A Filtering Algorithm for Efficient Retrieving of DNA Sequence

M Nordin A Rahman, M Yazid M Saman, Aziz Ahmad and A Osman M Tap

Abstract—DNA sequence similarity search is an important task in computational biology applications. Similarity search procedure is executed by an alignment process between query and targeted sequences. An optimal alignment process based on the dynamic programming algorithms has shown to have O(n

m) time and space complexity. Heuristics algorithms can process a fast DNA sequence alignment, but generate low comparison sensitivity. The biologists frequently demand for optimal comparison result so that the perfect structure of living beings evolution can be constructed. This task becomes more complex and challenging as the sizes of public sequence databases get very large and are increasing exponentially each year. The aim of this study is to develop a filtering algorithm in order to reduce the iteration of dynamic programming process and therefore an efficient process of retrieving a set of similar DNA sequences in database can be made. The algorithm filtered the expected irrelevant DNA sequences in database from being computed for dynamic programming based optimal alignment process. An automaton-based algorithm is used to develop the filtering process proposed. A set of random patterns is generated from query sequence are placed in automaton machine before exact matching and scoring process is performed. Extensive experiments have been carried out on several parameters and the results show that the developed filtering algorithm removed the unrelated targeted sequences from being aligned with query sequence

Index Terms—Exact string matching, Aho-Corasick algorithm, sequence comparison, Smith-Waterman algorithm.

I. INTRODUCTION

A routine operation for biologists is to query a new discovered sequence against a collection sequences databases such as GenBank and EMBL to find a list of similar sequences. DNA sequence comparison is among the fundamental tools in computational molecular biology.

Basically, this process is used for determining the similarity (or distance) between sequences. The obtained similarity degree can be used to infer the functionality or evolution history of the sequences. The best possible way to achieve the optimal result of similarity between sequences is using Smith-Waterman algorithm [12] which based on dynamic programming method and running in quadratic time complexity O(n×m). The Smith-Waterman algorithm is likely inefficient should it applied to a large-scale sequence similarity search process. With the rapid growth of public sequence databases, modern biologists rely on tools that could detect the similarity between sequences efficiently and at the same the optimal alignment results are retained. Therefore, a specific approach is needed to remove the highly expected irrelevant targeted DNA sequences (sequence database) from being performed for alignment process.

Another important algorithm considered in computational biology applications is exact string matching processing. String matching has broad applications; for instance in bibliographic search, lexical analysis, web search engines and recently as a filtering purpose for DNA sequence searching [6][3]. Both single pattern and multi-patterns are stressed in string matching application domain. Single pattern matching problem consists of finding all occurrences of a pattern with length a specific length, n, in a text with particular length, m. Meanwhile, the multi-patterns string matching problem consists of finding all occurrences of patterns in P with total length n in a text with length m. One of the most excellent exact multi-patterns string matching algorithms is Aho-Corasick algorithm [1]. By means of multi-attribute patterns matching, [3] has successfully improved the Aho-Corasick algorithm. This algorithm used automaton theory that can be implemented by using directed tree data structure. Using this automaton structure, the algorithm efficiently can scan all occurrences of any patterns in a targeted text string. Aho-Corasick algorithm calculates pre-processing phase where all the patterns are placed in a keyword trees. The algorithm uses the failure links to speed up matching and it can be calculated in linear time. The time complexity of this algorithm is O(n) for preprocessing time and O(m + k) for search time, where m is the targeted text length, n is total patterns length and k represents the number of occurrences of patterns in the text.

The filtering technique has been utilized to enhance the DNA sequence comparison speed. The technique is used in BLAST [2], FASTA [11] and PatternHunter [9][10] algorithms. Those algorithms utilize the idea to focus only on the regions, which share some patterns and assume those patterns to have potential similarity. However, this filtering

technique may miss a group of potential regions that have high homologous features between the compared sequences. This paper proposes a new model that attempts to eliminate the irrelevance targeted DNA sequences from being executed for optimal local alignment. Using a linear runtime multi-patterns exact string matching algorithm, a set of random patterns (subsequences) from query sequence is scanned to the all targeted sequences in database. The targeted DNA sequences that have a significant low of exact matching score are removed from being executed for dynamic programming based alignment process.

The rest of the paper is organized as follows: The next section reviews some previous works in filtering based method for computing DNA sequence similarity search. Section II discusses the detailed of the filtering algorithm for effusion retrieving of DNA sequence. Experimental results of the proposed model are presented in Section III. Conclusion is placed in Section IV.

A. Related Works

Smith-Waterman algorithm is one of the techniques that use dynamic programming in computing homology level between DNA sequences. The algorithm compares every base in the evaluated sequences to produce a precise local alignment. Therefore, this technique is very slow should applied to a large-scale DNA sequence similarity search problem. In order to improve the efficiency, many ideas have been proposed such as filtering method. Filtering process is trying to discard irrelevant subsequences in both compared sequences from being executed for rigorous Smith-Waterman algorithm. BLAST [2], FASTA [11] and PatternHunter [9][10] are the common tools that use filtering approach for computing approximate sequence similarity search.

BLAST and FASTA are introduced at the end of the 1980s and both rely on the so-called hit-and-extend heuristic, which can be implemented using hashing and lookup tables [4]. FASTA and BLAST are faster than Smith-Waterman algorithm because they examine only a portion of the potential alignments between two sequences. Ideally, BLAST looks for small identical patterns in both sequences and try to extend in both direction of the pattern until the obtained score is lower than a given threshold. National Center for Biotechnology Information (NCBI) has introduced a web based system to BLAST that allows biologists from over the world to query their new discovered genomic sequences against the public sequence databases such as GenBank and EMBL. The steps in BLAST algorithm is depicted in Figure 1.

Steps in Basic Local Alignment Search Tool (BLAST)

1) *Generate seeds* - break a query sequence into a set of contiguous regions. The old version of BLAST uses of consecutive 11 nucleotide bases

2) *Scanning the database* – search seed that have perfect match or similarity score greater than a threshold value in the targeted sequence in database.

3) *Extend the seed matching* – from both sides, seed perfect matching is extended into a significant value length (the score by X below its maximum value). This step is also called alignment step. Gaps insertion is allowed during extension process. A standard scoring system is used to establish the alignment; +1 is given for a match and −3 for a mismatch residue. Although BLAST may use other scoring system such as PAM or BLOSUM.

Figure 1: The steps in BLAST tool

FASTA identifies some common short patterns called ktup, carefully chosen. During the comparison process, an offset table is updated, reporting the number necessary shifts for ktup belonging to both sequences to be aligned. Finally, the offset table contains offsets that highlight the best position(s) for achieving good alignments. FASTA tool is freely used and can be accessed at European Bioinformatics Institute (EBI) website. The detailed steps executed in FASTA algorithm can be viewed as in Figure 2.

Steps in Fast Homology Search All Sequences (FASTA)

1) *Look for hot spots* – find initial regions in query sequence. A lookup hash table is used to scan a *ktup* match characters between query and targeted sequence

2) *Find ten best diagonal runs* – rescore to find top ten initial regions using such as PAM250 scoring matrices. The score of the diagonal run is the sum of the hot spots scores and the inter-spot scores. The best scoring initial region is given as the *init-1* score

3) *Apply "joining threshold"* – non-overlapping regions should be joined. The sum of the scores for each individual region minus a joining penalty for each gap is calculated. This sum score is given as the *init-n* score and then it would used to rank all the sequences in database

4) *Perform for local alignment process* – full Smith-Waterman algorithm is applied to the highest scoring sequences.

Figure 2: The steps in FASTA tool

A commercial PatternHunter is a new generation general-purpose homology search tool that employed heuristic method. The PatternHunter discovers short word matches under spaced form. A spaced form is represented as a binary string of length l, where a 1 bit at a position means that a base match is required at the position and a 0 bit at a position means that either a base match or mismatch is acceptable at the position [9][10]. PatternHunter looks for

runs of eighteen consecutive nucleotide bases in each sequence. From these eighteen consecutive nucleotide bases only eleven nucleotide matches are required according to the 1s in the string 111010010100110111. This string (or pattern) is also known as spaced seed. This method significantly increases the hit number of homologous region while reducing bad hits. Consequently, PatternHunter is able to find more homologous regions than BLAST and FASTA as well. In an experiment, PatternHunter performs at a speed over hundred times faster than BLAST at the same sensitivity [10]. PatternHunter uses a variety of advanced data structures for handling data manipulation in its algorithm specification. Those data structures include queues, priority queues, hash tables and a variation of red-black tree.

II. THE MODEL

DNA sequence similarity search process becomes more complex and challenging as the sizes of public sequence databases get very large and are increasing exponentially each year. Generally, the scopes of this research can be specified as studying, designing and implementing a filtering algorithm for performing optimal DNA sequence searching model efficiently. Therefore, the developed solution model can be formalized as follows: *Let T = {t₁, t₂, ..., tₖ} be a collection of DNA sequences from a set of databases and q is a query DNA sequence; let q be a fixed score threshold and F be an alignment scoring function. By means of an optimal local alignment that using dynamic programming based technique, A, of (q, tᵢ), find R Í T where " rᵢ Î R has score F(A) ³ q.* There are five phases involved in the proposed model. These phases include *Query initialization, Patterns generating, Patterns scanning, Ranking* and *Optimal local alignment.* Figure 3 shows the general flow of five main phases in the model.

The query is a new DNA sequence discovered by biologist. Using a random algorithm, a set of random patterns with length l characters are generated from q and can be denoted as, P = {ρ₁, ρ₁, ..., ρ_δ}. The number of patterns *(d)* that can be generated may varies which give different patterns *scanning* results. However, for the first attempt the model decides to use the following formula to calculate the number of patterns used:

$$d = 0.8 * q_L \qquad (1)$$

where q_L is query length. There are two conflict factors that are created during exact patterns scanning. Sensitivity of exact matching will be increased if many patterns are used. However this condition will decrease search speed. Reducing the number of patterns will increase the search speed but it can reduce the comparison sensitivity. This circumstance also happens in PatternHunter II similarity search tool. To increase the sensitivity in homology search, PatternHunter II increases the number of seeds and reduces the weight of a single seed [10]. However, those actions increased the runtime because the search engine will generate more random hits.
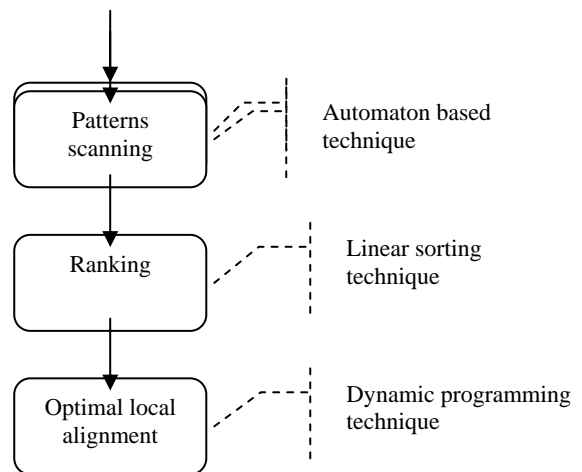

Figure 3: The model process flow

Hunting for a pattern in a targeted DNA sequence is much easier if the size (length) of pattern is sufficiently small. The small patterns will be assured that appear in many locations in a targeted DNA sequence. Although there will be many different size of small patterns, the model considered nine different pattern sizes (five to 13 characters) to be experimented. Figure 4 shows the general algorithm for generating random patterns from a query sequence. The algorithm generates the patterns based on a set of random integers, RN = {rn₁, rn₂, ..., rn_δ}. *rnᵢ* indicates that the pattern *i* is started at array index *rn* in targeted sequence.

Algorithm: *generate random patterns*

```
Input:      1) query, q
            2) Number of patterns, δ
            3) Pattern size, λ
            3) A set of random integers, RN = {rn₁,
                        rn₂, ..., rn_δ}
Output: A set of patterns, P = {ρ₁, ρ₁, ..., ρ_δ}.

L1.  δ ← 0.8 * q_L
L2.  Generate, RN
L3.  nop ← 0
L4.  for i ← 1 to δ
        L5.  patt ← " "
        L6.  for j ← (rnᵢ+1) to (rnᵢ+1 + λ)
                L7.  patt ← patt + q[j-1]
        L8.  ρᵢ ← patt
        L9.  P ← P + {ρᵢ}
        L10. nop ← nop + 1
L11. return P
```

Figure 4: Algorithm for generating random patterns

Figure 5 exhibits eight patterns with length, λ = 6 generated from a query sequence. The patterns can be denoted as, P = {"AAATGA", "TTGCCC", "GCCCTA", "TACATC", "CACCAG", "AAACAT", "CGAGGG", "GTCCAA"}. Since the starting index of a pattern is

Query initialization — Initialize by user

randomly generated, the patterns may overlap. Some patterns will share the same contiguous characters. However, allowing overlapped patterns will help to increase the sensitivity of ranking process. The creation of overlapped patterns gives a set of various patterns with the same some contiguous characters. After patterns have been generated, the model will read the DNA database. The retrieved DNA sequences are temporarily stored into computer memory before patterns scanning is started.
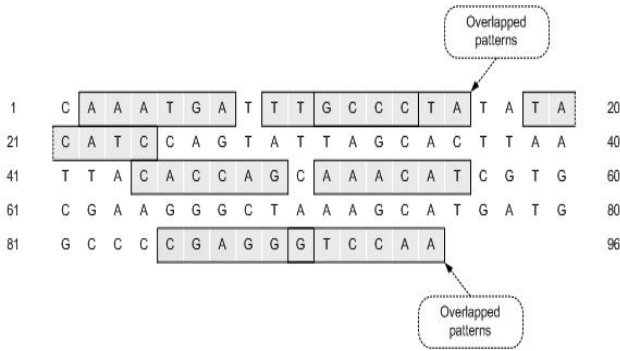

Figure 5: Generating the patterns from a query sequence

Let, $P = \{\rho_1, \rho_2, \ldots, \rho_\delta\}$ is the set of generated patterns from a query. The model treats each $\rho_i \in P$ as being distinct even if there are multiple copies of it in P. The overlapped patterns will help to increase the sensitivity of similarity degree comparison between query and targeted sequences. The proposed model assumes that sequences with highest exact matching score are the most likely potential to be similar with the query sequence. The next process is to scan all $\rho_i \in P$ that are identical to the given t. Sometimes it is called as exact set matching problem. This process is done by using Aho-Corasick algorithm search technique.

The Aho-Corasick algorithm search uses a keyword tree to store the series of patterns. The DNA sequence alphabet size is relatively small and has a fixed size. The size of this nucleotides alphabet is four. Therefore, the constructed keyword tree will not have many root branches (edges). This minimal number of edges from a root makes exact patterns matching process becomes more simple and efficient. Theoretically, if an alphabet is a fixed size, then to construct the keyword tree for P is in O(n) time, where n is the total length of all patterns. In this study, the problem is to determine if a substring in targeted sequence from database, t_i, completely matches some string in P. By means of this condition, the utility of a keyword tree is clear. The patterns are encoded into a keyword tree (g), and when an individual targeted sequence from database is presented, a walk from the root of g determines if there are substrings in t_i are in the g.

Using this keyword tree, each targeted DNA sequences in database is processed in a single pass. Figure 6 illustrates the algorithm used to search the query patterns in targeted DNA sequences in the database. From the theorem given by [1], if P is a set of patterns with total length n and t is a DNA sequence from database of total length m, then one can find all occurrences in t of patterns from P in O(n) preprocessing time (to build keyword tree) plus O(m + k) search time, where k is the number of occurrences. Once the scanning

process is completed, the exact matching score for each pattern found will be calculated.

The choice of a scoring function that reflects biological or statistical observations about known sequences is important to producing good alignments. This model used BLOSUM62 [7] or PAM250 [5] scoring matrices for assigning exact matching score to the processed DNA sequences. The found pattern is given exact matching score which can be calculated by:

$$\pi(\rho_i, t_a) = s(\rho_i[0], t_a[j]) + s(\rho_i[1], t_a[j+1]) + \ldots + s(\rho_i[\lambda-1], t_a[j+\lambda-1]) \qquad (2)$$

At this point, s is similarity score between two appropriate characters and j is the initial index location where pattern is found in t_a. Once scanning all $\rho_i \in P$ in a t is completed, the total exact matching score for t is computed, and represented by:

$$\varepsilon(t_a) = \sum_{b=1}^{d} e(r_b, t_a) \qquad (3)$$

Finally, the model will have a set of total exact matching score and can be denoted as, $Y = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k\}$. The general algorithm for this scoring process is depicted in Figure 7.

```
Algorithm: pattern scanning
Input:      1) keywords tree
            2) DNA sequences database, T = {t₁, t₂ …, tₖ}
Output: The matching status for each pattern, patternFound

// construct automaton machine
L1. goto function
L2. failure function

// search ∀ρᵢ in t = b₁b₂b₃… bₙ and calculate for total exact
matching score t, ε(t)
L3. for a ← 1 to k do
        L4. for i ← 0 to δ
            L5. patternFoundᵢ ← 0
        // perform Aho-Corasick search
        L6. for i ← 0 to sequenceLength
            L7. a ← bᵢ
            // use goto function
            L8. while (g(state, a) = null)
                L9. state ← f(state)
            L10. state ← g(state, a)
            // process output
            L11. pi ← 0
            L12. while (not end of output state)
                L13.    patternFoundₚᵢ ← 1
L14. return patternFound
```

Figure 6: Pattern scanning algorithm

Algorithm: Exact matching scoring

Input: 1) patternFound status for each pattern

2) Set of patterns, P = {ρ₁, ρ₁, ... , ρ₅}

2) Scoring matrix, BLOSUM62

Output: Exact matching score for a targeted sequence, ε

L1. totalScore ← 0

L2. for i ← 0 to δ

L3. if(patternFound$_i$ = 1)

L4. for j ← 0 to λ

L5. score ← getScore(ρ$_i$,

BLOSUM62)

L6. totalScore ← totalScore +

score

L7. ε(t) ← totalScore

L8. return ε(t)

Figure 7: Calculate the exact matching score algorithm

To minimize the total cost of retrieval the most expected similar DNA sequences, we need to sort the sequences in descending order based on its exact matching score. The model expected that the most similar DNA sequences to the query are to be placed at the top of ranking and therefore optimal alignment process can directly focus on this group. An efficient sorting algorithm Quick-Sort has been used to complete this task. The runtime of this algorithm is O(n log n). Typically, Quick-Sort is significantly faster in practice than other $O(n \log n)$ algorithms, because in the most architecture its inner loop can be efficiently implemented.

The ranking simplifies to evaluate uncertain information in DNA sequence record according to certain criteria. In general, this ranking process is not guaranteed for highly expected similar DNA sequences to a query are positioned at top of the ranked list. Therefore, dynamic programming based techniques such as Smith-Waterman algorithm is still required to compute accurately the degree of similarity between two DNA sequences. The ranking will just make the next stage in the model possible quicker to select the DNA sequences for computing the local alignment. Figure 8 illustrates the order of targeted DNA sequences after ranking process.

t₁ ACCATTTCGCGTTAAATCGCG

t₂ TTTCGGCTATATATCGGCT

t₃ GGGCCTCTTAAATTCTCCTC

t₄ GGCTCCCTATATCTCTCGGAGAGA

t₅ CCGGATATAGCCTATT

t$_{k-2}$ AAAAATTCTCTCCGGGCTATAT

tk₋₁ CGCTCTTTTCTCTGGAGAG

t$_k$ CCCGGGATCTCTCTAGGCGCGA

Degree of expected similar to be decreased

Figure 8: A ranked DNA sequences

In order to obtain optimal local alignment, we implemented the Smith-Waterman algorithm that uses bi-dimensional arrays. Moreover, to guarantee the good scoring factor, BLOSUM62 (or PAM250) scoring matrix has been applied. The implemented Smith-Waterman algorithm is designed to support constant gap penalty function. The constant gap penalty function gives each gap the same score no matter how long it is. Figure 9 depicts the detailed of Smith-Waterman algorithm which is implemented in the model. If Dq and Dt are the best local alignment with length L produced from an alignment process, then the alignment score between Dq and Dt can be denoted as:

$$\Omega(\Delta q, \Delta t) = \sum_{i=0}^{L} s(\Delta q[i], \Delta t[i]) \qquad (4)$$

where s is similarity score between $Dq[i]$ and $Dt[i]$ that suggested by BLOSUM62 or PAM250 scoring matrices. The percent identity formula for the aligned of sequences Dq and Dt is given by the following equation:

$$I(\Delta q, \Delta t) = \left(\frac{R}{L}\right) \times 100 \qquad (5)$$

where R is number of matching residues after alignment process. Let S be a set of optimal alignment score and C be a set of identity score. If A is denoted as optimal local alignment process, then the computation output of optimal alignment score and identity can be represented as:

$$A(q, t_i) = \{\Omega_i, I_i\} \qquad (6)$$

where $\Omega_i \in S$ and $I_i \in C$.

Algorithm: Smith-Waterman

Input: 1) Query sequence, q

2) Targeted sequence,t

3) Scoring matrices e.g. BLOSUM62

Output: Alignment best scoring, bestScore

bestScore ← 0

// compute for the query and targeted sequence length

L1. q$_L$ ← count for the q bases

L2. t$_L$ ← count for the t bases

// initialize the base condition

L3. for i ← 0 to q$_L$

L4. for j ← 0 to t$_L$

L5. F$_{ij}$ ← 0

// recurrence condition

L6. for i ← 1 to q$_L$

L7. for j ← 1 to t$_L$

L8. choice1 ← 0

L9. choice2 ← F$_{i-1,j-1}$ + similarityValue

L10. choice3 ← F$_{i-1,j}$ - gapPenalty

L11. choice4 ← F$_{i,j-1}$ - gapPenalty

L12. F$_{ij}$ ← maximum(choice1, choice2, choice3, choice4)

L13. if F$_{ij}$ > bestScore

L14. bestScore ← F$_{ij}$

L15. Return bestScore

Figure 9: Smith-Waterman algorithm

III.   EXPERIMENTAL RESULTS

We use Java 2 Platform Standard Edition Version 1.5.0 in our project implementation and Microsoft Windows XP Professional as operating systems. The Java programming language and its environment is designed to solve a number of modern and complex problems. It is a mature language, ready for widespread used and integrated with other environment and architecture as well. The implementation is undergoing object oriented software development approach. The processor used was 1.8GHz Intel Pentium IV with 640MB RAM.

Figure 10 illustrates a state diagram that shows the detailed behavior of operations in the model. The diagram shows the instance changes state depending on the message(s) that it receives. The diagram splits the states in the model into five main categorizes: initialization state, exact matching state, ranking state, alignment state and reporting state. Initialization state consists of three sub-states: read query, generate patterns and read database. Exact matching state includes the Aho-Corasick algorithm operations: construct automaton machine, Patterns searching (scanning) and Exact matching scoring.   Ranking state serves only for one sub-state known as Ranking. Alignment state consists of Smith-Waterman algorithm operations: Optimal local alignment and Alignment scoring.   The final state is Reporting which consists of Generate reports. All the related reports to sequence similarity are generated within this sub-state.

In order to obtain realistic experimental results, five GenBank databases have been downloaded from *Bio-Mirror* website [14]. These databases are maintained by National Center for Biotechnology Information (NCBI). The databases correspond to *viral* sequences and contain 345,383 sequences with 335,979,422 base pairs (bp).   There are 30,000 sequences are selected and used in the experiments. This process is done by picking up the first 6,000 sequences from each database. Query sequences are selected randomly from these 30,000 selected sequences.   The model also utilized the BioJava library [13] to manipulate and extracting genome database from GenBank. Pattern length (λ) is used as a parameter to evaluate the performance and reliability of the developed model. The model considered percent identity, I, is equal to 30% as a threshold or a confident value for accepting the targeted DNA sequences that have similarity to the query.   Throughout the performance evaluation, the model attempts to retrieve 10% sequences in T that are satisfies the threshold value ($\theta = 3,000$).For each query, the experiments used various pattern lengths which are preset between five to 13 bases. Figure 11 shows the experimental results for those 10 queries. The figure represents the number of SW algorithms iterations versus different of λ. For the queries such as Query 1, Query 2, Query 3 and Query 4, the number of targeted DNA sequences to be processed under SW algorithm are very consistent with pattern length between five to 12 bases.   However, the number of targeted DNA sequences involved in optimal alignment processed is increased when the pattern lengths used are equal or greater than 13 characters.   Table 1 exhibits the experimental result from ten queries and λ is preset to ten characters.   The experimental results show that the proposed filtering mechanism can discard irrelevant DNA sequences from

being executed for rigorous Smith-Waterman algorithm. The implemented filtering technique successfully generated a group of DNA sequences from databases that have a highly potential similarity to the query sequence.
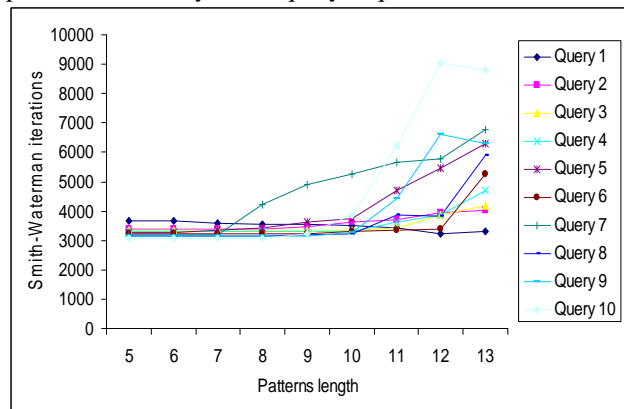

Figure 11: Number of Smith-Waterman algorithm iterations versus patterns length

TABLE 1: SMITH-WATERMAN ITERATIONS DISCARDED ($\lambda = 10$)

Query No.	No. of Smith-Waterman performed	Smith-Waterman iterations discarded (%)
Query 1	3488 (488)	88.37
Query 2	3617 (617)	82.94
Query 3	3350 (350)	88.83
Query 4	3325 (325)	88.92
Query 5	3756 (756)	87.48
Query 6	3318 (318)	88.94
Query 7	5243 (2243)	82.52
Query 8	3245 (245)	89.18
Query 9	3282 (282)	89.06
Query 10	3880 (880)	87.07

For instance, Query 2, the model successfully retrieved θ sequences that are definitely relevant to query after executing the optimal local alignment process to 3,617 targeted DNA sequences. Based on the percent identity (I) threshold value, there are 617 sequences from the group are not relevant to the query and are executed for Smith-Waterman algorithm. Therefore, 26,383 targeted DNA sequences (82.94%) have been skipped from the exhaustive local optimal alignment process. Meanwhile, for Query 10 there are 3,880 DNA sequences from the database have been performed for optimal local alignment process with 880 sequences are determined as irrelevant to the query. The result exhibits that 87.07% or 26,120 DNA sequences from the database have been discarded from rigorous Smith-Waterman algorithm.

Obviously, the model yields high efficiency processes for optimal DNA sequence similarity search with low computational numbers for O(n × m) time complexity Smith-Waterman algorithms. Therefore, the time taken for retrieving a set of similar sequences from databases to a query is been minimized. The execution analysis of the proposed model can be summarized as follows:

- For any query sequences, the patterns of length 13 characters and above will cause inefficient filtration process. The best possible pattern lengths that could be used and most appropriate with all queries in various

lengths are seven to12 characters. This pattern length range size will produce a good quality of ranking;

- If the pattern length is less than six bases used in the experiment, a bad quality ranking is produced and therefore a huge number of Smith-Waterman algorithm iterations will be executed for the targeted DNA sequences in the database. In other words, the related targeted DNA sequences will not placed at appropriated position before Smith-Waterman algorithm is executed;
- When $l \geq 13$, many patterns cannot be found in targeted sequences and consequently, exact matching score value will be 0. This situation will produce a bad ranking quality and;
- If query length is too short, for example below 500 characters, the most appropriate pattern length should be used are 10 to 12 characters. These sizes will give a good quality ranking position output.

The quality of the proposed filtering technique is evaluated by Average Normalized Rank (ANR) [8]. The ANR describes the quality of the whole ranked list of DNA sequences after filtering process. Generally, the ANR can be defined as:

$$\alpha = \frac{1}{NN_w} \sum_{i=1}^{i=N_w} (R_i - \frac{N_w + 1}{2}) \qquad (7)$$

where N is the number of targeted objects in the database, N_w is the number of wanted targeted objects from the database and R_i is the rank of each wanted targeted object in the database.

Clearly, N_w can be referred to q. Furthermore, this precise measurement is also shows the consistencies of the filtering quality. As defined by [10], a has a value of 0 when the wanted targeted objects have all been sorted on top. The a has value close to 0.5 if targeted objects are randomly shuffled in the list. If the targeted objects are all sorted at the bottom then the a value is closed to 1. From the experiments, we can perceive that all the required targeted DNA sequences are sorted on top of the list (below than 10,000 out of 30,000). As a result, if all the required variables are placed in the defined ANR metric and calculations are executed, then all the a values will become close to 0. This condition is true to all pattern lengths exclusive of query length which is too short such as below than 500 characters. However, this problem can be fixed if pattern lengths are set to 10, 11 or 12 characters. As a result, the proposed technique has successfully generated a precise and consistent DNA sequence ranking.

IV. CONCLUSIONS

Generally, it is reasonable to align two sequences by using classical Smith-Waterman algorithm when the sequences length is not very long. However this approach is not applicable for large-scale DNA sequence comparison that requires for more time and spaces complexity. A mechanism to discard the unrelated or irrelevant sequences compared to a query is highly demanded. To address the outlined problem, we customized automaton-based multi-patterns exact string matching that acts as a filtering process. The technique rapidly filters all those targeted DNA sequences that are likely irrelevant to the query. Only the relevant sequences are considered for execution in Smith-Waterman algorithm process. In other words, the proposed filtering technique can reduce the size of dataset. As a result, the processing time for retrieving a set of targeted DNA sequences, which are similar to the query, can be minimized.

The ranking result produced by the model has been compared to the BLAST tool. The model has successfully generated a good ranking result as produced by the BLAST tool. The proposed model has applied Smith-Waterman algorithm to the whole bases in the compared DNA sequences, and therefore it will generate a complete alignment structure. Hence, it can produce better alignment structure compared to BLAST tool which is examined only a portion of the potential alignments between the compared DNA sequences. In order to improve the performance of the model, clustering the database can be considered as an immediate action. The experiment shows that two or more DNA sequences can have the same exact matching score and positioned at the same ranking. This condition infers that those DNA sequences may be included in the same 'cluster'. Hence, to improve the efficiency of the model a further database classification can be done.

REFERENCES

[1] Aho, A. V. & Corasick, M. J. "Efficient String Matching: An Aid to Bibliographic Search", Communication of the ACM, Vol. 18(6), 1975, 333 – 340.

[2] Altschul, S. F., Gish, W., Miller, W., Myers, E. and Lipman, D. J. "Basic Local Alignment Search Tool", Journal Molecular Biology, Vol. 215, 1990, 403 – 410.

[3] Ando, K., Kinoshita, T., Shishibori, M. & Aoe, J-I. "An Improvement of the Aho-Corasick Machine", Information Sciences, Vol. 111, 1998, 139 – 151.

[4] Csuros, M. and Ma, B. "Rapid Homology Search with Two-Stage Extension and Daughter Seeds", Proc. of 11th International Computing and Combinatorics Conference, 2005, 104 – 114.

[5] Dayhoff, M. O., Schwartz, R. M. & Orcutt, B. C. "A Model of Evolutionary Change in Proteins". In Atlas of Protein Sequence and Structure, 5(3), Dayhoff, M. O. (ed.), 1978, 345 – 352.

[6] Gusfield, D. Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1997.

[7] Henikoff, S. & Henikoff, J. G. "Amino acid substitution matrices from protein blocks". Proceedings of National Academic Science USA, 89, 1992, (pp. 10915 – 10919).

[8] Huang, M., DeMenthon, D., Doermann, D. & Golebiowski, L. "Document Ranking by Layout Relevance". Proceedings of the 8th International Conference on Document Analysis and Recognition, 2005, (pp. 362 – 366).

[9] Li, M., Ma, B., Kisman D. & Tromp, J. "PatternHunter II: Highly Sensitive and Fast Homology Search" Genome Informatics, Vol. 14, 2004, 164 – 175.

[10] Ma, B., Tromp, J. and Li, M. "PatternHunter: faster and more sensitive homology search", Bioinformatics, Vol. 18(3), 2002, 440 – 445.

[11] Pearson, W. R. and Lipman, D. J. "Improved Tools for Biological Sequence Comparisons", Proc. of the National Academy of Sciences of the USA, Vol. 85, 1988, 2444 – 2448.

[12] Smith, T. F. and Waterman, M. S. "Identification of Common Molecular Subsequences", Journal of Molecular Biology, Vol. 147, 1981, 195 – 197.
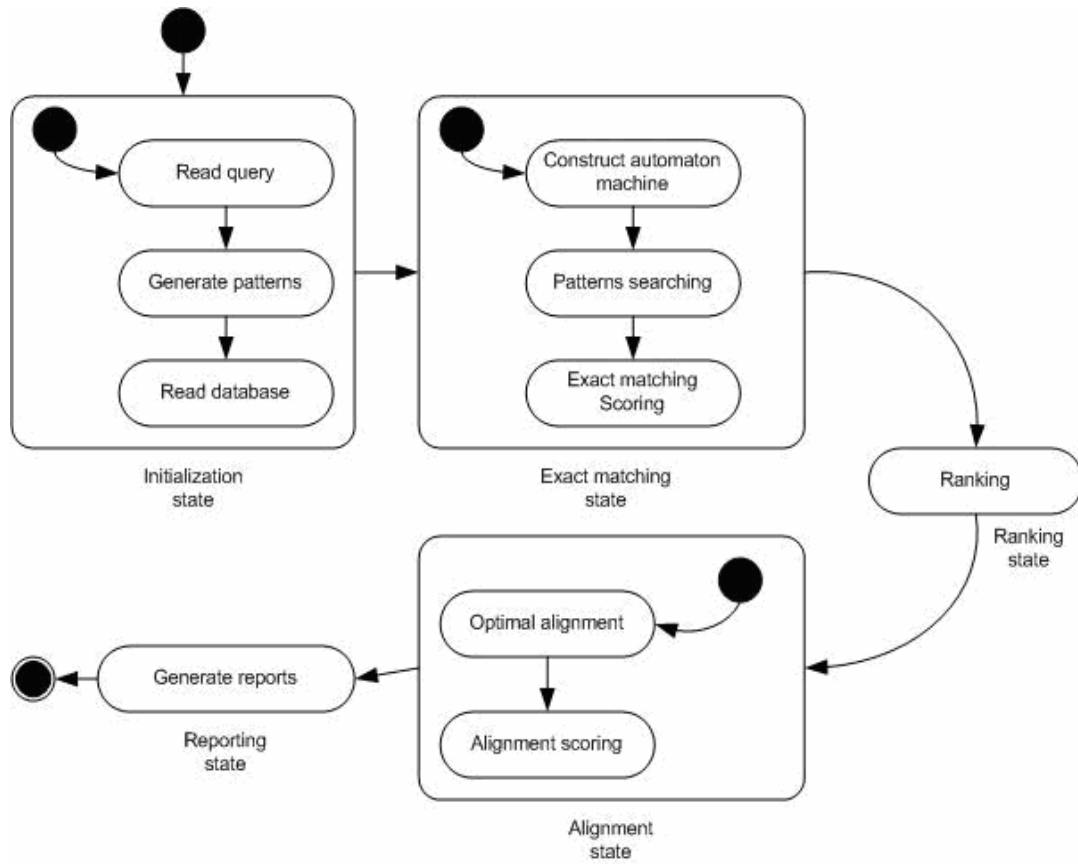
[13] BioJava Library Website, http://biojava.org/wiki/main_page

Figure 10: The model state diagram