Fast and Efficient Hashing for Sequence Similarity Search using Substring Extraction in DNA Sequence Databases

Robinson Silvester .A
National Bureau of Agriculturally Important Insects, Bangalore, Karnataka, India.

J. Cruz Antony
National Bureau of Agriculturally Important Insects, Bangalore, Karnataka, India.

M. Pratheepa, PhD
National Bureau of Agriculturally Important Insects, Bangalore, Karnataka, India.

ABSTRACT
Emergent interest in genomic research has resulted in the creation of huge biological sequence databases, however search and retrieval of relevant information from these databases takes a lot of processing time, when performed conventionally as size of databases containing DNA sequences is huge. Hence, providing an efficient searching mechanism is mandatory. In this paper we present an efficient search mechanism using Hashing techniques. Initially, the data is hashed and indexed according to different window sizes. During this process, we eliminate redundancies and only record patterns with distinct elements and provide them with corresponding hash values. During the search phase, the search string is checked for the size of the window and if it exceeds the maximum limit of 4, then it is divided. The first part is considered as the search string and the search is made. After the confirmation of the index, the strings that follow the current indexed string are matched with the search string and finally the confirmation is made. The simulation results show that the current methodology provides faster results, while occupying lesser memory.

Keywords
Hashing; Sequence Similarity by Hashing; Substring Extraction; DNA Sequence

1. INTRODUCTION
Deoxyribonucleic Acid (DNA) contains genetic information specifying the biological development of all cellular forms of life. This information is encoded in sequences of nucleotides within the DNA molecules. DNA contains four types of nucleobases, *viz*., Adenine (A), Cytosine (C), Guanine (G) and Thymine (T) **[1]**. Each of DNA sequence in organisms is formed by thousands or millions of these bases arranged in random order. In recent years, the importance of storing this information has been realized and currently there are two standard formats for storing these information into database, which includes the FASTA and GenBank flat file format. Conventional search and retrieval of data from these databases takes a lot of time as the size of these databases are huge. Sometimes, a similarity search may require several hours or days to complete. Hence, it is very important to develop an efficient search mechanism.

Many algorithms have been developed for sequence matching, the most fundamental one being Naïve String Matching algorithm **[2]**, which is the simplest and least efficient way of searching a string inside another string. Another string matching algorithm is the Boyer-Moore algorithm, like the string matching with finite automation **[2]** which does preprocessing of the pattern to allow the faster searching. Based on these, Kalsi et al. **[3]** performed an experimental comparison of the most efficient algorithms for searching biological sequences. In addition in **[4], [5]** Faro and Lecroq presented an extensive evaluation of (almost) all existing exact string matching algorithms under various conditions, including alphabet of four characters and DNA sequences. Navarro and Raffinot presented a comparison **[6]** of all matching algorithms on biological sequences, including multiple pattern matching algorithms. More recently, Faro and Lecroq conducted another experiment on fast searching in biological sequences using multiple hash functions **[7]** and also D.Nassimi and M.Joshi developed a hash based scalable technique for parallel bidirectional search **[8]** taking into account the most recent solutions. Though both the algorithms provides an efficient way of hashing, the drawback is that the time taken for indexing and searching when the strings of longer sequences are used.

Basically a string matching algorithm uses a window to scan the text **[9]**. The size of this window is equal to the minimal length of a pattern in the set of patterns. It first aligns the left end of the window and the text, thereafter it checks if any pattern in the set occurs in the window (this specific work is called an attempt) and then shifts the window to the right **[10]**. It repeats the same procedure until the right end of the window goes beyond the right end of the text. The best algorithms for searching DNA sequences are based on filtering methods. Specifically, instead of checking at each position **[10]** of the text if each pattern in the set occurs, it seems to be more efficient to filter text positions and check only if the contents of the window looks like any pattern in the set.

When a resemblance has been detected a naive check **[11]** of the occurrence is performed. In order to detect the resemblance between the pattern and the text window, efficient algorithms use bit-parallelism or character comparisons **[12]**. Both techniques can be improved by using condensed alphabets and hashing.

The rest of this paper is organized as follows. Section 2 provides overall system architecture, Section 3 describes the hashing technique in detail, Section 4 provides the results and discussions, and Section 5 provides the conclusion of the current work.

2. METHODOLOGY

2.1 System Architecture

The technique of substring extraction is subdivided into two phases. The first phase deals with reading the actual data *D* and preparing the indices *I*. The index table helps in finding the substring **[7]** for the user. The indexing mechanism is carried out in three parts using different window sizes each time. For each window size *w*, the substring *S* from *D* is considered. *S* is eliminated of all redundancies and the value **[13]** of the final pattern is recorded.

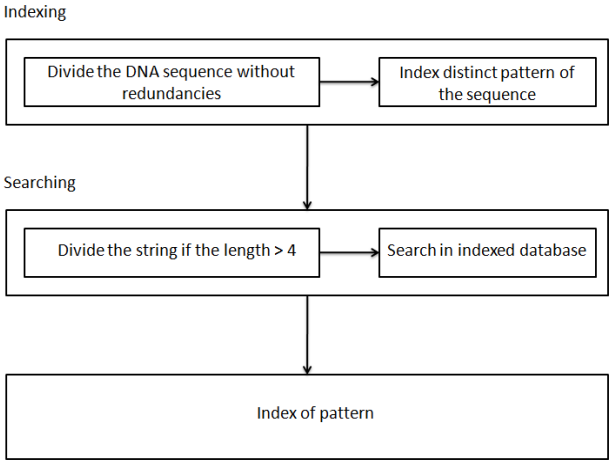

Figure 1: System Architecture for Indexing and Searching of DNA sequences

The second phase deals with searching for a given string (*Sub*) in the data. The length of *Sub* is checked and if it is found to be greater than 4, then the string is divided into *pre* and *post.* These represent the initial and the final portions of *Sub. Pre* and *post* values are used as the search strings to find the index value of *Sub.* Figure 1 represents the entire system architecture of the indexing and searching mechanism.

2.2 Fast and Efficient Hashing for Sequence Similarity Search using Substring Extraction in DNA sequence databases

As described earlier, DNA sequences are formed of four nucleobases, A, C, G and T. Hence, our database is restricted to these four characters, which occur repeatedly and in random orders. Indexing of these data is performed prior to the extraction process. The indexed table serves as the base for the second phase, i.e. the substring extraction phase.

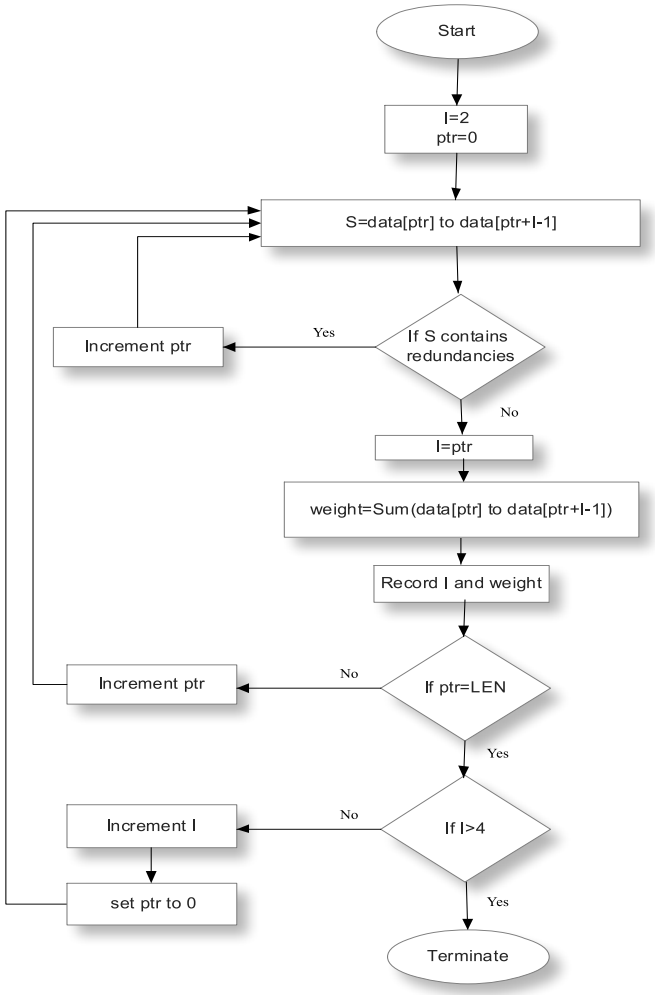

Figure 2: Indexing of DNA Patterns

Each base in the DNA sequence is assigned a numerical value, which helps to recognize the pattern. Here, we assign the values 0, 1, 2 and 3 to A, C, G and T respectively.

The indexing phase begins by initially setting the window size *w*. The window sizes that are considered are *w= {2,3,4}*. Considering the window value to be 1 is the same as searching the entire database *D1*, hence 1 is eliminated. Since we use only four bases for representing DNA sequences, considering a value greater than 4 will apparently lead to redundancies. Our basic processing involves removal of redundancies; we consider only three windows for our processing. The initial window size is set and the data *D* is scanned. The set of strings that come under the window *sub* are taken and checked for redundancies. If redundancies have been detected, then, the current *sub* is ignored and the window is moved to the next *sub*. If the pattern is found to be distinct, then, its corresponding numerical values are added *(weight)* and are recorded along with the index *I*. The window is then moved to the next base for redundancy check and *weight* calculation. This process is repeated for all the window sizes and the index database is prepared. The indexing phase is represented in Figure 2.

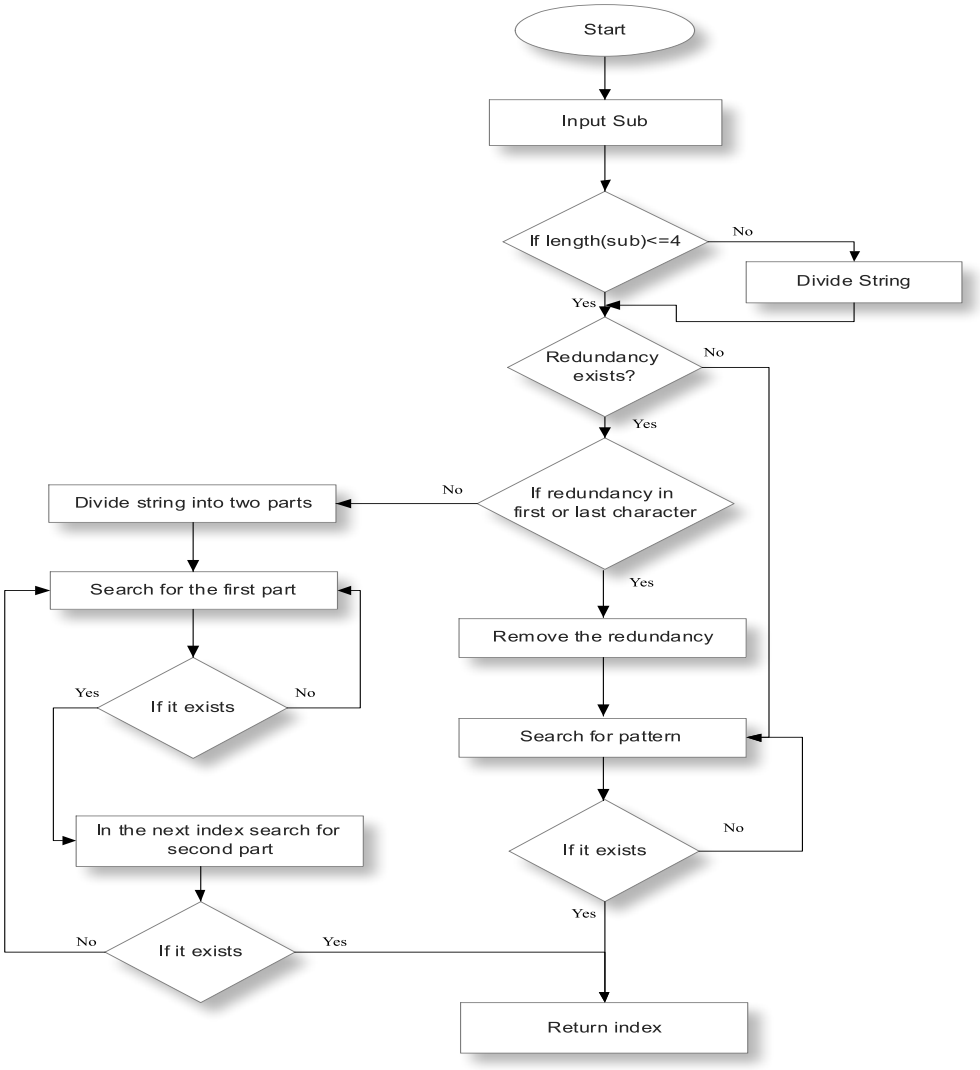

Figure 3: Substring Division and Search

The search phase can further be sub divided into two phases, namely, searching and division. The length of the substring (*sub)* is examined. If it is found to be greater than 4, then the division phase is initiated. Else *sub* passes through the usual search process. In the division phase, *sub* is divided into two parts. The initial part called the *pre* contains the first 4 characters of the substring. The remaining string is divided into string of 4 characters and is stored in the *post* array **[8]**. *Pre* is then passed to the search process for further processing. In the search phase, the presented string is checked for redundancies. If the redundancies occur at the beginning or at the end of the string, then it is removed and the new string without redundancies is obtained. The location of redundancies and the character that is redundant is recorded. The non-redundant string is searched for, in its corresponding window depending on the current length. If the pattern is found, redundancies are verified by checking the previous and the next index values in the same window and the resultant index *(RI)* value is returned. If *sub* has been divided into *pre* and *post*, then the post values are verified by checking the preceding index values of *RI* in the current window. A special case occurrence is the redundancy of all the characters in *sub* (Eg. AAAA, CCCC). This usually returns only one character after the redundancy check. This search is performed in the window of the actual size of *sub*. *RI* is found by examining the consecutive records that have an index gap equal to the length of the string – 1, i.e.(length(*sub)-1)* prior to the removal of redundancies. Figure 3 shows a flowchart representation of substring division and search.

3. RESULTS AND DISCUSSION

The simulation was carried out in an Intel Core i7 system running Windows 7, with 2 GHz CPUs. The programming was written in C#.NET, and was run in Visual Studio 2008. SQL Server 2005 was used as the backend. The simulation results **[14]** showed faster retrieval rates and lesser memory consumption. Removal of redundancies while performing the

indexing process helps is lesser memory consumption. This leads to lesser number of entries in the indexing table, which in turn helps in faster processing.

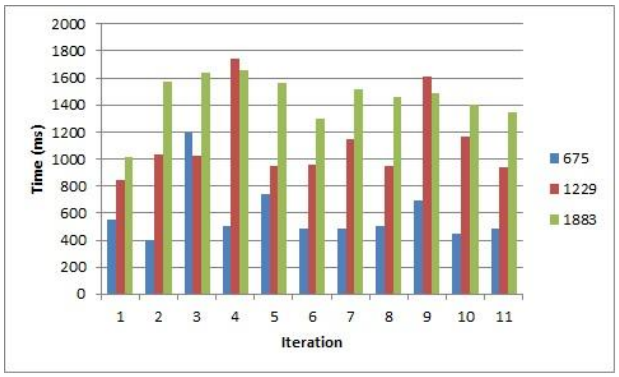

Figure 4: Time taken for indexing

Figure 4 shows the time taken for indexing when strings of various sizes are used. The legends show the number of database entries created. Here we can see that the average time taken for creating a database of size 675 rows is 589.4545ms, 1229 rows are 1123.2727ms and 1883 rows are 1451.09091ms.

During the process of searching, we can see some unusual high and low times, this is due to the usage of CPU by other services in the system. These spikes and lows are considered as noise and are leveled, and the average time taken is considered for analysis.

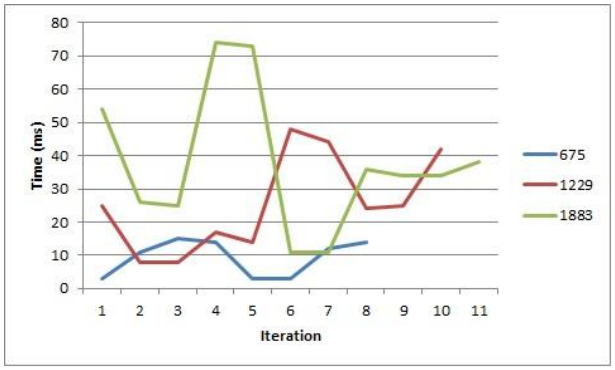

Figure 5: Time taken for search with a substring of 2 characters

Figure 5 shows the time taken for searching in databases of various sizes when considering a search string of 2 characters. The average time taken for searching a string of two characters in a database of 675 rows is 9.375ms, 1229 rows is 25.5ms and 1883 rows is 37.81818ms.

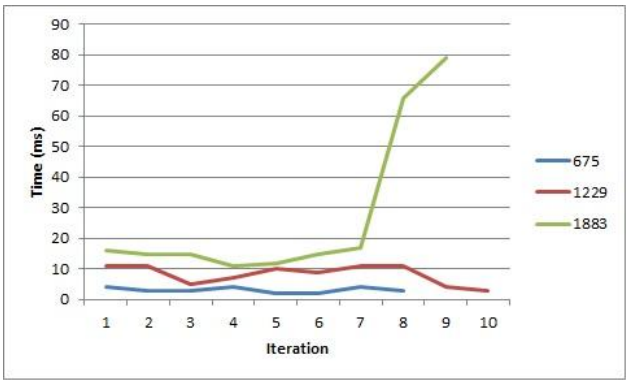

Figure 6: Time taken for search with a substring of 3 characters

Figure 6 shows the time taken for searching in databases of various sizes when considering a search string of 3 characters. The average time taken for searching a string of two characters in a database of 675 rows is 3.125ms, 1229 rows is 8.2ms and 1883 rows is 17.3333ms.

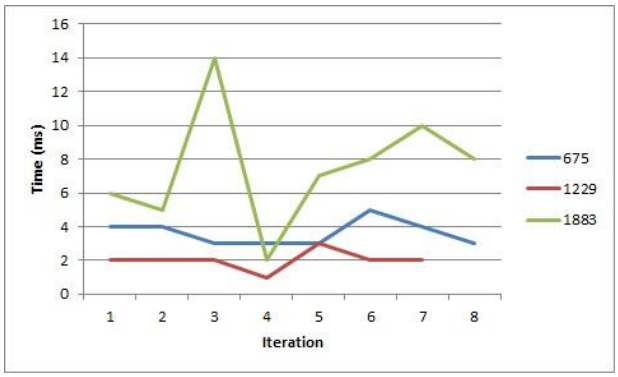

Figure 7: Time taken for search with a substring of 4 characters

Figure 7 shows the time taken for searching in databases of various sizes when considering a search string of 4 characters. The average time taken for searching a string of two characters in a database of 675 rows is 6.2ms, 1229 rows is 2ms and 1883 rows is 3.6ms.

Algorithm used in **[7]** provides an efficient way of hashing, but additional bit operations are used on the strings, hence might become time consuming when strings of long sequences are used. Further, **[8]** considers repeated values for the processing, which will lead to longer strings. In our process, since redundancies are reduced, the maximum size of strings available will be 4 (since only 4 characters are used for representing DNA sequences). Hence the size of substrings is reduced. Further, we perform the searching operation by eliminating the substrings, hence we perform searching for the first part and verifying it with the successive sequences. This process reduces the need for searching longer string sequences. This reduces the time of processing and reduced database size, which is not possible in **[7]** and **[8]**.

4. CONCLUSION

Searching for substrings in a DNA sequence is a tedious job, where the system searches in a large number of entries for the current sequence. This consumes a lot of time and CPU cycles. Reduction of this search even to a small extent will yield faster results when implemented in a large-scale environment. In our process, we can see that even though the search time increases with the size of the database, the relative increase is considerable. I.e. The order of increase is low when compared to the increase in size of the database. From the Figure 5, 6 and 7 we can see that the time taken for searching reduces as the size of the search string increases. As the size of the search string increases, the search time in all the three databases comes to an almost constant state. Eliminating the spikes and lows (caused due to the intermission of other services) as noise, we can show that the current process provides faster searching capabilities.

5. ACKNOWLEDGEMENT

We express sincere thanks to The Director, NBAII and Dr.S.K.Jalali, CCPI, (NAIP-NABG) and all members of NAIP-NABG project for their reviews on early drafts of this manuscript. Deepest gratitude to Madhusmita Panda and Sharath Pattar for their knowledgeable assistance and the greatest support. Special thanks to Malathy V.M who was abundantly helpful and offered invaluable assistance and guidance towards development of this work. The work has been initiated and supported by World bank funded National Agricultural Innovative Project on National Agricultural Bioinformatics Grid-Insect Domain is gratefully acknowledged.

6. REFERENCES

[1] Peter Snustard, Michael J. Simmons. GENETICS, Wiley 6th edition, www.wiley.com/go/global/snustad.

[2] Jones, Neil C. and Pevzner, Pavel A. (2004) "An Introduction to Bioinformatics Algorithms." Cambridge: The MIT Press. 148-226 and 311-337.

[3] Petri Kalsi, Hannu Peltola, and Jorma Tarhio, "Comparison of exact string matching algorithms for biological sequences", BIRD, 417–426, 2008.

[4] Simone Faro and Thierry Lecroq, "The exact string matching problem: a comprehensive experimental evaluation", CoRR, abs/1012.2547, 2010.

[5] Simone Faro and Thierry Lecroq "The exact online string matching problem: a review of the most recent results", ACM Computing Surveys, 45(2): to appear, 2013.

[6] Gonzalo Navarro and Mathieu Raffinot, "Flexible pattern matching in strings - practical on-line search algorithms for texts and biological sequences", Cambridge University Press, 2002.

[7] Simone Faro, Thierry Lecroq, "Fast Searching in Biological Sequences Using Multiple Hash Functions", Proceedings of the 2012 IEEE 12th International Conference on Bioinformatics & Bioengineering (BIBE), Larnaca, Cyprus, 11-13 November 2012

[8] David Nassimi, Milind Joshi, Andrew Sohn,"H-PBS: A Hash-Based Scalable Technique for Parallel Bidirectional Search", 1063-6374/95 1995 IEEE

[9] Zhou Bai Stefan C. Kremer, "Sequence Learning: Analysis and Solutions for Sparse Data in High Dimensional Spaces", 978-1-4673-1191-5/12/$31.00 ©2012 IEEE

[10] David Dittman, Taghi Khoshgoftaar, Randall Wald, and Amri Napolitano, "Similarity Analysis of Feature Ranking Techniques on Imbalanced DNA Microarray Datasets", 2012 IEEE International Conference on Bioinformatics and Biomedicine

[11] V.Hari Prasad, Dr.P.Y.Kumar, Dr. D. Vasumathi, "Adaptive segmentation of DNA sequences using SBC Tehcnique:A novel Algorithm", ICCCNT'12, July 2012, Coimbatore, India

[12] Gonzalo Navarro , Mathieu Raffinot , "Practical and flexible pattern matching over Ziv–Lempel compressed text", Journal of Discrete Algorithms 2 (2004) 347–371

[13] Nur'Aini Binti Abdul Rashid, Rana Ghadban, Hazrina Yusof Hamdani,Atheer A-Abdulrazaq, "Enhanced CAFÉ Indexing Algorithm Using Hashing Function", 978-1-4244-6716-7/10/$26.00, 2010 IEEE

[14] Maryam Nuser, Izzat Alsmadi, "Evaluating Graphical and Statistical Techniques for Measuring Similarity in DNA Sequences", 978-1-4673-1550-0/12/$31.00 ©2012 IEEE