Adapting Decision Tree-Based Method to Index Large DNA-Protein Sequence Datasets

Khalid Mohammad Jaber, Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan
Rosni Abdullah, School of Computer Sciences, Universiti Sains Malaysia
Nur’Aini Abdul Rashid, School of Computer Sciences, Universiti Sains Malaysia

Currently, the size of biological databases has increased significantly with the growing number of users and the rate of queries where some databases are of terabyte size. Hence, there is an increasing need to access databases at the fastest possible rate. Where biologists are concerned, the need is more of a means to fast, scalable and accuracy searching in biological databases. This may seem to be a simple task, given the speed of current available gigabytes processors. However, this is far from the truth as the growing number of data which are deposited into the database are ever increasing. Hence, searching the database becomes a difficult and time-consuming task. Here, the computer scientist can help to organize data in a way that allows biologists to quickly search existing information. In this paper, a decision tree indexing model for DNA and protein sequence datasets is proposed. This method of indexing can effectively and rapidly retrieve all the similar proteins from a large database for a given protein query. A theoretical and conceptual proposed framework is derived, based on published works using indexing techniques for different applications. After this, the methodology was proved by extensive experiments using 10 data sets with variant sizes for DNA and protein. The experimental results show that the proposed method reduced the searching space to an average of 97.9% for DNA and 98% for protein, compared to the Top Down Disk-based suffix tree methods currently in use. Furthermore, the proposed method was about 2.35 times faster for DNA and 29 times for protein compared to the BLAST+ algorithm, in respect of query processing time.

Additional Key Words and Phrases: Indexing Approaches, Searching Algorithms, Decision Tree, DNA, Protein, Bioinformatics, Data Mining

1. INTRODUCTION

One of the major challenges for researchers of computer science, is the increase in the amount of biological and genomic data, with growing numbers of users and rates of queries; where some databases are of terabyte size. In real life situations for any given query data in a large database, the need to quickly retrieve all data which are similar is of utmost importance. With the ever growing number of data deposited in the database, searching becomes a difficult and time-consuming task. However, this paper is intended to help biologists to easily, quickly and effectively search existing biological data.

This paper will focus on the decision tree indexing algorithm to build indexes for enormous DNA-protein datasets to reduce the computation time for searching algorithm, as well as reducing the space required to do the computation and store indexes. However, the protein is the most prevalent bio-molecule. It consists of a long chain of amino acid sequences that fold into a complex four dimensional structure. Structurally, each amino acid is one of 20 molecules consisting of an amine, a carboxyl group, and a variable molecular side chain that determines the identity and chemical activity of the amino acid. The Deoxyribonucleic Acid (DNA) consists of a double stranded helix, close to each other. Each strand of the DNA consists of a sugar (Deoxyribose), and a base (adenine, thymine, cytosine or guanine). Each of the four characters (A, T, C and G) in each strand builds the code for the technology of human life. The strands are connected by hydrogen bonds between adenine and thymine (A, T) and between cytosine and guanine (C, G). Each of these relations is called a base pair [Westhead et al. 2002].

This paper is organized in seven sections. The Section 1 explains the protein database problems and the contribution of this paper. Section 2 defines the Workflow of adapting the decision tree for indexing, followed by the benchmark for the proposed method in Section 3. Section 4 presents the data sets for DNA and protein, used for the evaluation of the
indexing model. Section 5 presents the experiments and analyzes the results. Finally, the authors conclude and present future work in the last section 6.

2. ADAPTING THE DECISION TREE FOR INDEXING: FRAMEWORK

Database indexing searching is one of the approaches adapted to achieve fast, accurate and efficient searching. Indexing genomic searching is a method that reduces the cost and time of the searching [Williams and Zobel 2002]. In other words, the index method can efficiently solve such kinds of memory storage capacity problems by cutting down the memory storage requirement and access time [Williams 2003; Jiang et al. 2007]. In the absence of other better methods, the index method is necessary to solve the serious problems arising from the use of the exhaustive search methods [Williams and Zobel 2002]. The methods used for genomic indexing approaches are classified under three categories which are shown in Figure 1.

![Fig. 1. Classification of Methods for Genomic Searching Approaches](image)

However, we discussed these approaches in previous works [Jaber et al. 2010b; 2010a]. In general, The searching approaches have many drawbacks which could be classified under three categories:

1. Storage problem: the capacity of results produced by the searching approaches are larger than the original data.
2. Memory problem: this problem has relationship with the storage problem, in which case to read the large data located on the disk and try to fit it into the main memory causes
problems, since the main memory’s capability is much smaller than the capability of the storage disk.

(3) Poor accuracy: this problem is due to difficulties in finding the correct query solution. Large databases increase the size of the search space, which in turn increases the opportunity to choose the wrong solution.

In some approaches, for an example, the RAMdb requires a large index, about twice the size of the original flat file database (storage problem) [Jiang et al. 2007]. In FASTA and BLAST, all sequences must be in the main memory when a program executes, therefore large space is needed (storage and memory problems) [Williams 2003]. Inverted Files have suffered retrieval accuracy and as such are not very useful for small query proteins with few SSEs (poor accuracy) [Gao and Zaki 2008]. Suffix trees, is a possible data structure for indexing, but it can only be built for small sequences due to memory bottleneck (memory problem) [Farach et al. 1998; Tian et al. 2005]. Therefore, due to the limitations of the searching approaches and the limitation of searching in large genomic data sets, which is a time consuming task and due to computing complexity, the adapting decision tree based-method was proposed, to index data from large protein databases to address these problems. To the best of our knowledge so far, no research effort has used decision tree as an indexing approach for biological data. However, the decision tree was observed to have been used in different fields as the indexing approach, such as in the information retrieval [Quellec et al. 2007], video indexing [Shearer et al. 2001], graph database [Irniger and Bunke 2004; 2008] and others. As well, the decision tree has the ability to deal with large data sets, as several previous effort tried to use it in large data; such as SLIQ [Mehta et al. 1996], SPRINT [Shafer et al. 1996], CLOUDS [Alsabti et al. 1998]. Furthermore, decision tree can be constructed without any domain knowledge or parameter setting and it tree has good accuracy, speed, robustness, scalability and interpretability [Han and Kamber 2006]. However, to clarify the problem area, Figure 2 shows the main steps of the proposed framework in general. In subsequent subsections, the phases are described in detail which involves:

(1) The downloading of DNA-protein datasets from public databases.
(2) Pre-processing phase.
(3) Building the index using decision tree algorithms.
(4) Storing the indices.
(5) Searching DNA-Protein queries in the decision tree indices.

2.1. Download DNA-Protein Datasets from Public Databases

The first step in this research is to download DNA and protein datasets from genomic databases such as GenBank and Swiss-Prot (an example of DNA sequence is given in Figure 3). The sequences are downloaded and stored locally in a text file, and used in the next phase (pre-processing phase). Furthermore, it is used for the evaluation the indexing model.

2.2. Pre-Processing Phase

The aim of this phase is to prepare the representation of the data sets to store in the databases. This phase is carried out in three steps. The first step in this phase is to retrieve data from the text file by opening the file, then reading the data which it contains line by line. After that, the second step is to extract the description of the sequence, such as a unique library accession number (ID), family name and so on. Figure 4 shows an example of the description of the sequence.

The third step is to process and build efficient data representations which are suitable for SQL databases, to reduce the processing time and space required. The data sets are represented as a collection of tuples (S) that may contain duplicates. Tuples is known also
as records, rows, samples or instances. Each tuple is represented as a vector of attribute values ($A$). Attribute is known also as field, variable or feature. The data set provides the description of the attributes and domains. In this paper, data sets are denoted as $D(A \cup Y)$, where $A$ denotes the set of input attributes containing $n$ attributes: $A = \{a_1, a_2, ..., a_n\}$ as shown in matrix Equation 1, and $Y$ represents the target attributes.
The possible value of attribute $a_i$ is denoted as $pos(a_i) = \{v_{i,1}, v_{i,2}, ..., v_{i,|pos(a_i)|}\}$. The set of all possible values is defined as a Cartesian product of all input possible attributes: $X = \{pos(a_1) \times pos(a_2) \times ... \times pos(a_n)\}$. In a similar manner, $pos(y_q) = \{T_1, ..., T_{|pos(y)|}\}$ represents the possible values of the target attributes. The training set consists of a set of $m$ tuples. Formally, the training set is denoted as $T_{set}(D) = \{(x_1, y_1), ..., (x_m, y_m)\}$ where $x_q \in X$ and $y_q \in pos(y)$. The difficulty of this step is in the method of selecting target attributes, so adoption of the target data to depend on the protein family name for protein data, and to depend on the type of chromosomes for the DNA data was used. After building data representations in preprocessing phase, these data were stored in databases, to allow for multiple ways of accessing the stored data and make it searchable. For a particular file system however, there is a fixed path of accessing data and it is difficult to search into it.

2.3. Building the Indices

The first step in this phase was to initialize some parameters such as number of target attributes, the data type (DNA or Protein), name of database and name of tables, and so on. The next step in this phase is to adapt building decision tree for large data sets. Figure 5 shows the flowchart used to build the decision tree. The building algorithm has several steps, which are as follows:

1. The first step is to initialize the splitting criteria. The information gained; called Entropy is used as a measure to split the attributes and compute in units called bits. During this step, statistical method is used to calculate the information gain.
2. Select an attribute which has the highest Entropy, placed at the root node and stored into the database (loading and storing data issues are discussed in section 2.4).
3. Create one branch for each possible value $pos(a_i) = \{v_{i,1}, v_{i,2}, ..., v_{i,|pos(a_i)|}\}$.
4. Select one possible value $v_{i,|pos(a_i)|}$ from $pos(a_i)$. Then, calculate the Entropy and select the highest value to place at the branch node and store in the database. After this, the process could be repeated recursively for each branch, using only those instances which actually get to the branch.
5. If all the instances in the training set belong to a single target value, the particular node is not split further and the recursive process down that branch terminates.
6. If the maximum possible value of the tree is reached, to stop developing the tree. Else repeat steps 4, 5 and 6.

2.4. Hybrid Decision Tree Indexing Model with Database Management System Using Structured Query Language

Due to the limitations noted with the previous research which tried to fit large data sets into the memory before an induction decision tree such as Catlett method requires that all data are loaded into memory before induction [Sug 2005; Rokach and Maimon 2005] and [Mehta et al. 1996] uses a data structure which scales with the dataset size, this data structure must be resident in the main memory all the time [Shafer et al. 1996]. Furthermore, in previous indexing algorithms works, all data sets are resident on disks and during the building of the index, they will be moved onto the memory such as [Joshi et al. 1998; Zaki et al. 1999; Srivastava et al. 1999; Sreenivas et al. 1999; Jin and Agrawal 2003; Jin et al. 2005; Ben-Haim and Tom-Tov 2010]. And due to performance of the merged trees such as [Chan and Stolfo 1997] proposed the approach to divide the datasets into many portions,
then each portion is loaded alone into the memory and used to induction a decision tree. After this, every decision tree is merged, to create a single tree. And due to the size of the storage indexing problem, the hybrid Decision Tree Indexing Model (DTIM) with Database Management System (DBMS) using SQL is proposed in the current study, to address the memory bottleneck and storage problems. However, substantial research efforts utilizes SQL data mining techniques to improve the scalability [Sattler and Dunemann 2001]. Some further examples of investigated SQL techniques are presented in [Sarawagi et al. 1998; 2000] for association rules, in [Ordonez and Cereghini 2000] for clustering and in [Slezak and Sakai 2009] for decision rule. This section discusses the hybrid method from many, different perspectives; such as connecting the proposed method with the database, loading the data for induction and data representation for storing the new indexes which were produced by the decision tree indexing model process. In the next section, the description of the searching query with the decision tree index is made in greater details.

2.4.1. Connect DTIM with Databases. Connection of the decision tree indexing model with DBMS has several benefits. For example, it allows for multiple ways of easy access, organization, management and retrieval of data stored in the database and makes it searchable via asking of simple questions in a query language. As for the particular file systems, there are fixed paths for accessing data which are slow, difficult and expensive to search and which are resident among all data sets in the main memory during processing computation.
To create a connection between the current proposed method and DBMS, latest connectors were used for MySQL, developed by Sun Microsystems. The MySQL connector for C++ provides an object-oriented API and a database driver for connecting C++ applications to the MySQL Server. Therefore, the main components for establishing the connection are the Driver, Connection, Statement, PreparedStatement, ResultSet and ResultSetMetaData.

Figure 6 presents the algorithm for establishing a connection to the DBMS by retrieving an object of sql::Connection from the sql::Driver object. An sql::Driver object is returned by the sql::Driver::get_driver_instance() method and sql::Driver::connect() method returns the sql::Connection object. However, the first parameter in the connection object is the TCP/IP, which is used to connect the system with DBMS server by specifying "tcp://[hostname][:port]/[schemaname]" for the connection URL (Figure 6 lines 14 and 25). Specifying the schema name in the connection URL is optional, and which if not set, then there is the need to select the database schema using Connection::setSchema method (Figure 6 line 28).
To prevent memory leaks and to reduce the memory usage, the dynamic memory was used, with the class template auto_ptr as shown in Figure 6 lines 25, 31 and 33. An auto_ptr object has the same semantics as that of a pointer. However, when it goes out of scope, it automatically releases the dynamic memory which it manages. In other words, when auto_ptr template class is in use, there is no need to free up the memory explicitly using the delete operator; e.g., delete con;

More highlight in respect of the hybrid DTIM with DBMS, there is no need to load all of a huge data set into the main memory for induction decision tree. Thus connection is created, then simple questions are asked using SQL, after which the results are retrieved, automatically closing the connection.

2.4.2. Indexes Data Representation. An efficient data representations for storing the indexes is proposed, which was produced by the decision tree indexing model process, to minimize the indexing size. Usually, the results produced by the decision tree are stored as a rule, which has many drawbacks such as the following [Zhou et al. 2003b; 2003a]:

1. Generated rules are often very complex than is necessary and contain redundant information.
2. Storage problems for large data and inefficient speed.
3. Memory bottleneck problems which need to fit all the rules in the memory to do the search.

On the other hand, the proposed data representation has several advantages as follows:

1. Efficient way to insert, delete and retrieve the search tree elements.
2. Minimizes the number of database queries by having only one query for each activity.
3. Minimizes the storage size.
4. Is generated efficiently and easily.
5. Is a solution to the memory problem, with no need to fit all indexes into the memory.

The indexes could be represented by five attributes which are: ID, Parent, Node, Element and Decision as shown in Figure 7. The ID represents the unique identification number for each node, Parent (P) represents the father ID for each node. The Node (N) represents a position of an element, depending on the sequence, Element (E) represents a set of attributes (A); where attribute (A) contains n attributes: A = \{a_1, a_2, ..., a_n\}. The possible value of an attribute a_i is denoted by \text{pos}(a_i) = \{v_{i,1}, v_{i,2}, ..., v_{i,|\text{pos}(a_i)|}\}. Finally, decision represents the target attribute (y) or the next step in the tree; where the possible value of (y): \text{pos}(y_i) = \{T_1, ..., T_{|\text{pos}(y)|}\}.

To clarify the proposed index representation, Figure 8 represents an example of the indexes representation. Figure 8 consists of two parts; the first part on the left represents the tree from the root node a_5 and one branch element A, until the last level of the indexes. The second part on the right is the indexes data representation. In the first instance, ID = 0 the parent P = −1, meaning that it is the root node, thus the decision is a_1, meaning that the next step is node = a_1 with parent = 0.

<table>
<thead>
<tr>
<th>ID</th>
<th>Parent (P)</th>
<th>Node (N)</th>
<th>Element (E)</th>
<th>Decision (y)</th>
</tr>
</thead>
</table>

Fig. 7. Indexes Data Representation
2.5. Search DNA-Protein Queries in the Decision Tree Indices

The fifth phase of this experiment involves the searching process. This involves using the indexes which were stored in the previous phase. Therefore, the searching process of the work could be summarized in the following steps:

*Step 1.* Enter the query (Q) or queries.

*Step 2.* Tokenize the query, called query chopping.

*Step 3.* Search the query in the decision tree indices, using SQL command.

*Step 4.* Save the result or display it.

From the review of previous methods, it was noticed that some of the popular indexing methods do not support the search for multiple queries, called Query multiplexing or packing such as CAFÉ [Williams and Zobel 2002], Top Down Disk-based (TDD) suffix trees [Tian et al. 2005; Tata et al. 2004]. To address this problem, the hybrid in the proposed method is equipped with a query packing, to handle multiple queries and reduce the overhead of reading the queries repeatedly.

Additionally, to enhance the searching speed, the query chopping technique was adapted to merge with the proposed method. Query chopping is dividing the individual query sequences into several segments, searching them independently and then merging the results back together. After adapting the query chopping for the proposed method, the query chopping definition is thus to divide individual query sequence into several segments, use particular segments to search them dependently and then show the result. Figure 9 presents the query sequence, which is divided into five segments and the segments needed for the query process only are \( a_5 \), \( a_1 \) and \( a_2 \), whereas segments \( a_3 \) and \( a_4 \) are not necessary in the query search process. Therefore, supposing there is a query length of 80 nucleotides and this uses only 50 nucleotides in the search process; therefore the use of particular nucleotide increases the speed of the search instead of the search process with all the 80 Nucleotides.

Before moving on to the next section, following is a clarification of an adapted decision tree (DTIM) compared to a normal decision tree. The DTIM has many improvements (advantages) such as the preprocessing phase, where the preprocessing phase is done in three steps. The important step in preprocessing phase is to retrieve and clean the data, while this

---

**Fig. 8.** Example of Indexes Data Representation
step is not existing in the normal decision tree algorithms. Moreover, the normal decision tree data representations are stored in a text file as a storage which causes a problem in memory and difficulties in processing and building the tree—especially for large data. While the DTIM data representation is stored in DBMS, which is fast and better to handle large data. Furthermore, the normal decision tree cannot deal with DNA-protein data directly for the reason that the DNA-protein is same as English text, and it does not have a target value, while the DTIM can deal with DNA-protein data due to the preprocessing phases. The DTIM is connected with DBMS instead of the file system which is available in the normal decision tree. However, the results produced from the normal decision tree are rules, whereas the results of the adapted decision tree are records in a table; the discussion of advantages and disadvantages is provided in subsection 2.4.2. Finally, the normal decision tree has no concept of query multiplexing and packing, while it is available in DTIM.

3. BENCHMARK
To evaluate the DTIM method, different tests were performed as follows: first, DTIM method was compared with the Top Down Disk-Based suffix tree [Tata et al. 2004; Tian et al. 2005], to test the indexing size. Secondly, DTIM method was compared with BLAST+ version 2.22.22, downloaded from [NCB 2010] to test the query processing time. Finally, computing the performance of the DTIM is tested by accuracy, recall, precision and elapsed time for building the indexes.

4. TEST DATA SETS
In the experiments, different sets of ten DNA sequences were used, extracted from human chromosomes 18, 19 and 21 with varying sizes from 10 thousand to 5 million bases, which were downloaded from GenBank. Furthermore, other different data sets of ten protein sequences were used with varying sizes from 10 thousand to 5 million bases, extracted from Swiss-Prot database. As query sequences, some subsequences were randomly extracted from the data sets.

5. EXPERIMENTS AND RESULTS
This section describes the experiment performed and the result of the experiment on the decision tree indexing model (DTIM). The first experiment conducted was to compare the indexing size between DTIM and TDD (Top Down Disk-Based suffix tree) download from http://www.eecs.umich.edu/tdd. The experiment was performed with two different data types; DNA and protein data. The reason for using this is that each nucleotide from DNA consists of 4 characters, whereas the protein data consists of 20 amino acids, which is more complicated. The sizes of ten data sets range from 10 thousand to 5 million bases.

The second experiment performed was to compare the DTIM with BLAST+, in term of query processing time. The number of queries was similar to the number of deposit sequences in the databases, to show the effects of performance of the proposed method.
The third experiment was to examine the time speed for building an index on one processor. The final experiment was to quantify the relative performance or retrieval effectiveness of the proposed method, used to measure accuracy, precision and recall.

5.1. Indexing Size Experiments

Figure 10 shows the DTIM index sizes for nucleotide data sets, while Figure 11 is for amino acid data sets, with changing data sizes. Furthermore, the current research approach was compared with the Top Down Disk-Based suffix tree approach in respect of the index size for both DNA and protein. The Top Down Disk-Based is a previous approach based on the suffix tree [Tian et al. 2005; Tata et al. 2004].

From the experimental results, it was observed that the index size increases linearly in proportion to the data size in both approaches. However, in comparison with TDD, the current approach saves an average of about 97% of storage space for DNA and 98.2% for protein data. In comparison, in the DNA-Protein data with original data sets, it was observed that the DTIM reduced the size of the data significantly. For example, the DTIM reduced 99% of the size of data set number 10, which contained 5 million nucleotides, whereas the size of the original data is about 5M while the size of indexing about 550K. This is enough evidence which supports that the proposed DTIM is appropriate for large data sets.

![Fig. 10. Index Sizes for DNA Data Sets](image1)

![Fig. 11. Index Sizes for Protein Data Sets](image2)
5.2. Query Processing Time

In this experiment, DTIM was compared with BLAST+ version 2.22.22, in respect of the elapsed time for query search. The elapsed time is the time spent on finding all the queries, which are submitted to the DTIM or BLAST+. The effects of variant data sets sizes was also examined, with changing number of queries on DTIM and BLAST+ methods.

Figures 12, 13 respectively show the DNA and protein queries processing time of DTIM and BLAST+ for exact match queries. The X axis denotes a varying data sets size, while the Y axis the elapsed time in unit of seconds. From this experiment results, it was observed that the query processing time increased linearly in proportion to the data size in both approaches (DTIM and BLAST+) as shown in Figures 12, 13. However, in Figure 12, it was observed that the BLAST+ achieves 1.48 faster than the DTIM when the DNA data set size is greater than 500K. On the other hand, DTIM shows good performance when DNA data set size is less than 500K and achieves about 2.35 times of query processing time compared with BLAST+. This is because of the BLAST+ methodology, where it examines every possible alternative to find a particular solution. In other words, BLAST+ needs to compare a query with each of the sequences in the database, depending on the sequence alignment techniques. In the case of the proposed approach, the data sets which are less than 500K is somewhat small in size, hence the BLAST+ completes the search quickly. However, when the size of data sets increased to more than 500k, DTIM achieved about 2.35 times speedup for the DNA and 29 times speedup for the protein, compared with the BLAST+ in terms of query processing time, as clarified in Figure 12 and 13. This is because of two reasons. The first was mentioned previously, in respect of the BLAST+ methodology. Secondly, is the combination of query packing and query chopping techniques with DTIM, to increase the speed in the process of retrieval queries. For example, DTIM packed 1412 queries together when it searched in the 100K and 72753 queries together when it searched in 5M.

5.3. Building Index Time

In this experiment, the time taken to build the index on one processor was calculated, with changing data sets size for the two groups of data; DNA and protein as shown in Table I. The aim of this experiment was to examine the time cost of building the index for the current approach DTIM.

The results shows that the required time to build the index increased exponentially in proportion to the data size. However, the required time to build 100K DNA data is about 72 seconds while for 100K protein data is 105 seconds. Compared with the 500K DNA data,
which is about 2531 seconds while in 500K protein data is about 1118 seconds as shown in Table I. Hence, this indicates that a large data takes a long time to build the index and the DTIM approach is not practical for large data sets. To improve the performance of DTIM in order to minimize the required time to build the index, the parallel version of DTIM will propose as future work.

### Table I. Building Index Time using One Processor

<table>
<thead>
<tr>
<th>Data</th>
<th>Size</th>
<th>Time (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNA</td>
<td>100k</td>
<td>72.2857</td>
</tr>
<tr>
<td></td>
<td>200k</td>
<td>265.76</td>
</tr>
<tr>
<td></td>
<td>300k</td>
<td>741.135</td>
</tr>
<tr>
<td></td>
<td>400k</td>
<td>1704.87</td>
</tr>
<tr>
<td></td>
<td>500k</td>
<td>2531.77</td>
</tr>
<tr>
<td>Protein</td>
<td>100k</td>
<td>105.103</td>
</tr>
<tr>
<td></td>
<td>200k</td>
<td>314.421</td>
</tr>
<tr>
<td></td>
<td>300k</td>
<td>542.843</td>
</tr>
<tr>
<td></td>
<td>400k</td>
<td>852.972</td>
</tr>
<tr>
<td></td>
<td>500k</td>
<td>1118.05</td>
</tr>
</tbody>
</table>

### 5.4. Evaluating DTIM

In this experiment, the retrieval effectiveness of DTIM was examined, using the measure of accuracy, precision and recall. Before doing this, the holdout method was used as shown in Figure 14. In this method, the data set was randomly partitioned into two independent sets; a training set and a test set [Han and Kamber 2006]. Typically, two-thirds of the data are allocated to the training set, and the remaining one-third are allocated to the test set. The training set is used to produce the index data, whose accuracy, precision and recall are estimated with the test set using confusion matrix. The confusion matrix is a useful tool for analyzing the DTIM, to calculate the accuracy, recall and precision.

From experimental results, it is observed that the accuracy of DTIM for DNA is slightly low but according to [Doolittle 1986] it is acceptable because when two amino acids or nucleotides are more than 30% identical, then the sequences are most likely homologous. However, low accuracy was achieved for many reasons; the important one is with regards to the sampling method (Holdout), which selected the training set randomly, leading to a pessimistic estimate because only a portion of the initial data was used to derive the mode.
Probably, this portion was not sufficient to learn the mode to achieve good accuracy. On the other hand, DTIM performs better accuracy in the protein data. The average overall accuracy for the ten data sets are about 90.48% for protein while for DNA data, it is about 55.89%. This is also enough evidence to support that the proposed DTIM is appropriate for large data sets in terms of accuracy.

Figures 15, 16 illustrates the comparison between average precision training test and average precision of testing set for DNA and protein respectively, whereas Figures 17, 18 illustrate the comparison between average recall training set and average recall testing set for DNA and protein respectively. The precision values for training set was mostly 100% in both DNA and protein, which imply that no false positive values were produced by DTIM. Testing sets for DNA produces the most false positives while testing set for protein produces a very small number of false positive. However, the DNA recall in the Figure 17 is better than the precision of testing data for DNA in the same figure, implying that DTIM for DNA produces less false negative. The recall values for protein testing set are above 79%, indicating that the false negative produced is low.

Overall, DTIM for protein performs better than DTIM for DNA. However this also supports that DTIM is appropriate for complicated data sets such as protein (20 amino acids) in terms of precision and recall.

6. CONCLUSION
Database indexing for larger biological data has become an important task in bioinformatics. The main objective for indexing is to reduce the computation time searching algorithm, as well as reducing the space required to do the computation and store indexes. To achieve
Fig. 16. Average Precision for Protein

Fig. 17. Average Recall for DNA

Fig. 18. Average Recall for Protein
this, the decision tree for indexing the large DNA-Protein data was adapted from many perspectives such as data representation, connecting the DTIM with DBMS to store the indexes which were produced from the model, loading data from DBMS to do the induction for DTIM, search in DTIM, workflow, benchmark, data sets, system requirement and so on. After this, the methodology was proved by extensive experiments using 10 data sets with variant sizes for DNA and protein.

From the experiment, it was observed that the proposed DTIM reduces the indexing space an average of 97.9% for DNA and 98% for protein compared with TDD and performs the queries in lower query processing time is about 2.35 times faster for DNA and 29 times for protein compared with BLAST+ algorithm. Furthermore, the experiment examined the time cost of building the index for DTIM and shows that the required time to build the index increased exponentially in proportion to the data size and therefore, to improve the performance of DTIM in order to minimize the required time to build the index, the parallel version of DTIM will be proposed in future work. Finally, it could be concluded that DTIM performed better accuracy, precision and recall for protein data set than DNA data set in the ten data sets tested. However, low accuracy of DNA data was derived, the accuracy of which needs to be improved in future work.

REFERENCES


