(m).

5.4.1 Aho-Corasick Algorithm – Pseudo Code

Pattern matching machine
Input: A text string x and a pattern matching machine M with goto function g, failure function f, and output function output
Output: locations at which keywords occur in x.
begin
state ← 0
for i← 1 until n do
begin
while g(state, ai) = fail do state ← f(state)
state ← g(state, ai)
if output (state) ≠ empty then
begin
print i
print output(state)
end
end
end

5.5. The Brute-Force Pattern Matching Algorithm

The Brute-force pattern matching algorithm compares the pattern ***P*** with the text ***T*** for each possible shift of ***P*** relative to ***T***, until either a match is found or all placements of the pattern have been tried. Its time complexity is O(nm) [12]

5.5.1 The Brute-Force Algorithm – Pseudo Code

function BruteForceMatch(T, P, m, n)
Input text T of size n and pattern P of size m
Output starting index of a substring of T equal to P or -1if no such substring exists
for I ← 0 until n do
begin

```
j ← 0
while j < m and T[i + j]  = P[j]
begin
increment j by one
if j = m
return  i
else return  -1
end
end
```

5.6. Boyer–Moore–Horspool Algorithm or Horspool Algorithm

The Boyer-Moore algorithm uses two heuristics in order to determine the shift distance of the pattern in case of a mismatch. They are referred as bad-character and the good-suffix heuristics. The idea behind this algorithm is, instead of considering the bad character for mismatch, they consider the rightmost character of the current text window to determine the shift distance in each case. [13]

BMH is an algorithm for finding substrings in the given strings. It is a simplified form of the Boyer–Moore string search algorithm. The time complexity for an average case is of O(N) on random text, and O(MN) in the worst case.

5.6.1 Horspool Algorithm – Pseudo Code

Horspool (P = $p_1p_2\ldots p_m$,T = $t_1t_2\ldots t_n$)
preprocessing
for $c \in \sum$ do $d[c] \leftarrow m$
for $j \in 1\ldots m\text{-}1$ do $d[p_j] \leftarrow m - j$
searching
$pos \leftarrow 0$
while $pos \leq n\text{-}m$ do
$j \leftarrow m$
while $j > 0$ and $t_{pos+j} = p_j$ do $j \leftarrow j\text{-}1$
if $j = 0$ then report an occurrence at pos+1
$pos \leftarrow pos + d[t_{pos+m}]$
end of while

a	A	C	G	T
HpBc[a]	1	6	2	8

***Figure 6: HpBc* table used by Horspool algorithm**

5.7 Reverse Factor Algorithm

Most of the algorithms match the suffixes of the test pattern like Boyer-Moore string algorithm. It is possible to match the prefix of the pattern also by scanning from the right to left of the string. We can do this by using the smallest suffix automaton of the reverse pattern of the string. This methodology is called as Reverse Factor algorithm. This algorithm parses the character of the window from right to left with the help of the automaton in its searching phase. This process continues till there is no more transition defined for the current character in the current automaton. By this we can find the length of the longest prefix of the pattern, which is matching. Knowing the longest matching prefix's length, we can calculate how much the right shift is required. The time complexity for the preprocessing phase is O(m) and for the searching phase is O(mn). [12]

5.7.1 The Reverse Factor Algorithm – Pseudo code

Input: A text string T and a pattern string P with lengths n and m respectively
Output: All occurrences of P inT.
Set i to 1;
Step 1: if i+m-1 >n then exit.
else let W is equal to T(i, i+m-1) be a window.
find LSP(W,P)
if |LSP(W,P)| is equal to m,
report an exact match is found at ti and a is equal to m-|LSP(W,P)|.
else, set a=m-|LSP(W,P)|.
increment i by a
go to Step 1.

5.8 The Turbo-BM Algorithm

The Turbo BM algorithm is a corrective measure or improvisation of the Boyer-Moore algorithm. Here the preprocessing is same but there is extra space is added comparing with the original Boyer-Moore algorithm. In this algorithm, we remember the last time suffix pattern which was matching. The major two advantages are: possible to jump over factor and enable to perform a turbo-shift. The time complexity for the preprocessing phase is $\boldsymbol{O}(m+\sigma)$ and the time complexity for the search is O(n). [14]

5.8.1 The Turbo BM Algorithm – Pseudo Code

```
let I is equal to 0 and memory allocated is  nil;
while i ≤ n-m do
begin
align pattern with positions t[i + 1 . . i  + m] of the text;
scan the text right-to-left from the position  i + m,
using memory  to reduce number  of inspections;
let x'  be  the part  of the text scanned;
if  x  =  p  then  report  a  match  at  position  i;
compute the shift shift_i  according  to  x  and  memory;
take i as  i + shift_i;  update  memory  using  x;
end
end.
```

5.9 Boyer-Moore-Smith Algorithm

In Smith Algorithm just compute the shift with the text character just next the rightmost text character of the window. This operation gives sometimes shorter shift than using the rightmost text character of the window. The preprocessing phase of the Smith algorithm consists in computing the bad-character shift function and the Quick Search bad-character shift function. Smith will take the maximum between the values. [11]

The preprocessing phase is in O(m+σ) time and the searching phase has a quadratic worst case time complexity.

5.9.1 The Smith String Matching – Pseudo Code

```
void SMITH(character  x, integer m, character y, integer n)
begin
```

```
  initialize  j, bmBc[ASIZE], qsBc[ASIZE];
 /* Preprocessing */
  preprocess preBmBc(x, m, bmBc);
  preprocess preQsBc(x, m, qsBc);
  /* Searching */
  initialize  j = 0;
  while (j<= n - m)
  begin
 if (memcmp(x, y + j, m) is equal to 0)
 OUTPUT(j);
 Equal j to j + max of bmBc[y[j + m - 1]], qsBc[y[j + m]]
end
end
```

5.10 Quick Search

This is one of the fastest algorithms available for string matching and faster than Boyer-Moore algorithm. The time complexity of this algorithm is O(m+n). Quick search algorithm resembles Boyer-Moore algorithm while detecting the matches between two strings. Quick search algorithm uses only the bad-character shift table, but Boyer-Moore algorithm uses both bad-character table as well as good suffix shift table. This is the major difference between them. The quick search operation will not depend on scanning the string pattern in any particular order. [25]

5.10.1 Quick Search Algorithm – Pseudo Code

```
quick_search(P,T)
n = length (T)
m = length(P)
T' = T.P
bcp = precompute_bad_character (P)
gsp = precompute_good_suffix(P)
s = 0
while bcp(T'[s+m-1]) > 0 do
s = s + bcp(T'[s+m-1])
while s ≤ n-m do
j = m – 2
while j ≥ 0 and p[i] = T'(s+j) do
 j = j-1
if j < 0 then print (s)
s = s + gsp (j + 1)
while bcp (T'[s+m-1]) > 0 do
s = s+ bcp(T'[s+m-1])
end
```

5. 11 Backward Nondeterministic Dawg Matching algorithm

It is the variant of the Reverse Factor algorithm. It uses bit-parallelism simulation of the suffix automaton of x^R. This algorithm is efficient if the pattern length is no longer than the memory-word size of the machine. The automaton is simulated with bit parallelism even without constructing it. In this algorithm, for each character, associated tables B will be precomputed with a bit mask expressing its occurrences. [26]

5. 11.1 BNDM Algorithm – Pseudo Code

```
/*  Preprocessing */
begin
memset (B,0,arraysize *sizeof(integer))
initialize s to 1
for i = m-1 to i ≥ 0 do
begin
B(x(i)) = B(x(i)) or S
left shift s by one bit
end
/* searching phase */
Initialize j to 0
While j<=n-m
begin
  i=m-1;
  last=m;
d is not equal to 0
while(i>=0  and d !=0)
begin
   d EQUALTO d  and s(y(j+1));
    i=i-1;
 if (d != 0)
 begin
if i >= 0
 last=i+1;
else
 output(j);
end
left shift d by one bit
if end
 j=j+last
end
end
```

6. GLOBAL SEARCH ALGORITHM

As our concentration is on local search algorithms, we are listing out very few global searching algorithms for our comparison purpose. Here we took only two algorithms even though there are a huge variety of algorithms available in present day scenario.

6.1. The Needleman–Wunsch Algorithm

This algorithm takes two sequences A and B and performs global alignment between them. The main usage of this algorithm is in protein and nucleotide sequence alignment in bioinformatics. This algorithm is using dynamic programming technique. This is the first application for dynamic programming in biological sequence comparison. This algorithm is also known as optimal matching algorithm. The time complexity of this algorithm is O(nm). [27]

6.1.1 NW Algorithm – Pseudo Code

```
F : {1, 2, . . . , n} × {1, 2, . . . ,m} → R
in which F(i, j) equals the best score of the alignment of the
two prefixes (x1, x2, . . . , xi) and (y1, y2, . . . , yj).
Input: two sequences X and Y
Output: optimal alignment and score α
Initialization:
Set F(i, 0) := −i x d for all i = 0 to n
Set F(0, j) := −j x d for all j = 0 to m
For i = 1 to n do:
For j = 1 to m do:
Set F(i, j) := max ( F(i − 1, j − 1) + s(xi, yj), F(i − 1, j) – d, F(i,
j − 1) – d)
Set backtrace T(i, j) to the maximizing pair (i',j')
The score is α := F(n, m)
Set (i, j) := (n, m)
repeat
if T(i, j) = (i − 1, j − 1) print (xi, yj)
else if T(i, j) = (i − 1, j) print (xi, −)
else print (−, yj)
Set (i, j) := T(i, j)
until (i, j) = (0, 0).
```

6.2. Hirschberg's Algorithm

Hirschberg's algorithm is a dynamic programming algorithm that finds the least cost sequence alignment between two strings. The cost is measured as Levenshtein distance. The Levenshtein distance is defined as the sum of the costs of insertions, replacements, deletions, and null actions needed to change one string to the other. Hirschberg's algorithm is the modified version of Needleman–Wunsch algorithm by adding divide and conquers technique. The general use of Hirschberg's algorithm is in computational biology to find maximal global alignments of DNA and protein sequences. This algorithm is generally used for optimal sequence alignment technique. BLAST and FASTA are suboptimal heuristics. Hirschberg's algorithm takes O(nm) time, but needs only O(min{n,m}) space. This algorithm is a space efficient one to calculate the longest common sequence between the test string and the available database. [28]

6.2.1 Hirschberg's Algorithm – Pseudo Code

Hirschberg(x,y) is
n = length(x); m = length(y)
If n <= 1 or m <= 1:
OUTPUT Alignment(x,y) using Needleman-Wunsch.
else:

A. mid = floor(n/2)
B. x_left = Prefix[x,mid]
C. x_right = Suffix[x,n-mid]
D. Edit[x_left] = Forwards(x_left,y)
E. Edit[x_right] = Backwards(x_right,y)
F. cut = ArgMin{Edit[x_left,Prefix[y,j]] + Edit[x_right,Suffix[y,m-j]]}
G. Hirschberg(x_left,Prefix[y,cut])
H. Hirschberg(x_right,Suffix[y,m-cut])

7. EXPERIMENTAL RESULTS

In this chapter we will run the 9 most general local algorithms for different length of test strings and find the time taken by them to find matching.[29] The values are entered in terms of seconds. The graphical representation will help us to visualize it and got clearer picture of which one is better for which string length. Here as we know the DNA alphabet consists of the four nucleotides A, C, G and T (standing for adenine, cytosine, guanine, and thymine, respectively) used to encode DNA. Therefore, the set of alphabet used are O = {a, c, g, t}. In this example the test text is consisted of 9,97,650 characters and we search 50 different patterns of each length from 10 to 100 characters in steps of 10. The text and the patterns used were taken from the GenBank DNA database. Kindly refer table 1 for understanding the comparisons.

We can notice from the graph that, for the test pattern the running time of the Reverse Factor algorithm (RF) is efficient in practice for our small set of alphabets used in DNA sequence. Even though the time taken by the KR algorithm is constant throughout the operation, the overall time taken is more than the RF algorithm. So it is not best suited for our application. The good choice will be RF algorithm for DNA pattern matching algorithms. Kindly refer the figure 7 for our understanding.

8. APPLICATION OF SPEICIES CLASSIFICATION ALGORITHMS IN OTHER AREAS

Species identification and classification technique can be extended as disease identification, classification of diseases and patients accordingly. While doing so, the personal details of the patients, their disease details, and the treatment they are undergoing, etc will be accessed from their clinical databases. To keep the secrecy of the patients and their personal details, we need to protect them. To do so, privacy preserving is needed. [16][21] As privacy preserving the database is another huge area, we didn't touch that in this paper.

9. CONCLUSION

The paper aims to incorporate data mining concepts in the classification technique. The paper is planned in such a way that features are extracted from the DNA sequence of different species with the aid of data mining concepts. A supervised classifier will be developed using artificial Neural Network (ANN) technique and the classifier will be trained using the extracted features. Because of the training process, the classifier will be well-prepared to classify the DNA sequence of the different species. So, given a DNA sequence, the classifier can classify effectively by identifying the species class to which the sequence belongs. The planned methodology will be evaluated using DNA sequence of different species and its performance will be appraised. The methodology can be implemented in the MATALB platform and can be evaluated. It is expected that the methodology can classify the sequence on the basis of its species class with remarkable classification accuracy.

A set of algorithms for string matching on binary strings and encoded DNA sequences has been presented. The experimental results shows which local searching algorithm is best suited for the DNA sequence searching. The same searching algorithm can be extended further in medical field for identifying and classifying the diseases. The use of ANN will help us to speed up our process as the neurons are already trained from known data. The only job pending there is to compare them with the test data. How fast the comparison is done and how effectively the memory space is utilized will depend on which algorithm is used in implementing the comparison task. As most of the species DNA databases are huge and they will occupy a vast amount of memory space, the efficient algorithm both in time complexity and space complexity will be the best suitable one. In this paper, we took only the time complexity into consideration. The space complexity has to be explored and the combination of these two will be the best among the best.

10. REFERENCES

[1] The website and Glossary maintained by Amateur Entomologists Society

[2] Zoheir Ezziane, Applications of artificial intelligence in bioinformatics: A review, Expert Systems with Applications 30 (2006) 2–10

[3] Jiawei Han and Micheline Kamber, Data Mining: Concepts and Techniques, Second Edition, Morgan Kaufmann publications, 2006.

[4] Arthur Lesk, Introduction to Bioinformatics, Oxford University Press, USA, 3 Edition, 2008.

[5] Simon Haykin, Neural Networks a Comprehensive Foundation, Prentice Hall Publications, II Edition, 1998.

[6] NeuralWare (2003), NeuralWorks Predict® Getting Started Guide for Windows, Carnegie, PA.

[7] Dr Yeshpal Singh and Alok Singh Chauhan, "Neural Networks in Data Mining" Journal of Theoretical and Applied Information Technology, 2005 - 2009

[8] Kwong-Sak Leung, Ka-Chun Wong, Tak-Ming Chan, Man-Hon Wong, Kin-Hong Lee, Chi-Kong Lau, and Stephen K. W. Tsui, . Discovering protein–DNA binding sequence patterns using association rule mining, Nucleic Acids Res. 2010 October; 38(19): 6324–6337. Published online 2010 June 6.

[9] Felix Autenrieth, Barry Isralewitz, Zaida Luthey-Schulten, Anurag Sethi, Taras Pogorelov, Bioinformatics and Sequence Alignment, June 2005

[10] Thompson JD, Plewniak F, Poch O. (1999). "A comprehensive comparison of multiple sequence alignment programs". *Nucleic Acids Res* **27** (13): 2682–90.

[11] Boyer, Robert S.; Moore, J Strother (October 1977). "A Fast String Searching Algorithm.". *Comm. ACM* (New York, NY, USA: Association for Computing Machinery) **20** (10): 762–772.

[12] CROCHEMORE, M., HANCART, C., 1999, Pattern Matching in Strings, in *Algorithms and Theory of Computation Handbook*, M.J. Atallah ed., chapter 11, pp 11-1--11-28, CRC Press Inc., Boca Raton, FL

[13] HORSPOOL R.N., 1980, Practical fast searching in strings, Software - Practice & Experience, 10(6):501-506.

[14] Evolutionary Computation 2 - Advanced Algorithms and Operations, edited by Thomas Baeck, D.B Fogel, Z Michalewicz, Taylor & Francis; I edition, November 2000

[15] Timothy Masters, Advanced algorithms for neural networks: a C++ sourcebook, Volume 1, Wiley, I edition, Apr-1995

[16] Kiran P, S Sathish Kumar and Dr Kavya N P, A Novel Framework using Elliptic Curve Cryptography for Extremely Secure Transmission in Distributed Privacy Preserving Data Mining, Advanced Computing: An International Journal (ACIJ), Vol.3, No.2, March 2012

[17] Jianbo Gao, Yan Qi, Yinhe Cao, and Wen-wen Tung, "Protein Coding Sequence Identification by Simultaneously Characterizing the Periodic and Random Features of DNA Sequences", Journal of Biomedicine and Biotechnology, Vol. 2, pp. 139–146, 2005.

[18] Shital Shah and Andrew Kusiak, "Cancer gene search with data-mining and genetic algorithms, Computers in Biology and Medicine", Vol.37, No.2, pp.251-261, February 2007

[19] Riccardo Bellazzi and Blaz Zupan, "Towards knowledge-based gene expression data mining", Journal of Biomedical Informatics, Vol.40, No.6, pp.787-802, December 2007

[20] Fayyad, Piatetsky-Shapiro, Smyth and Uthurusamy, "Advances in knowledge discovery and data mining", AAAI/MIT Press, 1995

[21] Kiran P, Sathish Kumar S and Dr Kavya N P, An Extended Conceptual Modelling for ETL Processes in Privacy Preserving Data Mining, International Conference on Computing and Computer Vision (ICCCV 2012).

[22] Jianbo Gao, Yan Qi, Yinhe Cao, and Wen-wen Tung, "Protein Coding Sequence Identification by Simultaneously Characterizing the Periodic and Random Features of DNA Sequences", Journal of Biomedicine and Biotechnology, Vol. 2, pp. 139–146, 2005.

[23] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001-09-01). "The Rabin–Karp algorithm". *Introduction* (2nd ed.). Cambridge, Massachusetts: MIT Press. pp. 911–916.

[24] Aho, Alfred V.; Margaret J. Corasick (June 1975). "Efficient string matching: An aid to bibliographic search". *Communications of the ACM* **18** (6): 333–340

[25] 25 CROCHEMORE, M., LECROQ, T., 1996, Pattern matching and text compression algorithms, in *CRC Computer Science and Engineering Handbook*, A. Tucker ed., Chapter 8, pp 162-202, CRC Press Inc., Boca Raton, FL.

[26] NAVARRO G., RAFFINOT M., 1998. A Bit-Parallel Approach to Suffix Automata: Fast Extended String Matching, In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science 1448, Springer-Verlag, Berlin, 14-31.

[27] Needleman, Saul B.; and Wunsch, Christian D. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". *Journal of Molecular Biology* **48** (3): 443–53

[28] http://www.cs.tau.ac.il/~rshamir/algmb/98/scribe/html/

[29] lec02/node10.html

[30] P.D. Michailidis and K.G. Margaritis On-line String Matching Algorithms: Survey and Experimental Results, *International Journal of Computer Mathematics*, Vol. 76, No. 4. (2001), pp. 411-434

[31] E.W.T. Ngai, Yong Hu, Y.H. Wong, Yijun Chen, Xin Sun**,** *"The application of datamining techniques in financial fraud detection: A classification framework and an academic review of literature*" Decision Support Systems, Volume 50, Issue 3, February 2011, Pages 559–569

[32] Arzu Şencan Şahin, İsmail İlke Köse & Reşat Selba, *"Comparative analysis of neural network and neuro – fuzzy system for thermodynamic properties of refrigerants*" Applied Artificial Intelligence: An International Journal, Volume 26, Issue 7, 2012, DOI: 10.1080/08839514.2012.701427

Table 1: Number of character comparisons for a DNA alphabet

m	BF	KMP	BM	BMH	QS	BMS	TBM	RF	KR
10	1.346732	1.116931	0.301635	0.373474	0.356066	0.242178	0.299615	0.26149	1.000031
20	1.348057	1.109472	0.271363	0.39388	0.365869	0.246332	0.269731	0.154379	1.000009
30	1.335037	1.097344	0.231094	0.357166	0.332303	0.218819	0.229967	0.110341	1.000023
40	1.342449	1.103267	0.209129	0.349426	0.332189	0.215021	0.208034	0.087277	1.000018
50	1.34136	1.106258	0.223401	0.365487	0.351301	0.225939	0.222463	0.074325	1.000022
100	1.3453	1.106325	0.19552	0.362155	0.350235	0.212253	0.19623	0.05325	1
Average	1.343156	1.1066	0.23869	0.366931	0.347994	0.226757	0.237673	0.12351	1.000017

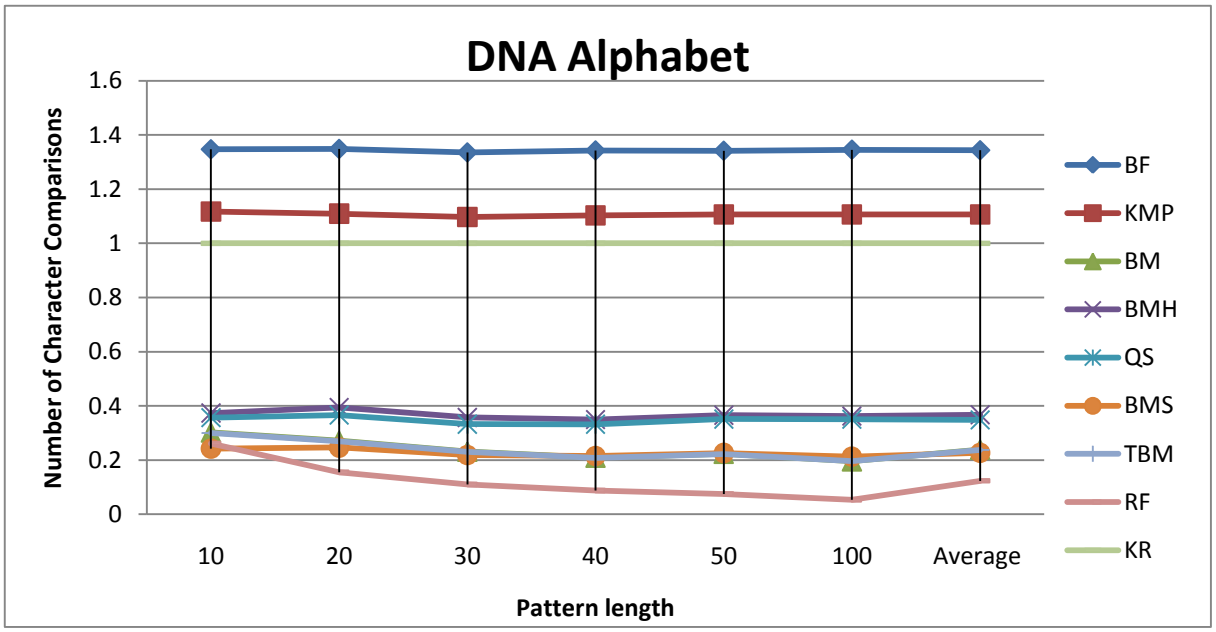

Figure 7: The graph for DNA alphabet