National Institute of Technology,
Karnataka

March 05, 2023

A Practical Algorithm for DNA Pattern Searching using Database-Based Approach

Authors

Freeson Kaniwa, Mpho Phuthego

Conference

IEEE International Conference on Bioinformatics and Biomedicine (BIBM) 2018

Reference Link

https://ieeexplore.ieee.org/document/8621219

Research Paper Study Assignment 01

Submitted by:

Tejas S Gowda
t.0180.g@pm.me
201CS163
CSE S1, 4th Semester

Submitted to:
Dr Annappa B
Department of CSE

CONTENTS

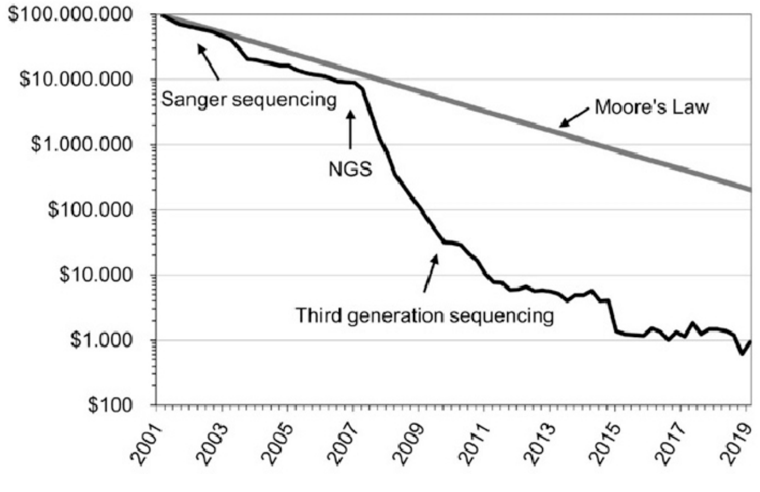
ABSTRACT

The paper discusses the growing importance of DNA sequencing data, particularly due to advancements in **next-generation sequencing technologies (NGS)** that make it cheaper and faster to sequence genomes. NGS has accelerated the sequencing of DNA and RNA. So with the increasing amount of data generated, there is a need for efficient tools and algorithms to analyze this data, particularly in the context of identifying **DNA patterns and repeats**, which have important biological applications.

The authors propose a database-based approach for searching DNA repeat sequences, using relational database management systems that support various data types. They discuss the potential advantages of using such systems for DNA analysis, such as their scalability and flexibility in handling large datasets.

PROBLEMS ADDRESSED

The paper addresses the problem of pattern searching in the rapidly growing volume of DNA sequences, which has become increasingly relevant[1] due to advancements in next-generation sequencing technologies. So newer methods and tools must be developed to catch up with this accelerated sequencing, producing Big Data[2-4]. The authors note that the cost of sequencing genomes continues to fall, resulting in a projected 2 billion human genomes (≈ 6 Exabases) by 2025, generating approximately 40 Exabytes of data[5].



[Figure 1] The departure of sequencing cost curve from Moore's law

This explosion of data poses significant challenges in terms of analysis, particularly on standard desktop computers, and has led to the development of various software tools aimed at improving the efficiency of DNA sequence analysis. The authors highlight the biological relevance of identifying DNA patterns and repeats[6-8], which can facilitate early disease diagnosis, crime investigation, and a better understanding of evolutionary processes.

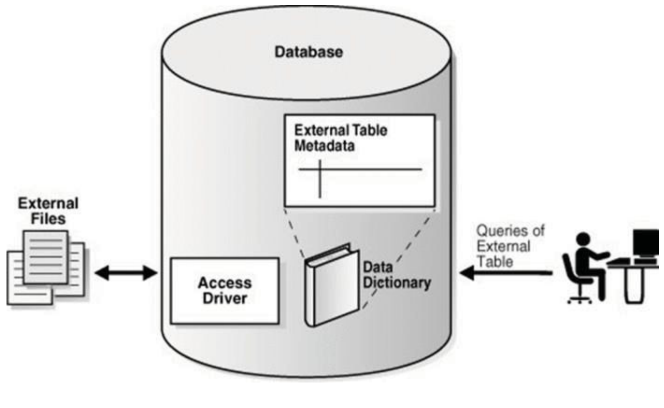
Overall, the paper aims to address these challenges by proposing a database-based approach for searching DNA repeat sequences, which leverages the power and scalability of relational database management systems. The authors hope that their algorithmic solution will contribute to the development of more efficient tools for DNA sequence analysis, ultimately advancing our understanding of genetics and its applications.

AUTHOR'S SOLUTION

TERMINOLOGY AND PRELIMINARY

Let $\Sigma = \{A, C, G, T\}$ be the set of bases in the DNA. A DNA sequence S (size = n) is constructed using the characters in Σ. A pattern P (size = k) represents the pattern to be searched in S. S_f is used to represent the saved file of the sequence S. \$ is used as a terminal symbol.

The oracle external table functions as a table for reading purposes only. It allows access to data from external sources, making it seem as though the data is within a database table[1]. While the metadata is stored in the database, the actual data remains stored outside of it. To interpret the external data for the database, an access driver in the form of an API is used.



[Figure 2] Oracle External Table[1]

RELATED WORK

This section discusses various algorithms and tools for pattern searching. **VMATCH** is a software tool that uses an enhanced suffix array data structure to search for exact matches in large sequences. **ClustDB** is an algorithm that searches for perfect matches in sets of large sequences using window alignment and performs better than **VMATCH** in terms of speed and sensitivity. **BLAST** is another algorithm that searches for high scores in sequence alignments using heuristics based on the Water-Smith algorithm. **MaST** is a recently proposed algorithm that uses the **Hadoop** platform for parallel processing to search for similar patterns on sequences, with improved performance for sequence sizes greater than 150MB. Pattern sizes vary between 5 and 15 characters long. The algorithms discussed differ slightly from the proposed algorithm but share the basic task of searching for similar patterns.

DATABASE-BASED ALGORITHM

The algorithm in short is called **DAPS** (Database-based Algorithm for Pattern Searching). As the name itself says the algorithm makes use of RDBMS to search for patterns in a DNA sequence using standard SQL queries like **LIKE**. Since SQL queries are known to be quick and efficient[9, 10].

The algorithm has two phases, these are,

01 OS Phase

To begin the algorithm, a marker is placed on groups of three characters using the sliding window technique[11]. This is shown in the OS-phase. The algorithm then proceeds with three cycles, where in the second cycle, the first character of the sequence is trimmed and a marker is placed on groups of threes again. This process continues for subsequent cycles.

02 DB Phase

All **(n - k + 1)** three-letter sub-strings are loaded into the external table by making use of the marker ($) to show the end of the string.

Example:

If S = {AAACGTTAACGTCAA} and k = 3.

OS Phase

Three files are created

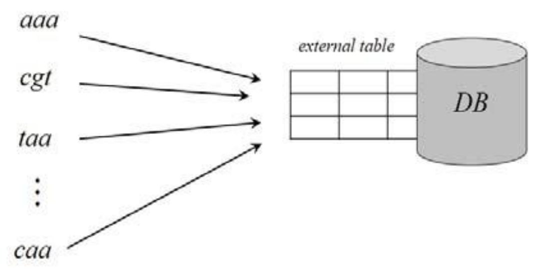
S_{f1} = {AAA\$CGT\$TAA\$CGT\$CAA}

S_{f2} = {AAC\$GTT\$AAC\$GTC\$}

S_{f3} = {ACG\$TTA\$ACG\$TCA\$}

DB Phase

The above file sequences are loaded into the oracle external table as tuples. This process is done in the DB-phase. All (n - k + 1) three-letter sub-strings are loaded into the external table by making use of the marker ($) to show the end of the string.



```
+-------+
|pattern|
+-------+
|AAA    |
|CGT    |
|TAA    |
|CGT    |
|CAA    |
|AAC    |
|GTT    |
|AAC    |
|GTC    |
|ACG    |
|TTA    |
|ACG    |
|TCA    |
+-------+
```

[Figure 3] DB Phase and the tuples in external table.

Now a simple standard query can be executed on the external table to retreive all the repeated pattern and their number of repeats

```
create table PatternCnt as
select pattern, count(pattern) as repeats
from Pattern
group by pattern
having count(*) > 1
order by pattern;
```

```
+-------+-------+
|pattern|repeats|
+-------+-------+
|AAC    |2      |
|ACG    |2      |
|CGT    |2      |
+-------+-------+
```

PSEUDO CODE

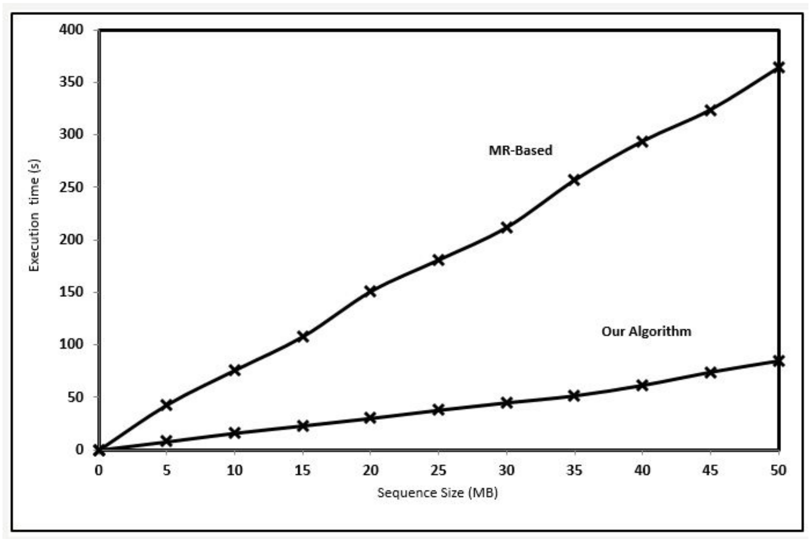
```
Initial Phase

1   k := |P|, T := { }, n := |S|, x := 0

OS Phase

2     read (S)
3     def marker(S)
4           for i ← 0; i + k to n do
5           sf ← isert ($) at S[i + k]

6   while (i < k)
7       marker (S)
8       s ← S.remove S[0]

DB Phase

9   read (Sf)
10  T ← Sf
```

They describe our algorithm with the basis of Example 1 explanation provided. The algorithm represents a given DNA sequence file into the database ready for standard SQL queries to retrieve similar patterns. They make reference to the algorithm as DAPS which represents Database-based Algorithm for Pattern Searching.

The algorithm begins by initializing all the relevant variables and then read the sequence (lines 1 - 2). A marker sub routine is defined responsible for marking the read sequence into the size of P. The marker inserts a terminal symbol $ or a terminal marker to show the end of each string and stores the file (lines 3 - 5). Line 6 ensures that the maximum number of cycles are not exceeded otherwise the marked strings will be duplicated and this is always less than size of the pattern P. The splitter is called n times as controlled by the WHILE loop (lines 6 - 7). Line 8 trims the first character out following the sliding window technique.

The DB-phase reads the stored marked file sequences and load them into the database (lines 9 - 10) from which SQL queries can be executed to retrieve similar patterns that is, exact or approximate repeats. We do not perform normalization on our imported data (sequences) as usually expected with any database records because the essence of normalization does not apply to our problem and will distort the purpose of pattern searching [12].

RESULTS



		Execution time ($k = 10$)	
Sequence Size *(MB)*	# of Records	MR-Based time *(s)*	Our Algorithm *(s)*
5	524288	43	8
10	1048576	76	16
15	1572864	108	23
20	2097152	151	30
25	2621440	181	38
30	3145728	212	45
35	3670016	257	51
40	4194304	294	61
45	4718592	324	74
50	5242880	364	84

[Figure 4] MR Based vs DAPS Algorithm, Both algorithms were executed with a specification of k = 10, that is the pattern size. Sequence size column shows the input sizes in megabytes which varies between 5 ≤ size ≤ 50 megabytes.

Dataset	Chromosome Size *(MB)*	#Records	Execution time *(s)*
Rice Chr. 12	≈ 22.14	2111498	54
Chr20	≈ 59,28	5927842	91
X-Chr.	≈ 149,69	14969033	232
Chr1	≈ 227,22	22721534	354
Chr2	≈ 237,20	23719582	382

[Figure 5] Execution times & number of records of our algorithm for chromosomes and the human genome

ISSUES IN THE PROPOSED SOLUTION

01 The algorithm shows improved performance in terms of execution time by a factor of 5. However, the algorithm fails to complete the execution of the whole human genome possibly due to low internal memory as opposed to the MR-based algorithm. Most possibly because map reduced algorithms has communication costs and therefore on small-sized inputs.

02 When the text size becomes larger than 1GB our method fails to complete the execution possibly due to the high memory requirements for larger sequences. This means our algorithm failed to execute the whole human genome as compared to the MR-based algorithm.

CONCLUSION

This paper presents a new method for searching DNA sequences for patterns using a relational database on a standard desktop computer. The database can also execute other queries to search for exact and approximate repeats, as long as the pattern size is known in advance. Our method has a construction time 5 times faster than the MaST algorithm, but it still has limited scalability due to the size of the entire genome sequence and internal memory. To improve performance in analyzing the entire human genome, other algorithmic techniques can be explored. Additional database indices and information can be incorporated to create a complete DNA database for more efficient searches.

MY SUGGESTION

External tables are read-only tables that allow access to data stored outside the database, such as in a flat file or on another database server. The metadata for the table is stored in the database, but the actual data remains outside. External tables are useful for scenarios where you need to query data from outside sources without importing it into the database, such as when dealing with large datasets that cannot fit into the database or when dealing with data that is frequently updated outside of the database.

On the other hand, in-memory tables are tables that reside entirely in memory and are used to improve performance for frequently accessed data. Data in an in-memory table is stored in RAM instead of on disk, which can lead to faster queries and reduced I/O. In-memory tables are typically used for scenarios where you need to perform real-time analytics on a large dataset or when WE have high transaction volume and need to access data quickly.

So, for more frequently used patterns a in-memory technique can be used. Another possible solution is to partition the input sequence into smaller segments and process them independently in parallel. This approach can reduce the memory requirements and improve the performance by utilizing the available resources more efficiently. Additionally, parallel processing can be implemented using distributed systems like Hadoop, which can scale horizontally to process large datasets.

MY IMPLEMENTATION

PSEUDO CODE

```
import required libraries

function main():
    driver = get_driver_instance()
    con = driver.connect("tcp://localhost:3306", "root", "1357")
    set the default schema of the connection to "sequencing"

    stmt = con.createStatement()
    stmt.execute("drop table if exists Pattern")
    stmt.execute("create table Pattern (pattern varchar(99));")
    close the statement object

    seq = read sequence.txt file
    pstmt = con.prepareStatement("insert into Pattern(pattern) VALUES(?)")
    k = 3

    for i in range(0, k):
        m = i
        while m + k <= seq.size():
            pattern = seq.substr(m, k)
            pstmt.setString(1, pattern)
            pstmt.execute()
            m += k

    stmt = con.createStatement()
    stmt.execute("drop table if exists PatternCnt")
    stmt.execute("create table PatternCnt;")
    close the statement object

    close the prepared statement object and connection object
    return 0

call the main function
```

01 Connect to the MySQL server using the get_driver_instance() and connect() methods.

02 Set the default schema of the connection to sequencing.

03 Create a statement object and execute SQL statements to drop the Pattern table if it exists and create a new Pattern table with one column pattern of type varchar(99).

04 Read the contents of the file sequence.txt into a string variable seq.

05 Create a prepared statement object with an SQL statement to insert a pattern into the Pattern table.

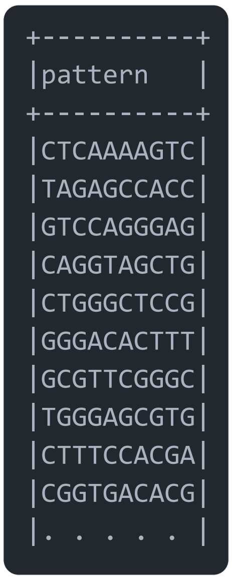
06 Loop k times and for each iteration, loop through the seq string with a step of k, and insert the substrings of length k into the Pattern table using the prepared statement object.

07 Create a new statement object and execute SQL statements to drop the PatternCnt table if it exists and create a new PatternCnt table with two columns pattern and repeats, where repeats is the number of times a pattern appears in the Pattern table.

EXAMPLE

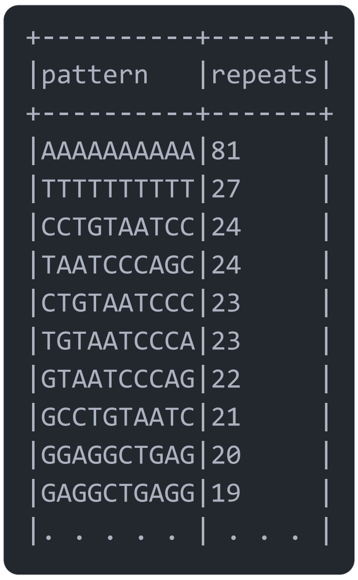
Let S = {TP53 tumor protein p53 [Homo sapiens] sequence} in Chromosome 17 from [7668421, 7687490] and let k = 10.

Loading S in the `sequence.txt` file and running it through the algorithm, we get



```
+----------+
|pattern   |
+----------+
|CTCAAAAGTC|
|TAGAGCCACC|
|GTCCAGGGAG|
|CAGGTAGCTG|
|CTGGGCTCCG|
|GGGACACTTT|
|GCGTTCGGGC|
|TGGGAGCGTG|
|CTTTCCACGA|
|CGGTGACACG|
|. . . . . |
```

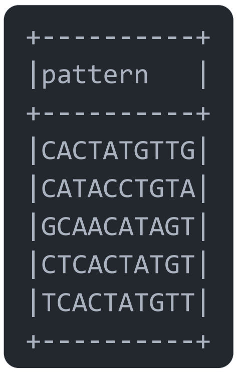
The first 10 patterns generated



```
+----------+-------+
|pattern   |repeats|
+----------+-------+
|AAAAAAAAAA|81     |
|TTTTTTTTTT|27     |
|CCTGTAATCC|24     |
|TAATCCCAGC|24     |
|CTGTAATCCC|23     |
|TGTAATCCCA|23     |
|GTAATCCCAG|22     |
|GCCTGTAATC|21     |
|GGAGGCTGAG|20     |
|GAGGCTGAGG|19     |
|. . . . . | . . . |
```

The first 10 patterns generated and their repeats in descending order

```
select pattern
from pattern
where pattern regexp ‘CA.*TA.*GT’
```



```
+----------+
|pattern   |
+----------+
|CACTATGTTG|
|CATACCTGTA|
|GCAACATAGT|
|CTCACTATGT|
|TCACTATGTT|
+----------+
```

Using a regexp to search for pattern in patterncnt

MY GITHUB IMPLEMENTATION LINK

https://github.com/dragonFly0180/dna_pattern_database_approach

REFERENCES

01 S. F. Khan and M. A. A. Raheed, "**A Developer's Guide To Database Management Systems: Using Oracle 10g RDBMS,**" 2015

02 S. Memeti and S. Pllana, "**Accelerating dna sequence analysis using intel (r) xeon phi (tm),**" in Trustcom/BigDataSE/ISPA, 2015 IEEE, 2015, pp. 222-227

03 M. Tahir, M. Sardaraz, and A. A. Ikram, "**EPMA: Efficient pattern matching algorithm for DNA sequences,**" Expert Systems with Applications, vol. 80, pp. 162-170, 2017

04 B. Soewito and N. Weng, "**Methodology for evaluating dna pattern searching algorithms on muliprocessor,**" in 7th IEEE International Conference on Bioinformatics and Bioengineering, 2007, pp. 570-577

05 F. Kaniwa, O. Dinakenyane, and V. M. Kuthadi, "**Parallel algorithm for indexing large DNA sequences using MapReduce on Hadoop,**" in 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2017, pp. 1576-1582

06 G. Benson, "**Tandem repeats finder: a program to analyze DNA sequences,**" Nucleic acids research, vol. 27, p. 573, 1999

07 K. Zhou, A. Aertsen, and C. W. Michiels, "**The role of variable DNA tandem repeats in bacterial adaptation,**" FEMS microbiology reviews, vol. 38, pp. 119-141, 2014

08 P. Meysman, C. Zhou, B. Cule, B. Goethals, and K. Laukens, "**Mining the entire Protein DataBank for frequent spatially cohesive amino acid patterns,**" BioData mining, vol. 8, p. 4, 2015

09 N. Ahmed, S. Ahamed, J. I. Rafiq, and S. Rahim, "**Data processing in Hive vs. SQL server: A comparative analysis in the query performance,**" in Engineering Technologies and Social Sciences (ICETSS), 2017 IEEE 3rd International Conference on, 2017, pp. 1-5

10 S. J. Kamatkar, A. Kamble, A. Viloria, L. HernándezFernandez, and E. G. Cali, "**Database Performance Tuning and Query Optimization,**" in International Conference on Data Mining and Big Data, 2018, pp. 3-11.

11 F. Kaniwa, V. M. Kuthadi, O. Dinakenyane, and H. Schroeder, "**Alphabet-dependent Parallel Algorithm for Suffix Tree Construction for Pattern Searching,**" arXiv preprint arXiv:1704.05660, 2017

12 E. L. Van Dijk, H. Auger, Y. Jaszczyszyn, and C. Thermes, "**Ten years of next-generation sequencing technology,**" Trends in genetics, vol. 30, pp. 418-426, 2014.