DE-BRUIJN GRAPH CONSTRUCTION USING HA-VEC APPROACH

A CASE STUDY

MEET THE TEAM

PROJECT SUMMARY

OBJECTIVE:

To use the Ha-Vec approach, we are trying to improve the genome assembly done through de bruijn graph by creating efficient memory management

PROCEDURE :

The following procedure were done by us in this case study ;

- Literature review was done on hashing and ha-vec approach on de-bruijn graphs.
- Background information on all the important terminologies was found.
- The method of the havec approach was found .
- The applications of this approach in bio informatics was found.

LITERATURE REVIEW

Research papers regarding the topic were searched and it was found that there is only one research paper available. The research paper is as follows,

- HaVec: An Efficient de Bruijn Graph Construction Algorithm for Genome Assembly

The link for the research paper is given in the conclusion

BACKGROUND INFORMATION

De Bruijn Graphs

- Directed graph
- Represents overlaps between sequences of symbols
- In bioinformatics, De Bruijn graphs are used for de novo assembly of sequencing reads into a genome.

The structure of de bruijn graph.

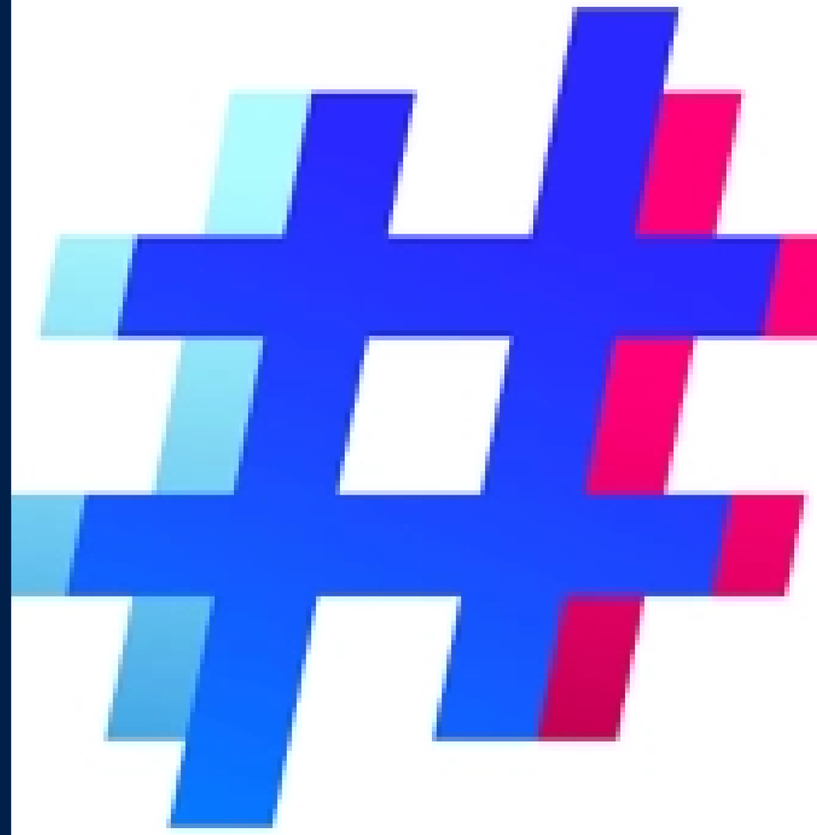
- Nodes - kmers in the reads ((k-1)mers).
- Edges - kmers in the sequence(reads), if overlap happens

Hashing

- Hashing is a technique to convert a range of key values into a range of indexes of an array.
- A hash function is any function that can be used to map data of arbitrary size to fixed-size values.
- The values returned by a hash function are called hash values, hash codes, digests, or simply hashes.
- Hashes are great for trading off accuracy, data storage size, performance, retrieval speed, and more.

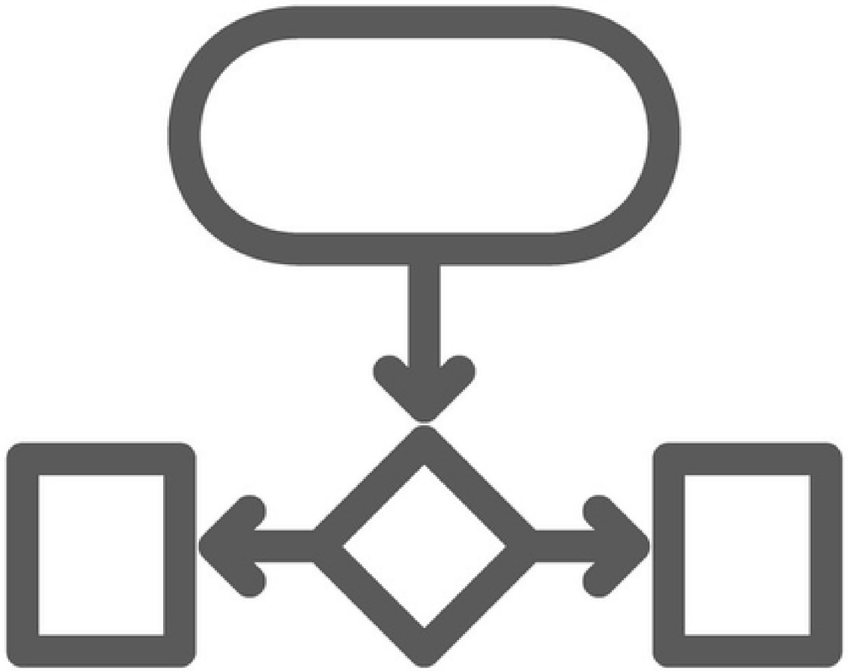
Hash Tables

- Hash Table is a data structure which stores data in an associative manner(key-value format) with direct access to its items in constant time.
- A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found.



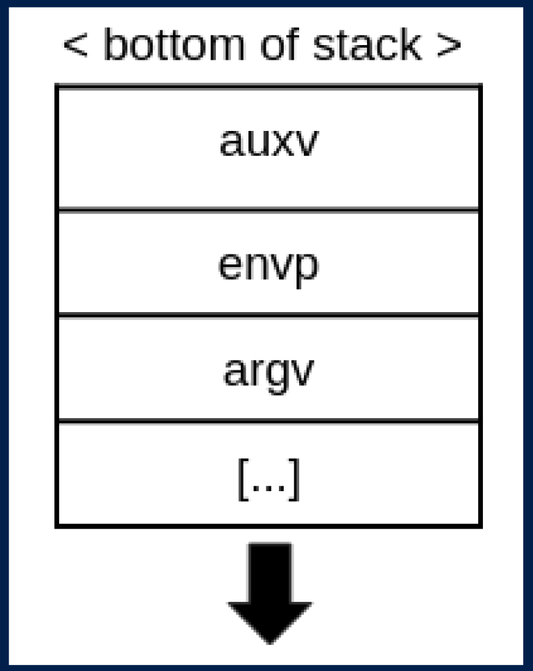
Bloom Filter

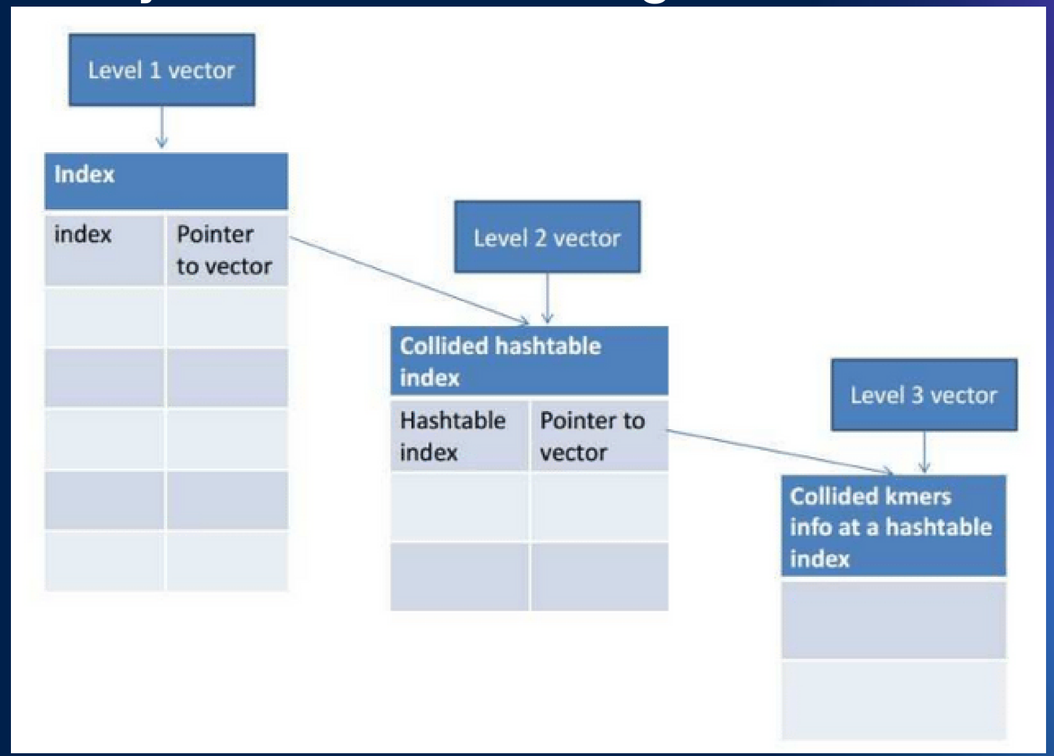
- A bloom filter is a probabilistic data structure that is based on hashing.
- The Bloom filter is a space efficient hash-based data structure that allows us to test whether an element is in a set.
- It consists of a bit array of m bits, initialized to zero. We also have h hash functions.
- To insert or test the membership of an element, h hash values are computed, which produce h array positions. To insert, we set all corresponding bits to 1.
- The membership operation returns yes if and only if each of the bits at these positions is 1.
- A negative answer means that the element is definitely not in the set.
- A positive answer indicates that the element may or may not be in the set.



Auxiliary Vectors

- The auxiliary vector is usually used to convey information from the operating system to the application.
- Only the terminating null auxiliary vector entry is required, but if any other entries are present, they shall be interpreted as follows.
- This vector is an array of the following structures.





Need For the Ha–vec Approach on De bruijn Graphs

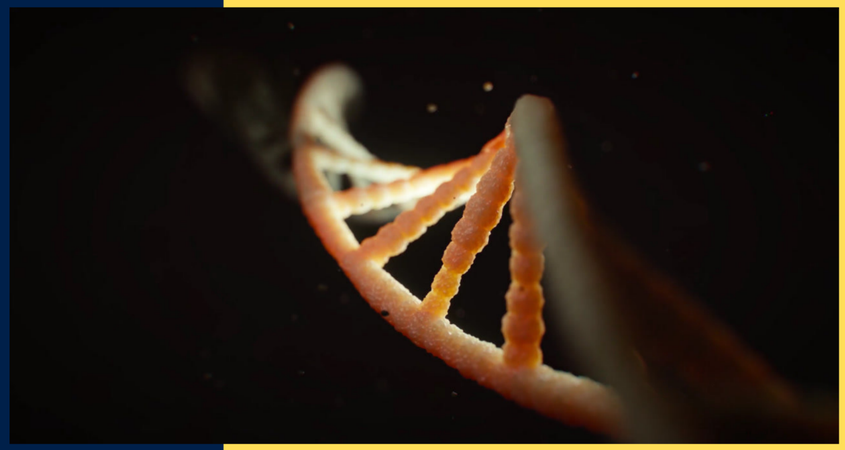
- The rapid advancement of next-generation sequencing technologies has made it possible to regularly produce numerous reads from DNA samples in sequencing laboratories.
- To this end, the de Bruijn graph is a popular data structure in the genome assembly literature for efficient representation and processing of data (a couple of hundred GB.).
- In the present state of the art, the main problem of the memory efficient Bloom filter representation of de Bruijn graph is the false positive calculation and the runtime.
- So we make an effort to alleviate these prob lems. In particular, we present a new algorithm based on hashing and auxiliary vector data structures and call this algorithm HaVec.

HAVEC'S CORE COMPONENTS

Ha-Vec approach depends upon two core components:

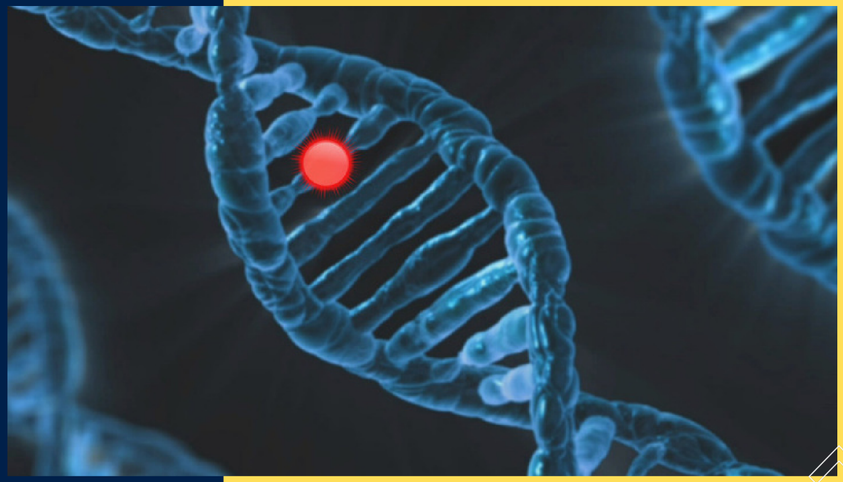
- the graph construction module and
- the graph traversal module.

The graph construction module uses a novel approach to quickly and accurately construct the de Bruijn graph while the graph traversal module is used to efficiently traverse the graph and identify the genome's contigs.



HAVEC'S ADVANTAGES

- Ha-Vec's graph construction module is faster and more accurate than existing approaches, making it ideal for large-scale genome assembly projects.
- Its graph traversal module is also faster than existing methods, allowing for faster and more reliable genome assembly.
- Overall, the Hap Vec approach can be used to improve the de Bruijn graph by reducing its size, increasing its accuracy, and making it more useful for downstream analysis tasks such as genome assembly and variant calling.



Methods & Algorithms

Method:

- First, we consider a read from some input file which may be in FASTA or FASTQ format.
- We have two hash functions hash1 and hash2
- However, the method can lead to false positives in the graph, where a false edge is created between two nodes if a neighbor node is falsely generated.
- The passage describes an approach to eliminate these false positives by using different quotient values when the same remainder is produced from dividing the hash value by the table size.
- The method involves using multiple hash functions, storing the presence information of the first node at the hash table, and using an auxiliary vector data structure for any additional nodes that cannot be stored in the table.

Methods :

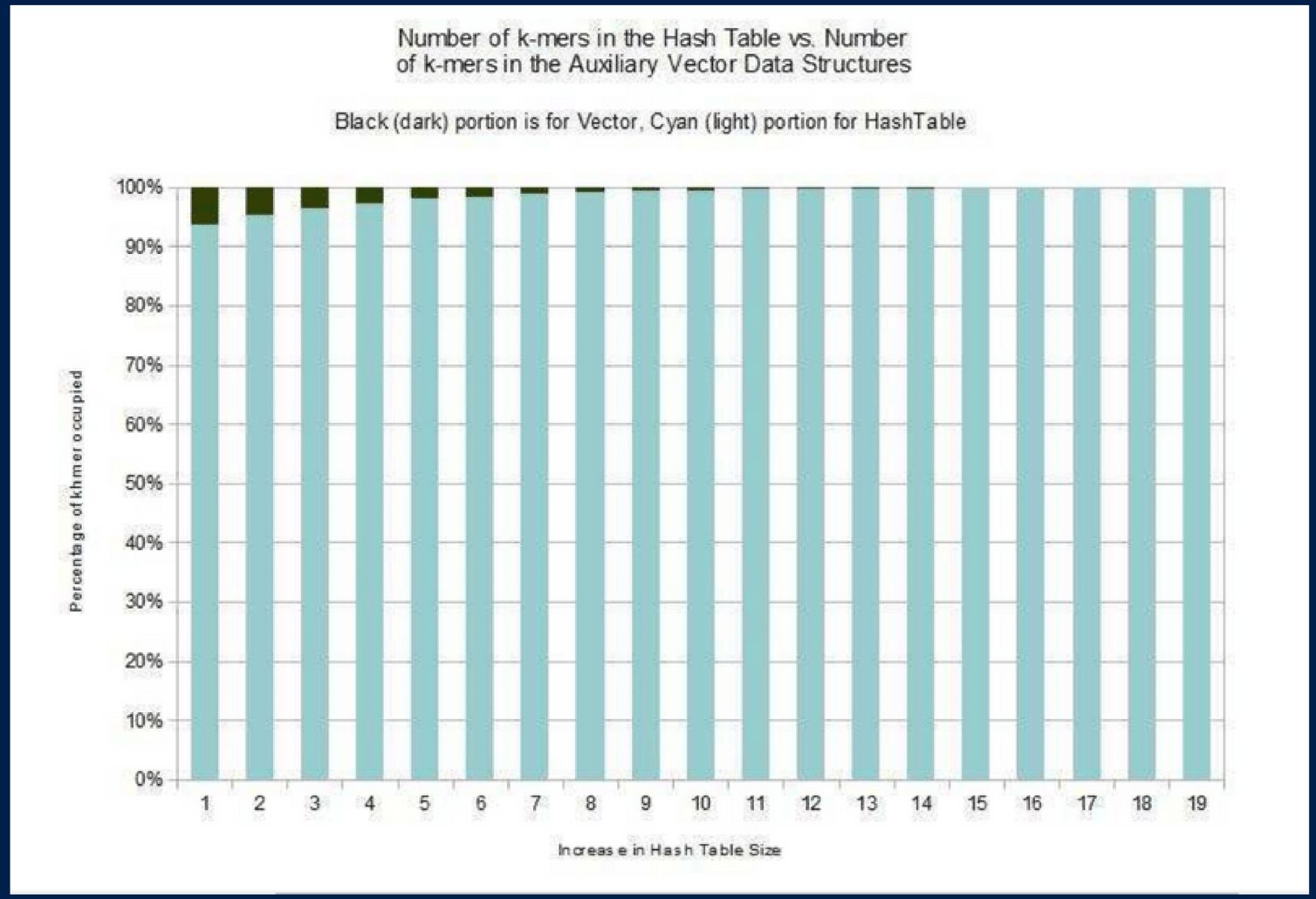
- Instead, the graph is constructed from k-mer information stored in a hash table and auxiliary vector data structures.
- The hash table is used for faster access and the size is dependent on the number of distinct nodes in an input file.
- The hash table uses 4 bits for outgoing neighbors, 3 bits to indicate which hash function was used, and the rest for the quotient value.
- The auxiliary vector data structures are used to store collided k-mers and are slower to update and access than the hash table due to three levels of indirection.
- The quotient is the result of dividing the hash value by the table size and a larger quotient size will require more memory.

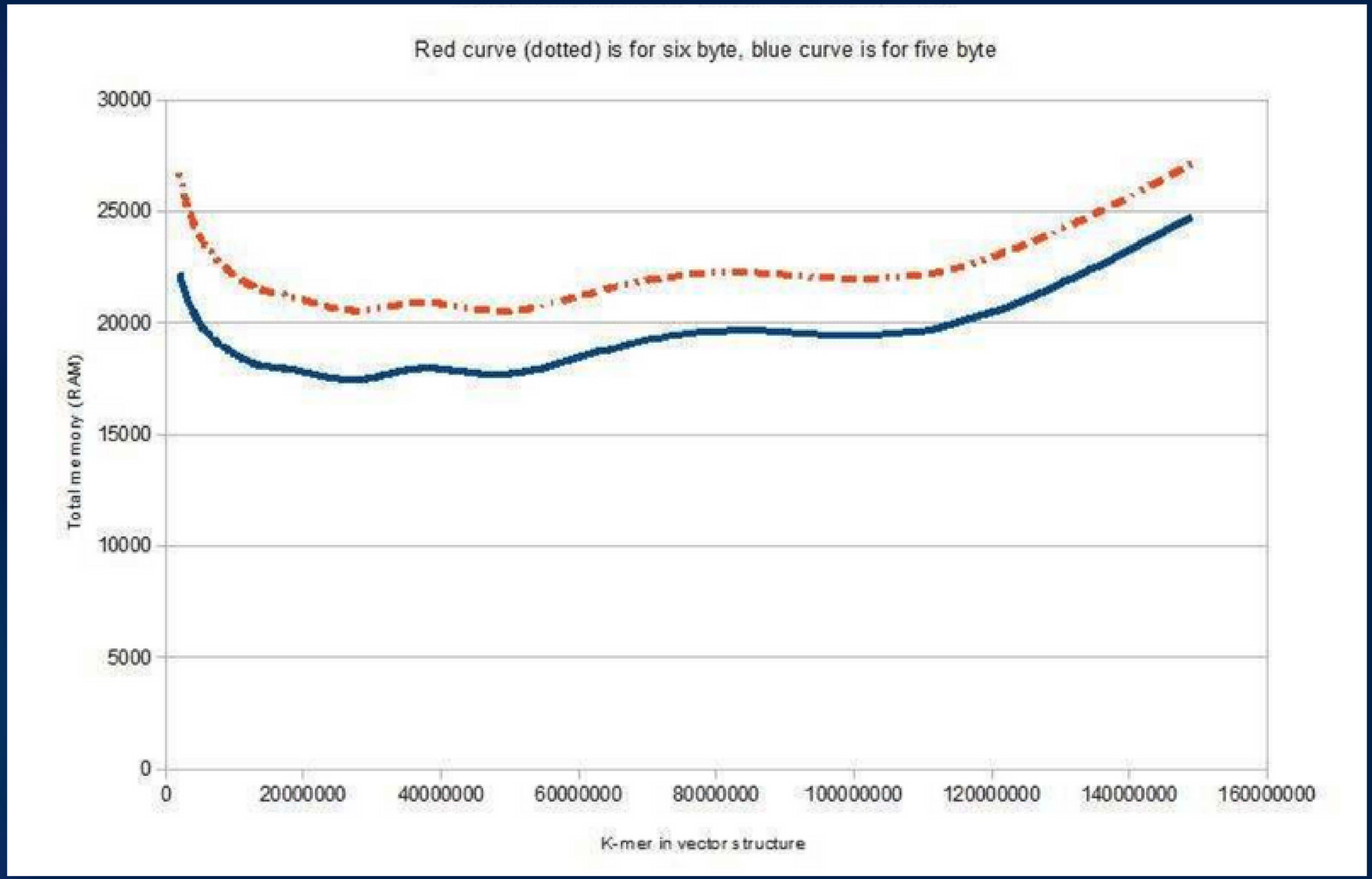
k-mers	Hash Values	Comments
GGCAA	57	
GCAAT	27	
CAATT	24, 36	
AATTG	52	
ATTGT	36, 27	Put in Vector
TTGTG	22	
TGTGT	34, 30	Put in Vector
GTGTG	49, 47	Put in Vector
TGTGT	34, 30	Found in Vector; Update
GTGTC	38, 25	Put in Vector

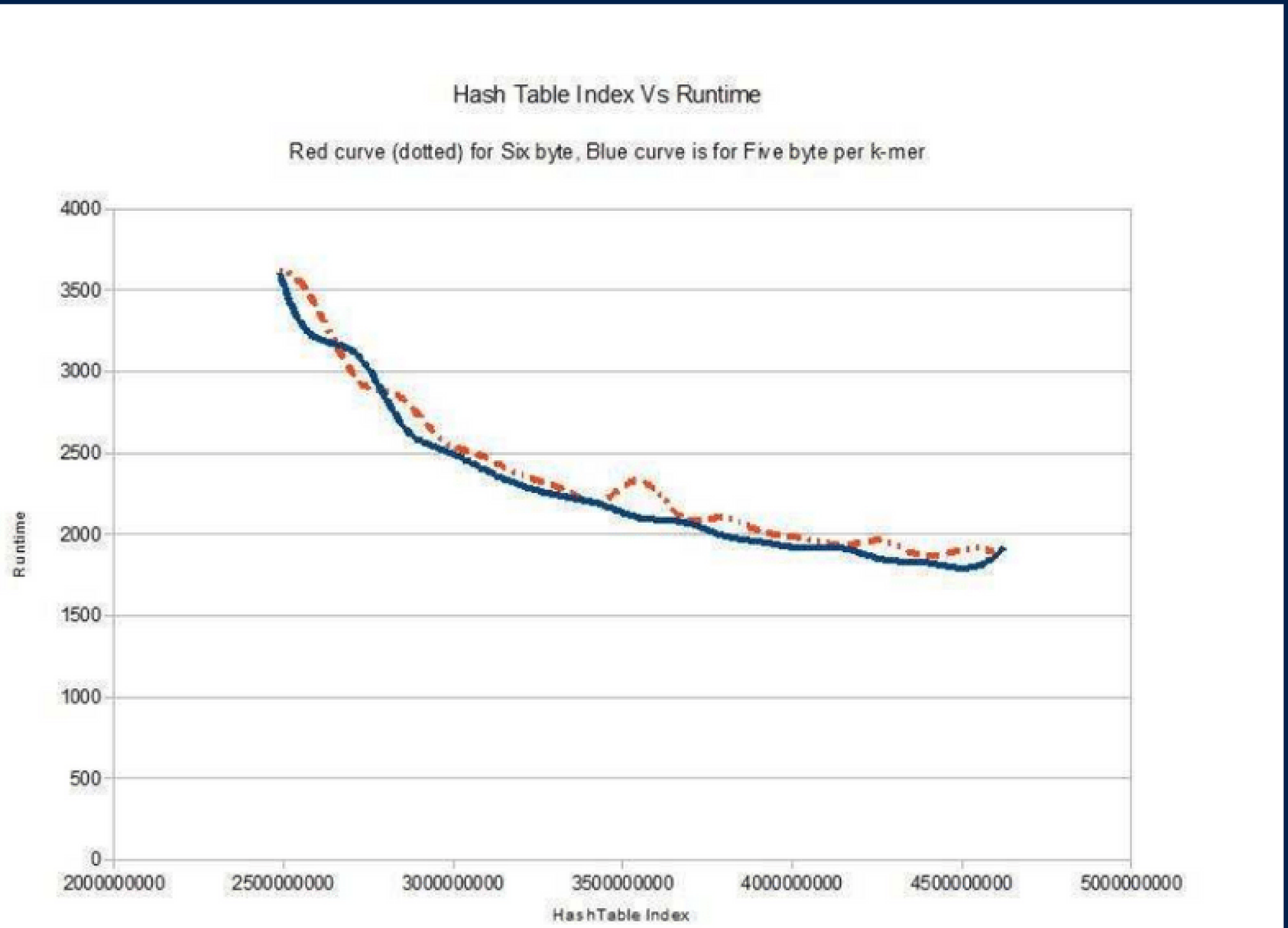
Index	Information
0	$2-1-T$
1	$0-0$
2	$5-1-T$
3	$3-2-G$
4	$0-0$
5	$2-1-T$
6	$0-0$
7	$0-0$
8	$4-1-T$
9	$0-0$
10	$0-0$

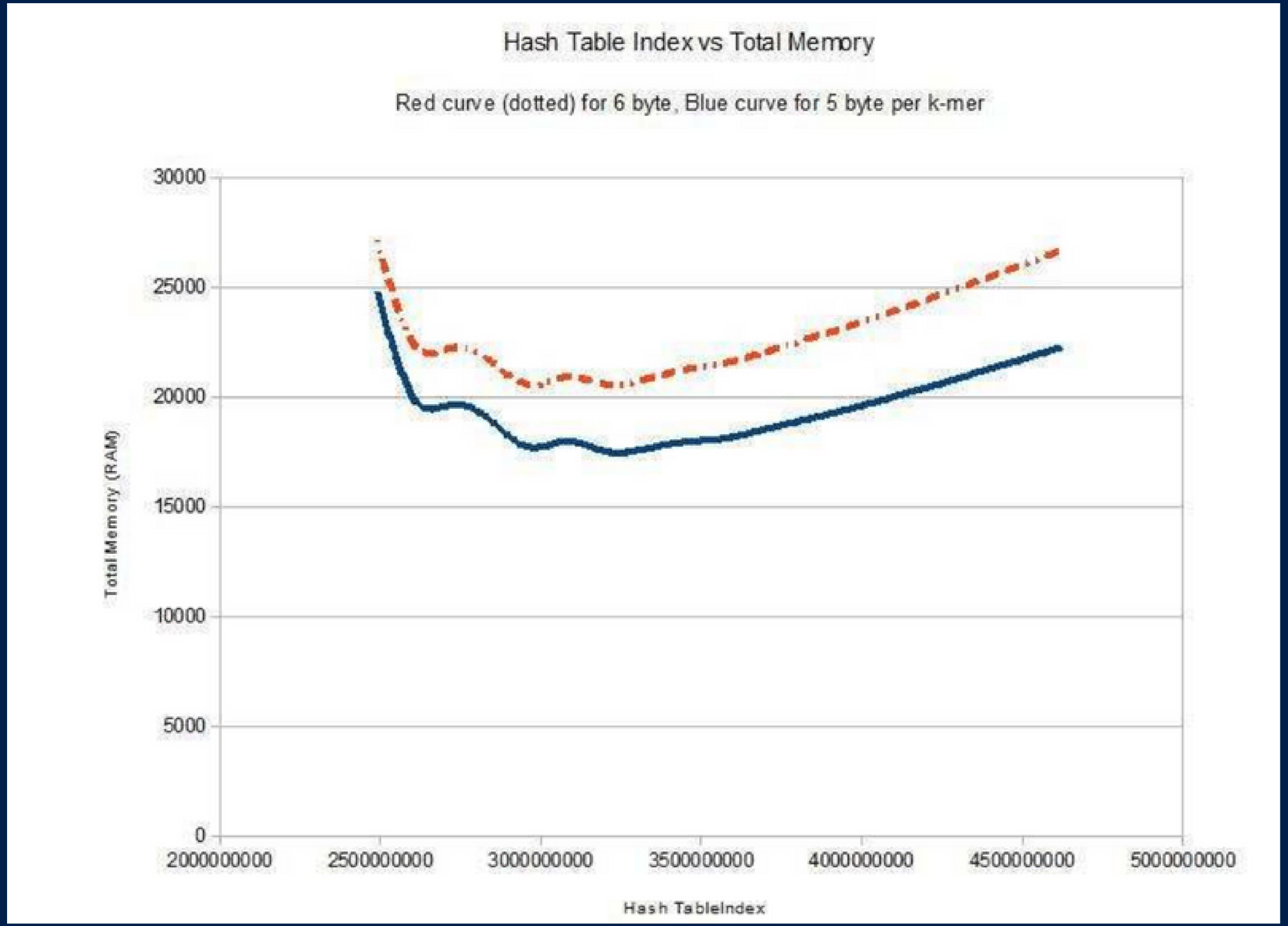
Results & Interpretation

The results are represented in various forms such a bar graphs and graphs as follows;




Red curve (dotted) is for six byte, blue curve is for five byte
30000
25000
20000
15000
10000
5000
0
Total memory (RAM)
0
20000000
40000000
60000000
80000000
100000000
120000000
140000000
160000000
K-mer in vector structure

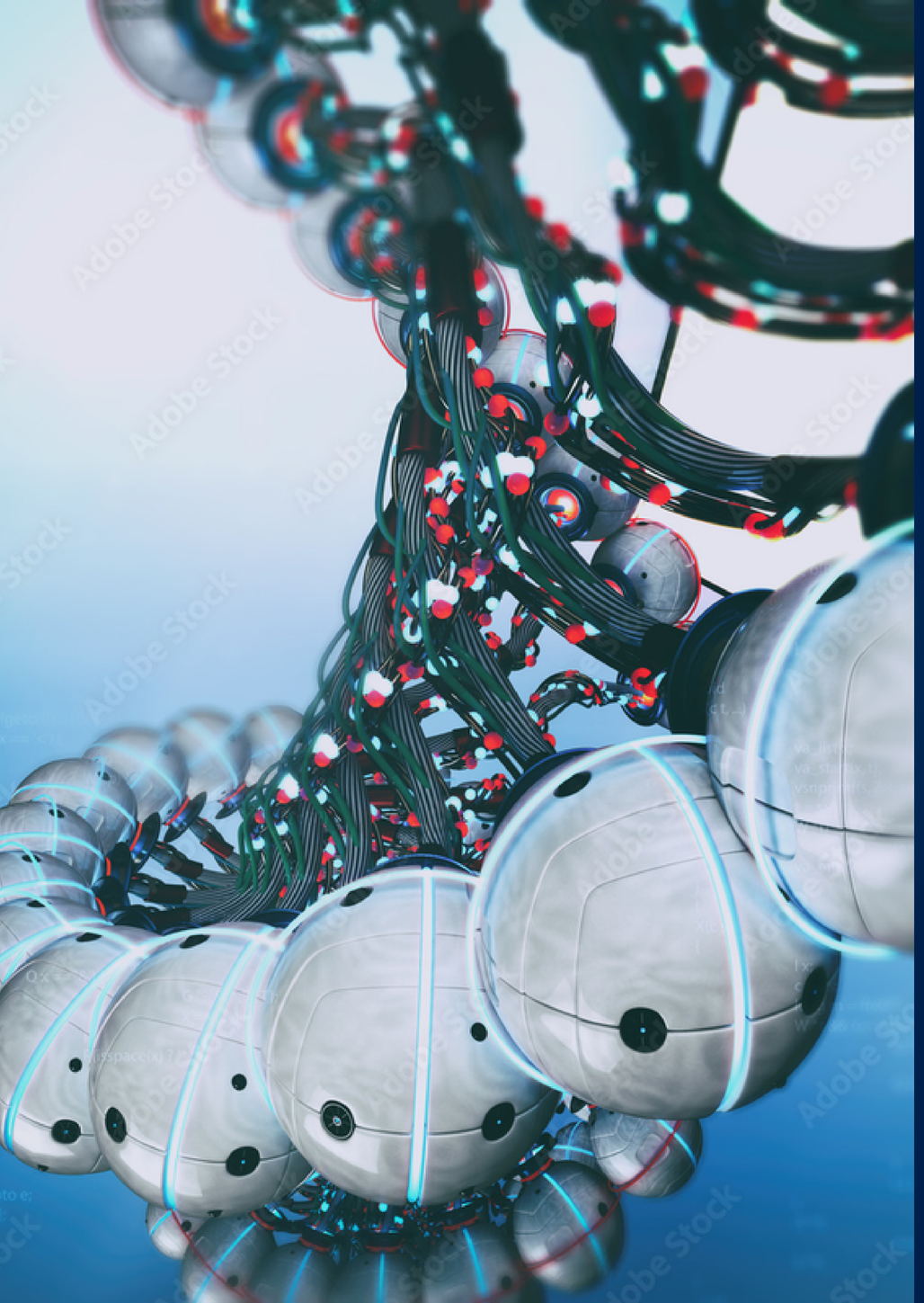

Hash Table Index Vs Runtime
Red curve (dotted) for Six byte, Blue curve is for Five byte per k-mer
4000
3500
3000
2500
2000
1500
1000
500
0
Runtime
2000000000
2500000000
3000000000
3500000000
4000000000
4500000000
5000000000
HashTable Index


Hash Table Index vs Total Memory
Red curve (dotted) for 6 byte, Blue curve for 5 byte per k-mer
30000
25000
20000
15000
10000
5000
0
Total Memory (RAM)
2000000000
2500000000
3000000000
3500000000
4000000000
4500000000
5000000000
Hash TableIndex

Interpretation of the Results

The following interpretations were arrived at from the results;

- Larger hash table size doesn’t always guarantee lower total memory consumption. Optimum memory use can be achieved when hash table size is 1.25 to 1.5 times the number of unique k-mers.
- As the hash table size increases, the number of k-mers in the hash table increases and the number of k-mers in the vector structure decreases which is certainly desirable. The same relation with the exact same values holds for both 5 bytes and 6 bytes implementations.
- The runtime decreases with the increase in the hash table size. Generally, runtime for the 6 bytes implementation is slightly higher than that of the 5 bytes implementation.



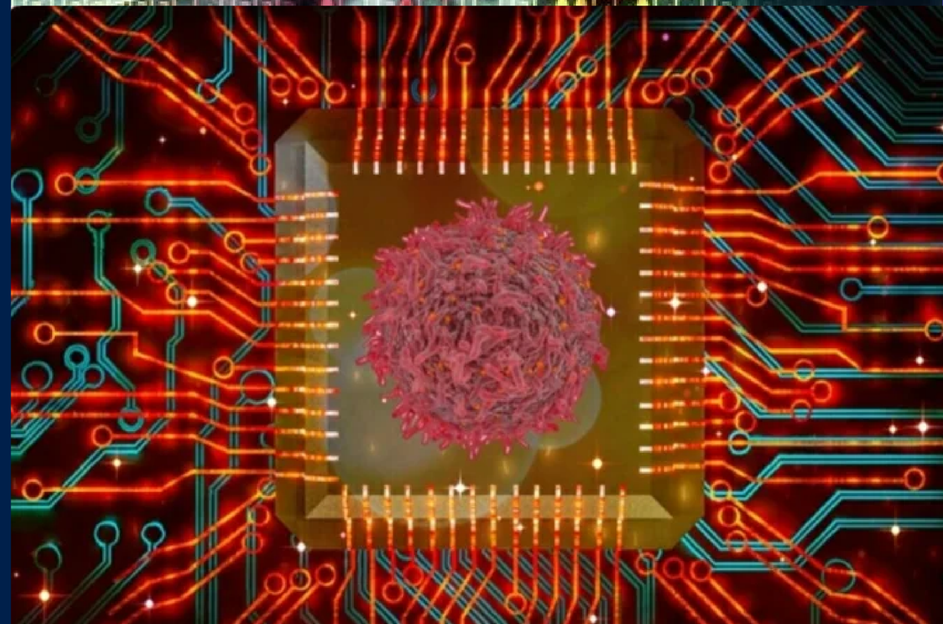
CONCLUSION

- In the whole assembly process, generally the graph construction procedure takes the major share of the time.
- This graph construction does not produce any false positives which makes it completely error free.
- This approach will be extremely useful in de Bruijn graph based genome assembly.

FUTURE SCOPE & LIMITATIONS

The approach can be further improved in the following aspects;

- Since auxiliary vector data structure is not available in python and only available in C and C++, it is difficult to add features of artificial intelligence and machine learning onto the approach.
- If more development and improvisation is done on this approach , we will be able to perform genome assembly very effficiently and quickly.



THANK YOU!