Int. j. inf. tecnol. (March 2017) 9(1):59–75
DOI 10.1007/s41870-017-0005-z

ORIGINAL RESEARCH

De-Bruijn graph with MapReduce framework towards metagenomic data classification

Md. Sarwar Kamal[1] · Sazia Parvin[2] · Amira S. Ashour[3] · Fuqian Shi[4] · Nilanjan Dey[5]

Published online: 23 February 2017

Abstract Metagenomic gene classifications are significant in bioinformatics and computational biology research. There are huge interrelated datasets that deal with human characteristics, diseases and molecular functionalities. Analysis of metagenomic reorganization is a challenging issue due to their diversity and efficient classification tools. Graph based MapReducing approach can easily handle the genomic diversity. MapReduce has two parts such as mapping and reducing. In mapping phase, a recursive naive algorithm is used for generating K-mers. De-Bruijn graph is a compact representation of k-mers and finds out an optimal path (solution) for genome assembly. Similarity metrics have been utilized for finding similarity among the De-Oxy Ribonucleic Acid (DNA) sequences. In reducing side, Jaccard similarity and purity of clustering are used as datasets classifier to classify the sequences based on their similarity. Reducing phase can easily classify the DNA sequences from large database. Extensive experimental analysis has demonstrated that graph based MapReduce analysis generate optimal solutions. Remarkable improvements in time and space have recorded and observed. The results established that proposed framework performed faster than SSMA-SFSD when classified elements are increased. It provided better accuracy for metagenomic data clustering.

Keywords MapReduce · Metagenomic · K-mers · De-Bruijn graph · Jaccard similarity

✉ Nilanjan Dey
neelanjan.dey@gmail.com

Md. Sarwar Kamal
sarwar.saubdcoxbazar@gmail.com

Sazia Parvin
s.parvin@adfa.edu.au

Amira S. Ashour
amirasashour@yahoo.com

Fuqian Shi
sfq@wmu.edu.cn

[1] Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh

[2] University of New South Wales-ADFA, Canberra, Australia

[3] Department of Electronics and Electrical Communications Engineering, Faculty of Engineering, Tanta University, Tanta, Egypt

[4] College of Information and Engineering, Wenzhou Medical University, Wenzhou, People's Republic of China

[5] Department of Information Technology, Techno India College of Technology, Kolkata, India

1 Introduction

Metagenomic is the study of organisms' population by fragmentation and sequencing of organism in our environmental ecosystem [1]. It is basically applied in microbial organism sampling from their host environment where species wise separation is difficult, complex and time consuming. Metagenomic process is important for such types of data clustering or classification which reduce data diversity. The microbial and biological data increased day by day. To handle such increasing biological data, metagenomic process play a vital role. Cloning and culturing of biological data provide new challenges for metagenomic data processing. So, metagenomic will dominant in gene expression, next generation sequencing, genome assembly, drug formulation, agriculture and genomic functionalities. Metagenomic ensures that new molecular or gene production performs in dynamic approaches. It contained several DNA sequences for experimental process. The DNA sequences contains the chain of nucleotide (A,C,G,T) for eukaryote cell or million of base pairs in human body. Thus, computational tools for gene finding are required for large biological data annotation. Position of gene is not a vital issue but it is perquisite for gene regulation, genome assembly and understanding the coding regions. Finding the genes from genome is difficult due to large data handling tools and space. Metagenomic investigate the natural data for analyze the shape, architecture or functional interaction in small species. Different microbes from multiple areas [2, 3] are used to analyze of the composite architecture community of their own environment [4]. It is important to analyze the microbe's genomic interaction or human metagenomic interaction for drug, foods or disease evaluation [5–7].

Several computational challenges are raised in metagenomic analysis. Complication informatics analysis is difficult due large amount of metagenomic data and its complexity. For example, it can be difficult to determine the genome in which reads are derived. Additionally, the diversity of species is difficult for read generation and not compares the sequence alignment [8]. Metagenomic analysis is trend to handle large volume of data to identify accurate result. However, it faces a computational tools and approaches. Sometimes, metagenomic contains unwanted host DNA of microbiota, thus filtration is required to separate the DNA portion from the host DNA for DNA sequencing. It is difficult to address the contamination problem. For example it is difficult to determine read from contamination genome. A metagenomic misleads analysis for contamination genome assembly. Metagenomic trends to expensive in complex sequence or microbial DNA analysis.

To handle these metagenomic challenges, several applications/approaches are used. Some graph based approaches handle the gene annotation, classification or prediction. Generally, graph is a mathematical tool that removes the error from genome assembly and generates an optimal solution. The De-Bruijn graph is a directed graph with set of nodes and vertex, G (V, E), where V is the set of nodes with k-mers and E is the connections between the nodes. It is considered an efficient approach that represents a sequence in terms of its k-mer components. The De-Bruijn graph is considered a principal analysis for short read sequencing assembly, where the sequences are divided into fixed length or k-mers and build a directed graph [9–11]. Figure 1 illustrated the simple example for the De-Bruijn graph.

Peng et al. [12] introduced a transcriptome assembler for the iterative De-Bruijn graph approach (T-IDBA), which is a modified De-Bruijn graph approach that rearranges the positive genome assembly from large irregular data sets.

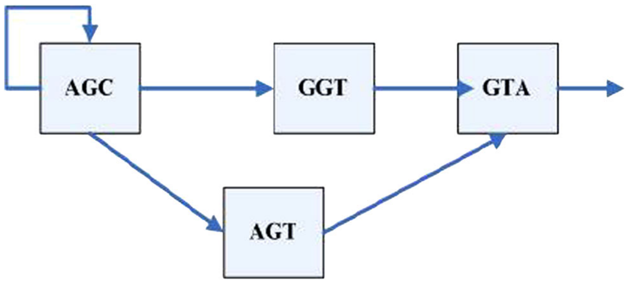

Fig. 1 A simple De-Bruijn graph

Valvet is another graph based computational tool that constructs an optimized De-Bruijn graph to discard the redundant nodes and to generate optimal genome assembly [13]. ABYSs [14] is a parallel and clustering based approach that handles a large volume of data by using De-Bruijn graph classification. This package strongly connected with k-mers node and need large storage for data sets. AllPaths [15] is another De-Bruijn based computational tools that manage the large scale data. This package handle the large scale read in an optimal way. Some graph portioning approaches [16, 17] also used to increase the quality of metagenomic data analysis with low local coverage of graph. Basically, De-Bruijn graph runs faster and assigns memory space efficiently than other comparative method. An optimized De-Bruijn graph performance is better than other gene assembly algorithm and gene annotation process.

Generally, data are increased day by day and handles by data processing. To handle the data processing, clustering; classification and sampling approaches are used to provide efficient and effective solutions. Big data handling is a challenging issue for the data mining due to system capacity and storage size. Researchers proved that traditional machine learning is failed for big data processing. So, efficient artificial and machine learning tools are required for big data analysis [18]. The ultimate objectives of big data processing is to achieve faster, scalable and optimal solutions. Map reducing [19] is a new tool for big data classification in which data are divided into small parts and combined into parallel process. Another map reducing approach is CLOSET [20], which developed based on cloud computing. Mapping and reducing both phase perform in distrusted clustering approach. The Google Map Reduced Framework [21] is a batching processing implantation strategy, where mapping and reduce phases allow fault tolerance mechanism for big data handling. MapReducing is a parallel processing system that incorporates the system design and large data classifications. It processes the data by using parallel software and hardware. It provides the programmer and user a compatible data processing environment in parallel mechanism. Thus, Map reducing is an optimal framework for big data classification.

The main contribution of the current work is to design an architecture that can handle large volume biological datasets. In this regards, the proposed approach employed the De-Bruijn graph centric MapReduce framework for Metagenomic clustering. Mapping part of MapReduce has constructed with *k*-mers generation, De-Bruijn graph and similarity matrix. On the other hand, the reduce part has organized with Jaccard similarity and purity o classification. The classification approach for metagenomic large data analysis used the De-Bruijn graph and Jaccard similarity matrix for large metagenomic data classification.

The structure of the remaining sections of the current work is as follows. "Section 2" introduced the literature review, followed by the system outline and methodology in "Sect. 3". There are few subsections that narrate in-depth of the system layout and overall methodology. "Section 4" delineated the relationships between Map and Reduce parts of MapReduce. Performance evaluations and implementation are accomplished in "Sect. 5". Finally, the conclusion is included in "Sect. 6".

2 Literature review

Recently, researches attempt to develop assemblers that support data processing in various applications. Yinan et al. [22] introduced the VirAmp, which is an embedded assembler that compared with traditional assembler by web based graphical user interface (GUI). This assembler supported data classification in parallel process. The classification process performed in single platform for large biological data processing and provided an interactive platform for the users. However, this package not efficiently handled the overlapped genomes as well as the time complexity was high with interactive genome sets. Chang et al. [23] proposed an application system called Bridgers that measured the genome arrangement by the help of de novo assembler. In this tool, Cufflinks based algorithm was used to overcome the limitations of de novo assembler. It reduced the computational time and storage than other assemblers. However, this tool did not fit in accuracy and sensitivity of Cufflinks algorithm and not efficiently handled the overlap genome. Edena is another graph based de novo assembler, which used suffix tree to handle the overlap genome sequence [24]. Edena used heuristic approach to find the overlap length and to construct a bidirectional graph. However, the graph traversing cost was too high with high space complexity. Wang et al. [25] suggested an ontological based approach called ClusDCA, which rearrange the information for all data sets that have unique activity of annotation function. In interconnected process, ontology took more time for data mapping. A mapping based algorithm can overcome the problem where reads were mapping into short read by using De-Bruijn graph [26]. Hashing function and other data structure techniques were used to handle the k-mers for graph mapping. This technique was used in metagenomic transcription to utilize the metagenome data.

Accurate gene finding and gene annotation process error free sample data are required for classifying the gene based on their functionalities. There are a large number of gene finding approaches, such as GENESCAN [27], GENEID [28] and GLIMMER [29]. On other side, gene prediction is a similarity based program, where similarity was measured between two gene sequences. Some similarity based programs are AGenDA [30] and GENEWISE [31]. Sequence similarity is important for many cases such as new discovered similarity sequences are added in current database which provides a compatible environment for next generation sequencing. Protein similarity region have a significant impact in phyla [32]. Similarity measure is a challenge for metagenomic read analysis. Recently, computation tools can address this problem. Some new programs are widely used to handle the metagenomic data processing. These programs are MetaGene [33], Orphelia [34] and GeneMarker [35].

MapReducing approach provides an efficient and convenient Metagenomic gene clustering environment to the scientist, researcher and reader. Several processes are used to design MapReducing framework. MapCluster [36] is a MapReducing clustering approach which has two phases: top-down separation and bottom up merging to cluster the Metagenomic data based on k-mers frequency and spearman distance technique. However, the relationships between these two phases are cumbersome. Yang et al. presented popular MapReducing approach called Hadoop, which provided high level development platform for Metagenomic clustering [37]. It facilitated a distributed file system and cloud environment for classification. Map reducing a computational framework with query supportable large database. MapReduce blueprint has translated by different programming language such as Pig [38], Hive [39] and SCOPE [40]. MapReduce has also applied to organize imbalanced datasets [41] Random Forest classifier in the field of large data handling scenario.

Variation in samplings and faster learning environment have addressed in the current work. However, threshold values for minority classes were absent during the mapping phase. Moreover, relationships among underlying datasets have ignored. In contrast, threshold values are assured by Jaccard similarity in our motivated research.

3 System architecture and methodology

Graph based MapReduce is an integrated framework that constitute a dynamic processing unit with several interlinked parts as illustrated in Fig. 2. Input sequences have collected from the national center for biological information (NCBI) and DNA data bank of Japan (DDBJ) references databases. Additionally, real world datasets of human have collected. Every DNA sequences input file is represented in FASTA formatas a text-based sequence alignment, which is passed into next phase.

The proposed system architecture in Fig. 2 is divided into four phases: Input, Output, Mapper and Reduction phase. In the mapper phase, DNA sequences are transformed into smallest units/fixed in length K-mers. K-mers have critical impact in bioinformatics as they determine the characteristics genome segments, where K-indicates the length and mers is a Greek word that means part. Suppose a DNA segment as is AGACCTAA, where AGAC is 4-mer, AGA is a 3-mer. Proper identification of k-mers helps faster sequences processing. Another pivotal benefit is that it exceptionally minimizes the search space during motif detections and other Metagenomic applications. After generating k-mers de brujin graph have designed for finding optimal path for genome assembly. Optimal sequences are encoded by generating similarity metrics. Similarity metrics is used to find out similarity between two sequences. Encoded value of similarity metrics is passed into next phase. In reduction phase, classification approaches for classifying sequences are utilized according to their similarity. Finally, Jaccard similarity and purity have imposed for sequence clustering.

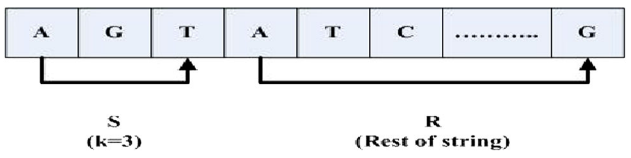

Fig. 3 Generate a substring (AGT, ATC, etc.) using recursive approach

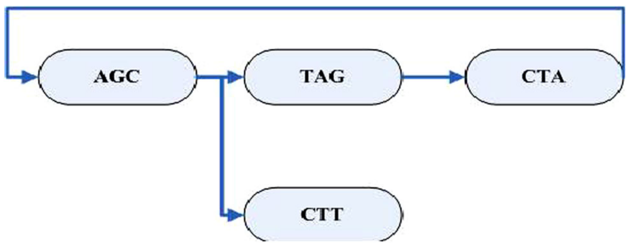

Fig. 4 A De-Bruijn graph with k = 3 mers

3.1 Generating k-mers

The sub parts are known as k-mers, where every K-mers contains key field that represent the biological information, these k-mers provides a feasible and compatible environment for genome assembly. Generating K-mers in a linear time is a challenging issue for DNA sequences analysis. A simple and optimal algorithm is proposed for generating desire substring from *n*-1 string length. The proposed algorithm is executed in constant time with linear time per string/substring generation. This strategy is simple recursive solution with certain parameters. It generates unique K-mers to reduce errors such as tips and bubbles, which are shown in overlapping K-mers generation (Algorithm 1).

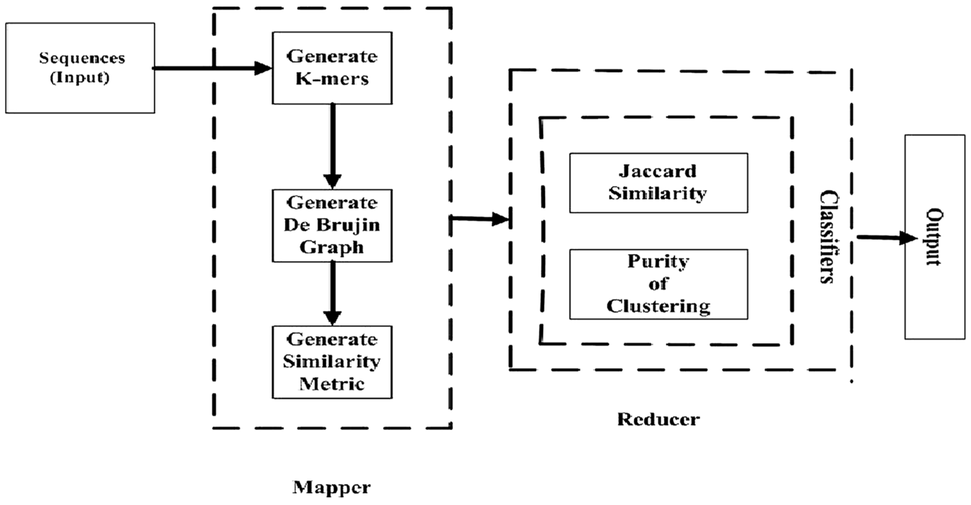

Fig. 2 System architecture metagenomic mapping

Algorithm 1: Generate k-mers(r, k)
```
{
// r is the given string contain A, G, C or T.
// k is desire substring length
If given string less than k
{
exit from process
}
For every string r
{
generate substring from j to k// j is initial position
// k is desire length
}
k-mers(r, k)// call the function recursively
}
```

Suppose a string $\beta = \beta_1\beta_2\beta_3\beta_4.........\beta_n$ with n length. If experiments generate m length of substring (k-mers), then subdivide every character from initial string to m position and next group construct recursively until last position. This algorithm always runs in constant time. The recursive function maintains three parameters, namely S is substring with desire length, R is rest of string and K is desire length. Figure 3 illustrates an example to generate a substring (AGT,ATC,…) using the recursive approach.

3.2 De-Bruijn graph

In genome analysis, De-Bruijn graph [42, 43] is built from nodes of all k-mer of a DNA sequence. A De-Bruijn graph is a directed graph with certain set of nodes and vertices. Let a directed graph G = {V, E} where vertices V are set of all k-mers and edge E is connected path between k-mer and k-1 mer. If k-mer coincide with v_1 and k-1 mer coincide with v_2, an edge is substituted between v_1 and v_2. Reverse edges are generated when two k-mer and k-1 mer are overlapped.

In de novo assembler [44], constructing de brujin graph is the first step by using all the k-mers and their neighboring nucleotides. Let all k-mers include four nucleotide A, C, G, T which encode in bit with 00, 01, 10, 11; respectively. Thus, every k-mer needs $2 \times k + 4 \times 2$ bits of memory. If a genome have N diversity, total memory space $= N \times (2 \times k + 4 \times 2)$.Sometimes, it is necessary to simplification of de brujin graph without any loss of information or data. Blocks or chains are interrupting in simplification but it is easily handle by using sub-graphs. This fragmentation reduces memory space and shortens the execution time. Whenever, node v_1 has only one incoming edge and v_2 has only one outgoing edge, these two nodes can merge. Basically, a De-Bruijn graph constructs by the following processes [45]:

1. Construction of k-mers: every overlapping k-mers are generated from DNA sequences or read. In the present work, an iterative recursive process is used to construct k-mers. Every overlap k-mers is considered as an adjacency neighboring. A unique k-mers is constructed for de brujin graph construction.
2. Node generation: in the de brujin graph each node N is represented by the unique k-mer. N node connected with N_1 node by using arc. The initial k-mer represented the start node and marginal and the last k-mers represented the ending node of de brujin graph.
3. Edge creation: a directed edge is connected from N node to N_1 node. K-mer of N node indicates as prefix and N_1 represent the suffix of the read. A repeated k-mer generates a circular arc.

Figure 4 demonstrates an example for a De-Bruijn graph with k = 3 mers. Map reducing approach [46, 47] is performed in parallel and designed for handling large data set. It handles large data sets by using parallel hardware and algorithmic approaches. It provides the users a compatible and feasible environment for large data processing. Basically, map reducing approach has two phases: mapping and reduce phase. In the map phase, input file are splits into N number of subparts with contains key filed. It handles a complicated operation because it operates every part at a time. Map reducer adds all spilt part in mapping list. All spilt portions combine in mapping phase. In genome assembly, large DNA sequence files are spilt into more sub parts (k-mers). Then all k-mers include in mapping list. For classifying or similarity measure, it combines all k-mers. De-Bruijn graph is used to connect all k-mers and makes a mapping among the k-mers. It makes a link between the k-mers and each link represent the mapping between the neighboring k-mers. Connected edges implicitly represent the presence of information between two neighboring nodes. To construct a graph in mapping phase indicates that it generates links between the k-mers for finding optimal solution.

3.2.1 *Error correction*

In graph construction process, nodes are formed from a significant number of chains, loops, blocks or directly connected edges due to overlapping or repeated k-mers. This type of formulation increases the graphs complexity, cost memory space and more time for processing. Thus, Graph simplification or error correction is important for finding an optimal or feasible path for genome assembly without any loss of data or information and reducing complexity. Graph simplification approach focuses on edge and node reduction or extraction optimal solution from original graph to reduce the operational complexity and redundant mapping among the k-mers. A simplified graph represents the optimal mapping by removing links, nodes or structures which is feasible for finding feasible path.

Error are created in the k-mers generation phase due to repeated pattern or during sequence generation or data processing. It is important to remove error after graph generation. Generally, erroneous data generates two type of error as demonstrated in Fig. 5, namely (1) tips are short and dead-end divergent in main path that occurred due to the end position of the sequence and low divergence and (2) bubbles which generated when two paths start and end from the same node due to polymorphism and biological data divergence. It generally caused random overlapped two or more tips. Bubbles are especially common of high diversity data and more frequent density of reads. Bubbles are also occurs for consequence of error in a long read or due the re-inserts of reads between the paths and makes a loop.

Some algorithmic approaches [48, 49] are used to remove tips and bubbles. The tip chain can be discarded in a straightforward manner without disrupts the connectivity. This discards information has only local effects but not affects the genuine sequence. Tips are corrected in two categories: length and minority count. If the tip length is less than the *k*-mers length, thus it can be removed. It is difficult to reassemble the short read. In the case of long tips, the arbitrary cut of tips which are greater than k-mer that can be generate two mistakes. A tip length is longer than k that indicates original read or biological data error. To resolve such problem, minority count is applied. The "minority count" is the properties where tips are connected with a junction and it have another path to reach the junction without traverse the tips. A tip can be removed, if the path from start node to junction node without traverse the tips, is reachable. Basically valvet [46] approach removed the tips in such two criteria. The graph will be simplified when no tips are present.

Tour bus algorithm [46] is for removing bubbles from De-Bruijn graph. Bubbles contain similar sequence and paths are redundant because they start and end at common nodes. Tour bus is used effective approach to find out feasible path or solution. Dijjakajta and breath first search [50] is used to find out redundant path in tour bus algorithm. This algorithm starts from arbitrary node and traverse along the directed graph. Distance is calculated from origin to end node for every possible path. Optimal path is considered which has minimum cost. Redundant paths are merged which belong maximum cost. Tour bus algorithm merged the node progressively and removes the bubbles from constructed graph (Fig. 6). Tips and bubbles correction are occurred recursively until the generation of simplified graph.

After reducing tips and bubbles, a simplified De-Bruijn graph is generated without loss of any information. A simplified De-Bruijn graph represents the optimal mapping between the spilt portions (k-mers) of large data sets. An optimal mapping provides a compatible environment for users or programmer. In mapping phase optimal De-Bruijn graph provides a high scalable and low fault tolerance environment. From that environment the optimal paths or

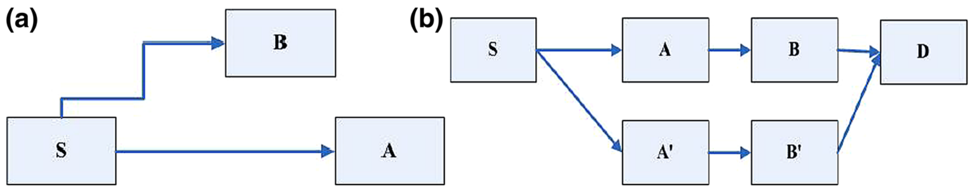

Fig. 5 Error in construction De-Bruijn graph. **a** Tips, **b** bubbles

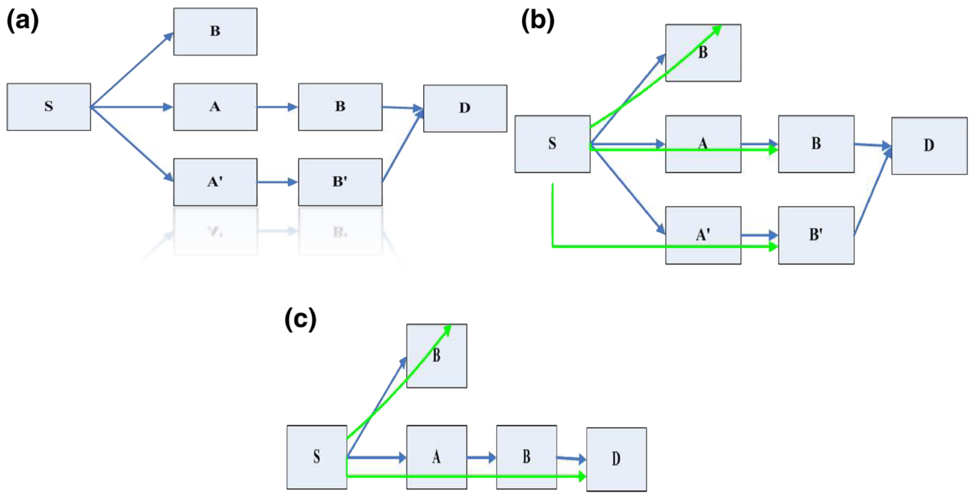

Fig. 6 Bus Tour algorithm for bubbles correction **a** original graph, **b** node are traverse randomly (*green line*) and calculate distance. *A, B* and *A′*, *B′* nodes start and end at same node. Distances are calculated progressively for both multipath and **c** *A′*, *B′* nodes are merged with *A*, *B* because the path cost of *AB* is minimum

DNA sequences can be determined, which similarities are measured by generating similarity metric.

3.3 Similarity matrix

The DNA similarity is concerned with understanding the DNA information and biological analysis. It is mainly performed by the comparison of different heterogeneous or homogenous organism. Sequence comparisons or similarity measurements play vital rule for biological functional or evaluation inference. Sequence alignment or similarity measure compared by searching common character pattern among the related sequences [51].

String matching is a challenging issue in pattern reorganization or similarity measure. Approximate string matching is a factor in matching algorithm, small difference indicates that compared strings are almost similar and zero return indicates exact similar strings.

In order to handle DNA sequences, it is essential to differentiate between global and local alignment. Local alignment is a common approach for DNA stands, which is more effective for dissimilarity measure between certain regions of motif sequences. Alignment algorithms search maximum matching score for source and target sequences. Maximum matching character or nucleotides indicate high score and mismatch or penalty for alignment is less alignment score. The standard way of DNA sequence by a string with consecutive chains of characters (A, C, G, T). Typically, string matching problem character of first position is checking of one strings (reference sequence) with the first location of other strings (query sequence).This matching approach proceed until the last character of query sequence as illustrated in Fig. 7. This method is time consuming if large data set are used for DNA alignment. Dynamic algorithm is used in efficient DNA sequence alignment [45, 46].

A similarity matrix has designed for DNA sequence length with equal or unequal lengths. These sequences are generated from De-Bruijn optimal path traverse. Similarity between optimal solutions (sequences) in mapper phase. Classification of sequences are performed from similarity matrix, then the classifier reduces the sequence class in reduce phase.

In the current work, the similarity matrix has three phases: initialization, matrix fill and trackback. Suppose a sequence S_1 with m length and another sequence S_2 with

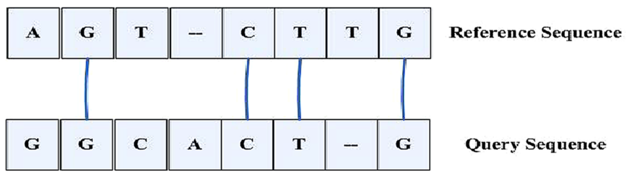

Fig. 7 Local sequence alignment

n length. The algorithm operates on $(n + 1) \times (m + 1)$ matrix M. For each of the matrix element (i, j)th are calculated by matching and mismatching score (Algorithm 2).

```
Algorithm 2: Similarity_matrix(S1,S2,m,n)
{
//$S_1$ is a DNA sequence with $m$ length
//$S_2$ is another sequence with $n$ length
Begin Check
{
For every  $i$=0 to $m$ and every $j$=0 to $n$
{
If $M[i]$ and $M[j]$ is equal then
$M[i, j]$=1
Else
$M[i, j]$=0
}
}
```

3.3.1 Initialization

In this phase, matrix M is initialized with two DNA sequences in top row and left most column. These sequences are generated from De-Bruijn graph and placed two sequences among the optimal sequences for similarity measure. Two sequences AGCTC and AGTT for similarity matrix have considered. These two sequences are unequal length, so the gap sign (−) can be used in the matrix initialization, which presented in Table 1.

3.3.2 Matrix fill

Matrix M is starting to fill up from left top corner to bottom right cornet. Every cell of the matrix is filled up considering two possible options: if the element or nucleotide i and j is similar or not similar. If two nucleotide in (i, j)th position is similar, thus assign '1', otherwise assign '0'. When, every cell is filled up by 1 or 0, then the whole matrix M is consider as filled up matrix. The formulation of matrix fill up is as follows:

$$S(i,j) = \begin{cases} 1; & i == j \\ 0; & i \neq j \end{cases}, \tag{1}$$

where, $S(x, y)$ is the score function of similarity or dissimilarity score for DNA sequence. Every cell is filled up using Eq. (1) and after competition phase bottom right corner holds the last position of the scoring value.

Table 1 Initialization and matrix fill

	A	G	T	T	–
A	1	0	0	0	0
G	0	1	0	0	0
T	0	0	1	1	0
G	0	1	0	0	0
C	0	0	0	0	0

3.3.3 Trackback

Trackback approach is used to determine the number of similar nucleotide. It starts from bottom right cell and visit every diagonal cell up to left top cell. In traversing process, the total number of '1' is counted. Total number of 1s indicated similar nucleotide between two sequences as shown in Table 2. Then, the similarity ratio can be measured, which used to classify the sequences into two classes, namely in same group or not.

Construction a similarity matrix it is convenient to find out similarity between two sequences (Algorithm 2). A sequence is AGTGC and another one is AGTT, Table 2 is used to construct a similarity matrix between the two sequences. After trackback, the total number of 1s is computed, which found to be 3 (Table 2) and the longest DNA sequence length is 5. Thus, finally the similarity ratio is computed as follows:

$$\text{Similarity ratio} = \frac{3}{5} \times 100\% = 60\%. \tag{2}$$

Afterwards, the similarity group can be classified based on the threshold value of the similarity ratio. These similarities reduce the ungroup sequence in reducing phase and reduce cost of memory in big data handling.

3.4 Jaccard classifier

Jaccard index or Jaccard similarity coefficient is a statistical similarity measurement approach between two data sets to indentify the degree of similarity between them. A significant number of supervised and unsupervised classification algorithms, such as UCLST [52], CD-HIT [53], which used for Metagenomic samples classification. Improved Metagenomic assembly allows spices diversity and reduces computational complexity inferred from successful sequence classification or grouping. In MapReducing approach, Jaccard classifier is used for Metagenomic classification in the reducing phase. Jaccard similarity uses minimum hashing [54] for quickly and accurate computation for whole Metagenomic classification. Clustering result provides different taxonomic class and using threshold value provides a same classification class. Jaccard classifier provides a quality of classification class and reduces groups in reducing phase.

Given a sequence represented as a set of k-mers, the similarity between two sequences can be defined by Jaccard classifier. If a sequence S_1 with k-mers sets denoted by

Table 2 Similarity matrix and Trackback process (green shade)

	A	G	T	T	–
A	1	0	0	0	0
G	0	1	0	0	0
T	0	0	1	1	0
G	0	1	0	0	0
C	0	0	0	0	0

J_{S1} and second sequence S_2 with k-mers sets denoted by J_{S2}, then Jaccard similarity can be defined as:

$$S(S_1, S_2) = \frac{|J_{s1} \cap J_{s2}|}{|J_{s1} \cup |J_{s2}}, \tag{3}$$

here $S(S_1, S_2)$ is the similarity measurement scores. The expression by independent permutation S_n is defined. An independent family, F (classifier group) is defined with any set of independent family element X is randomly chosen. Then, the probability P is given by:

$$P(\min\{\ \varphi(x\}\ ) = \varphi(x) = \frac{1}{X}, \tag{4}$$

where $\varphi(x)$ is the independent probability of X that equal to the minimum element of X under the permutation of φ. Thus, any the probability of an element becomes the minimum number of permutation values [48]. Given a set of input l with k-mers (maximum length = 4^k) and a min-wise independent permutation φ, then the similarity measure for two data sets J_{S1} and J_{S2} is given by:

$$P(\min\{\ \varphi(J_{s1}\}\ ) = P(\min\{\ \varphi(J_{s2}\}\ ) = \frac{|J_{s1} \cap J_{s2}|}{|J_{s1} \cup |J_{s2}} \\ = S(S_{1,}S_2). \tag{5}$$

The minimum permutation probabilities of two sets are equal to the similar of the Jaccard similarity. The similarity function provides the degree of their similarity. Based on their similarity, the sequences are classified into groups. Calculated similarity value of two sequences is equal or greater than prescribed threshold value θ, then this two sequences are in same group (Algorithm 3).

Algorithm 3:Jaccard_classifier (A,S)
```
{
// A is initial array
//S is whole sequences
RepeationBlock
{
Choose $S_i$ from S // where $S_i$ is a DNA sequence
Assign A[$S_i$] as a new cluster label
Remove $S_i$ from S
          Loop start $S_j \rightarrow$ S do
{
if ( $\frac{|S_i \cap S_j|}{|S_i \cup |S_j}$ )$\geq \theta$ then
{
Add A[$S_i$] to A[$S_j$]
                    Remove $S_j$ from S
}
} Loop End
Until S is empty and all sequenced are completed
 // Repeating block end
return A
}
```

When the threshold is assigned 0.95 then reducer phase allows those clusters which similarity is 95%. That's mean 95% similarity sequences are allowed by reducer which is in same class.

3.5 Purity of clustering

Clustering purity is an external evaluation approach for clustering quality and the performance of class generation. Clustering evaluation demands a reliable and independent measure for determining the elements of clusters, which are coherently connected similar or dissimilar groups. In common groups, every element shares the same feature and categories. The most popular cluster evaluation is clustering purity. Purity [55] focuses on the frequency of cluster categories in each same cluster. Let C is the set of cluster, N is the set of cluster categories and M is the number of cluster items in each cluster. The Purity expression by measuring the maximum precession value is given by:

$$\text{Purity} = \sum_i \frac{|C_i|}{M} \max\left(\text{Precission}\ \left(C_{i,}, N_j\right)\right). \tag{6}$$

The precision of cluster group C_i with N_j categories can be defined as:

$$\text{Precission}\ \left(C_i, N_j\right) = \frac{|C_i \cap N_j|}{|C_i|}. \tag{7}$$

The purity determines the percentage of each cluster items consist in each cluster. It takes its value [0; 1]. In the current study, all clusters items are considered to exit in same cluster group, which indicates 100% accurate clustering group. In map reducing approach, reducer measures the cluster purity. Based on purity of clustering, reducer ranked cluster group in cluster list. In cluster list, highest purity cluster belongs in top position and so on. Purity of clustering provides the reducer to represent the clustering group in optimal way and efficient manner.

4 Correlation between approaches in Map and Reduce phase

Mapping operation is performed through three phases: K-mer, De-Bruijn graph and similarity matrix. Input sequences are spilt into N numbers of unique k-mers. K-mers generation is a substring process by using recursive algorithm. Subparts of sequences (K-mers) are the initial stage of mapping phase [56–58]. K-mers are small unit of input sequence. A collection of K-mers are constructing De-Bruijn graph. In De-Bruijn graph every nodes are consist the k-mers and every edges are mapping direction between the k-mers. De-Bruijn graph represents the total mapping scenarios among the k-mers. Optimal solutions (sequences) are generated from De-Bruijn graph by random traversing. The closeness of generated sequences is measured by using similarity matrix. Closeness is measured by similarity ratio. Reducing phase allows those sequences which covers desire similarity ratio. Satisfied sequences whose covers the similarity ratio are passed in reduced phase. In reducing phase, Jaccard similarity is used to classify the sequences in certain class. Maximum similarity between the sequences belongs the same class as illustrated in Fig. 8. Purity of classification arranges the classification groups in an order based on their purity value. Purity of classifier indicates that specific class belong maximum

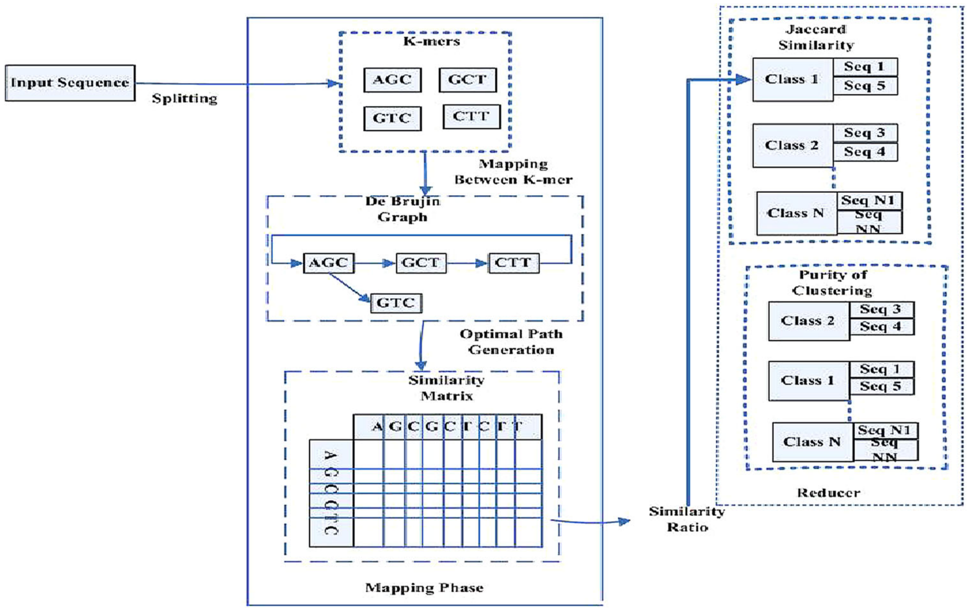

Fig. 8 Correlation among the components of mapping and reducing phase

accurate sequences in same class. According to their purity value, the class is ordered and is represented in a hierarchical structure.

5 Result and implementation

In the current study, object oriented programming (OOP) using Java was applied for the development of this analysis. Java is platform independent framework that helps to design a small memory space system. To evaluate the performance of the proposed framework (MapReduce-based technique), three different types of datasets from three different species, namely the Escherichia coli, *Gluconobacter oxydans* and *Acinetobacter baumanniias* given in Table 3 were collected and evaluated [59]. Every datasets have unique sequence id with spices name and different cluster group. The sequences have different length of Reads. Reads are split into different length of k-mers.

The simulation results measured several metrics to evaluate the Metagenomic data analysis by computing the accuracy rate, speed up and execution time using the following expressions:

- Accuracy: measures how accurately classifier classified the elements within a group. It counts the total number of classified groups to total number of elements. It expresses in percentage (%) as follows:

$$\text{Accuracy} = \frac{T_G}{T_N} \times 100\%, \tag{8}$$

where T_G is the total number of groups and T_N is total number of elements.

- Speedup: is a ratio between execution time of traditional approach and MapReduce-based approach (proposed), which given by:

$$\text{Speed}_{\text{up}} = \frac{T_{td}}{T_{mr}}, \tag{9}$$

where T_{td} is a run time for traditional approach and T_{mr} is the run time.

- Timeliness or execution time: measures the total time spent by the Map reducing phase including every operational steps in the proposed framework.

These evaluation metrics are used to evaluate the proposed framework; additionally comparisons to other approaches are conducted.

5.1 Performance of *k*-mers

The generations of k-mers evaluation are measured by execution time and speedup. For comparing the error rate, it is required to construct De-Bruijn graph. In De-Bruijn graph construction, the sequence error is added and enhanced the graph complexity for finding optimal paths. For that reasons, the k-mers performance evaluation is important for accurate graph generation. Different lengths of k-mers are used to measure the execution time and speedup for every length of k-mers. Moreover, comparative speed up is compared with MapReduce-based approach and the previous research work MSuPDA in [60].

In k-mer generation linear recursive substring generation is used. The MSuPDA is used, divided and conquered approach for sequence segmentation. The K-mers performance evaluation for different k-mer lengths for the

Table 3 Three types of metagenomic data records

Species name	Cluster number	Reads
Escherichia coli	3	49,996
Gluconobacter oxydans	4	99,998
Acinetobacter baumannii	2	4000

Table 4 K-mers performance evaluation for different k-mer length

Species name	Number of read	K-mers length	MapReduce-based approach			MSuPDA		
			Execution time (s)	Speed up (%)	Accuracy (%)	Execution time (s)	Speed up (%)	Accuracy (%)
Escherichia coli	49,996	3	102.34	72.4	78.5	130.21	65.3	69.56
		5	98.45	75.6	77.5	128.25	64.3	68.70
		9	84.5	79.5	79.1	125.34	62.5	63.21
Gluconobacter oxydans	99,998	3	145.35	67.25	74.6	176.45	59.4	58.6
		5	139.45	69.10	72.10	167.45	58.3	57.5
		9	135.72	72.32	69.78	165.34	54.9	56.9
Acinetobacter baumannii	4000	3	35.5	80.3	77.5	63.6	72.5	64.6
		5	32.6	79.8	80.3	58.4	72.5	66.5
		9	28.5	77.5	81.5	55.7	70.5	64.3

different dataset types is illustrated in Table 4. In addition, the proposed approach is compared to the MSuPDa algorithm based on execution time, speedup and accuracy (Table 4).

Table 4 illustrated that the recursive substring generation algorithm required linear time, while the MSuPDA spent more time for every k-mers generation. Moreover, the MSuPDA is less accurate than the proposed approach in k-mers generation. When the spilt portions not satisfy the k-mers length, the MSuPDA discarded the length, however the recursive algorithm overcome the problem. Table 4 established that the proposed framework generated accurate unique k-mers and generated an error free De-Bruijn graph. Figure 9 demonstrated a comparative analysis of the proposed algorithm and to the MSuPDA approach.

Figure 9 depicted that the proposed approach reduced the execution time with respect to the increase of k-mers lengths. When, the k-mers length is increased, the proposed approach required less time to split the whole genome reads. The MSuPDA required similar execution time for every length of k-mers as demonstrated in Fig. 9a. However, the MapReduce-based technique achieved higher speed up than MSuPDA for every k-mers length as illustrated in Fig. 9b. When the lengths of k-mers increased, the speed up value of the proposed algorithm is slightly increased. Similar accuracy is noticed for every k-mers generation from sequence reads. Increases the sequence length have no remarkable effects on the proposed approach accuracy. The MSuPDA accuracy is decreased with increased data sets. In MapReduce-based analysis, about 80% accuracy rate are computed where MSuPDA achieve maximum 70% accuracy as demonstrated in Fig. 9c.

A comparative analysis of the proposed approach to the MSuPDA in terms of the execution time and the accuracy rate for the three data sets is given in Fig. 10.

Figure 10a established that the *Gluconobacter oxydans* spices required more time than other two spices because near about 1 million reads belongs in this data set. On contrary, the *Acinetobacter baumannii* data set required less timing for k-mers generation because it has near about 4000 reads. The proposed approach required linear time for k-mers generation for every data set. Figure 10b illustrated the accuracy comparison between the proposed approach and MSuPDA for three data sets. More accuracy is noticed for k-mers five and nine of newly proposed approach for *Escherichia coli* and *Acinetobacter baumannii*. The accuracy rate of these two spices is near about more than 75%, while the MSuPDA measure the accuracy for the same spices is not greater than 70%. In both algorithms measure less accuracy for every

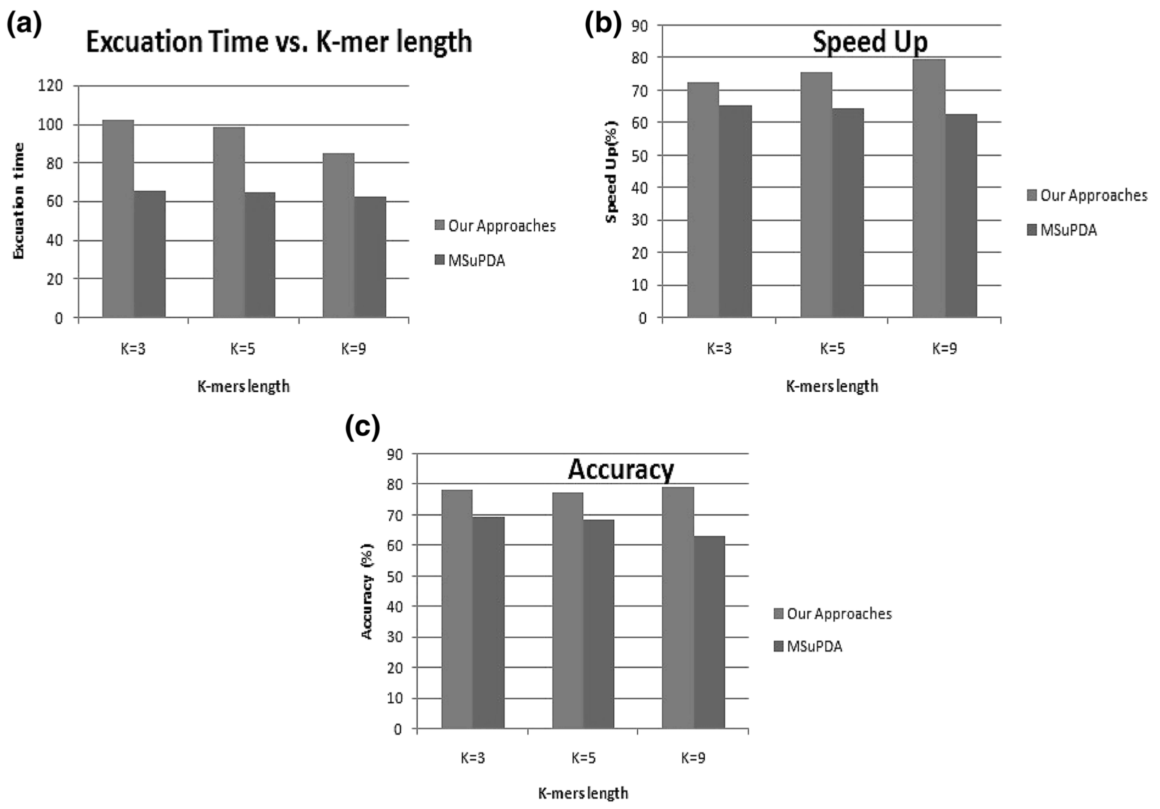

Fig. 9 Performance evaluation for different k-mers of *Escherichia coli*: **a** execution time measurement, **b** speed up, **c** accuracy

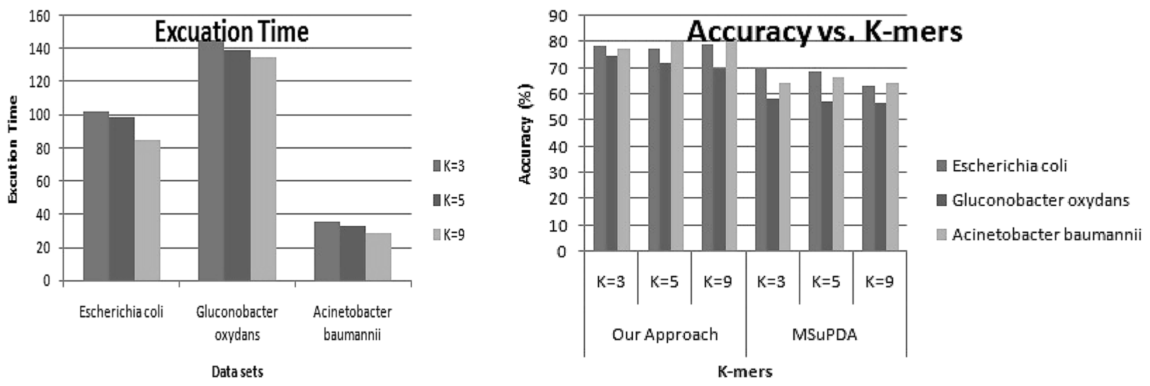

Fig. 10 Execution time and accuracy rate for three data sets. **a** Execution time of k-mers (3, 5 and 9) of the three spices, **b** accuracy comparison between our approach and MSuPDA

Table 5 Performance evaluation of De-Bruijn graph

Sequence length (bp in thousands)	K-mers	Construction time (s)	Accuracy (%)	Error rate (%)
3500	5	202.50	85.34	10.5
	10	175.57	83.21	11.25
	15	174.45	80.50	12.75
4200	5	220.50	81.50	14.25
	10	212.23	79.45	15.50
	15	204.45	78.50	17.50
4600	5	230.15	77.50	9.50
	10	224.50	75.50	11.50
	15	219.45	74.50	15.50

Table 6 Performance evaluation of Edena

Sequence length (bp in thousands)	K-mers	Construction time (s)	Accuracy (%)	Error rate (%)
3500	5	242.45	75.34	20.5
	10	255.50	73.21	18.45
	15	248.45	72.50	19.85
4200	5	270.45	72.45	24.25
	10	275.25	73.55	19.50
	15	280.50	72.45	18.75
4600	5	285.50	75.65	22.50
	10	290.55	72.45	19.45
	15	305.75	70.55	18.75

k-mers of *Gluconobacter oxydans* spices. When the data size is increased execution time and speedup increase and accuracy becomes decreased.

5.2 Performance of De-Bruijn graph

In noise free simulation, De-Bruijn graph is an efficient approach for genome assembly. It establishes a link among the read by generating k-mers. The proposed technique has constructed an error free graph for finding an optimal path. Construction time, error rate and accuracy for finding optimal paths are the performance parameters for graph construction. Optimized De-Bruijn graph or error free De-Bruijn graph ensure that the performance rate will be high. De-Bruijn-based MapReduce (proposed) method is compared to another graph based approach Edena [24]. Edena used suffix tree array for overlapping reads. Edena is a bidirectional graph construction and can traverse twice between two nodes. It generates high complexity for traversing to find optimal path. Several k-mers generation performance with construction time and accuracy has been tested under De-Bruijn graph-based MapReduce as given

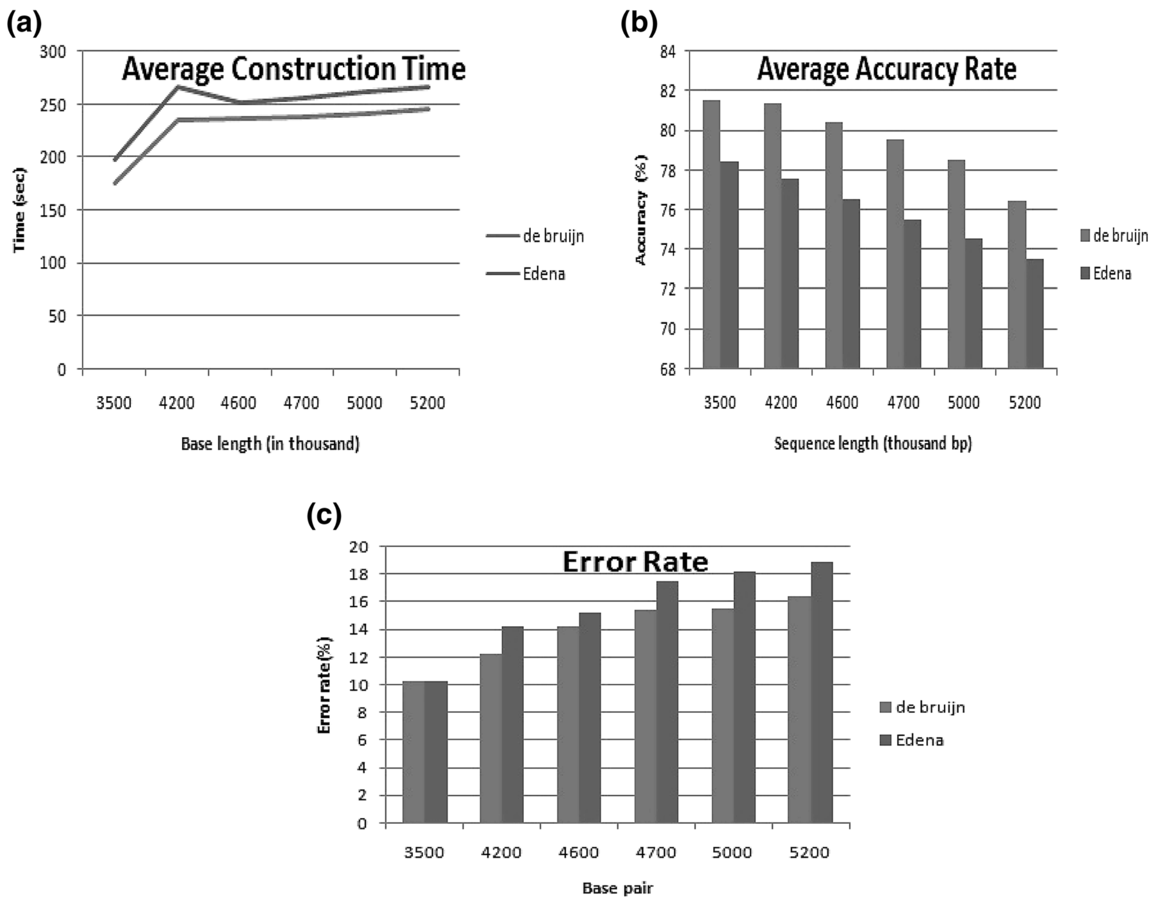

Fig. 11 Comparative analysis between De-Bruijn graph and Edena. **a** average excitation time for graph generation, **b** accuracy, **c** error rate

in Table 5, while the performance evaluation of Edena is given in Table 6.

Table 6 demonstrated that Edena required more time for establishing link between the nodes because it used bidirectional approach. Accuracy rate of the Edena is comparatively lower than the accuracy of the proposed De-Bruijn graph based MapReduce approach. Moreover, the error rates are higher than the obtained using the proposed approach in Table 6. Thus, the De-Bruijn graph is faster than Edena during graph construction.

Figure 11 illustrated a comparative analysis between the De-Bruijn graph and Edena with respect to the average excitation time for graph generation, accuracy and the error rate.

Figure 11a depicted that both algorithms suffer excessive execution time when sequence length is increased. The accuracy of De-Bruijn graph is slightly decreased, when the sequence length is increased for every k-mers. Figure 11b illustrated that the De-Bruijn graph average accuracy was not less than 75%, while the Edena average accuracy was near to 72%. In addition, the error rate of both algorithms was computed. The error rate measures the numbers of redundant nodes are used in graph construction. A simplified graph reduced error and computational complexity. Figure 11c demonstrated that the De-Bruijn and Edena had similar error rate for base length 3000(thousands) and error rate still similar for every sequence length. However, the Edena error rate is slightly increased due to overlapping k-mers and bidirectional edges creation.

5.3 Accuracy of mapping and reduction capabilities

In this section, the accuracy of mapping and reduction capabilities of the proposed framework was narrated, where the accuracy rate is important when mapping and reduction approaches are used with large date sets. High mapping and reduction rate indicates the classification time for new instances. System complexity is reduced when the reduction rate is increased. Another MapReducing technique, namely the SSMA-SFLSD [61] is considered with De-Bruijn graph based MapReduce. This algorithm operates two steps: selection and generation. In selection step, the memetic algorithm [61] is used to select the promising number of instances from the datasets. In generation phase, different evaluation algorithms in [62, 63] were used to adjust the selected instances.

Table 7 Accuracy evaluation of the proposed approaches for different data length

Sequence length (bp in thousands)	K-mers	Mapping time (s)		Classification time (s)	
		Avg	Std	Avg	Std
3500	5	835.34	123.45	1124.26	235.45
	10	774.56	105.45	1150.77	215.67
	15	723.23	98.25	1560.35	202.35
4200	5	945.67	137.25	1235.67	255.65
	10	867.56	124.58	1103.45	225.34
	15	853.45	118.35	1045.75	205.33
4600	5	1022.55	145.50	1450.25	265.32
	10	987.45	138.20	1225.55	254.57
	15	923.25	125.25	1105.75	248.56

Table 8 Accuracy evaluation of SSMA-SFLSD for different data length

Sequence length (bp in thousands)	K-mers	Mapping time (s)		Classification time (s)	
		Avg	Std	Avg	Std
3500	5	1845.34	403.45	1579.93	557.01
	10	1779.26	395.85	1606.44	537.23
	15	1693.23	388.65	2016.02	523.91
4200	5	1905.47	537.05	1691.34	577.21
	10	1827.56	524.38	1559.12	546.9
	15	1843.65	518.32	1501.42	526.89
4600	5	2022.35	545.50	1905.92	586.88
	10	1997.05	538.20	1681.22	576.13
	15		527.25	1561.42	570.12

Tables 7 and 8 included the running time calculations for the mapping and reduction phase with different sequence length for the proposed framework and the SSMA-SFLSD, respectively. During the reduction phase, classification time is considered as a reduction time; also the mean (average) and the standard deviation are computed for mapping time and classification time. Mapping time measure from total time spends in mapping phase.

Tables 7 and 8 established that when the length of k-mers increased, the classification time is decreased. In the proposed framework, the mapping phase required more time than the reduction process because in mapping runs more sub-process than reduction phase. Figure 12 demonstrated a comparison between the proposed approach and the SSMA-SFSD with respect to the average running time.

Figure 12a established that in the mapping phase, the proposed frame work required less time than the SSMA-SFSD. The mapping time is lightly increased when sequence length is increased. In small sequence length, the k-mers length has no significant impact in mapping phase,

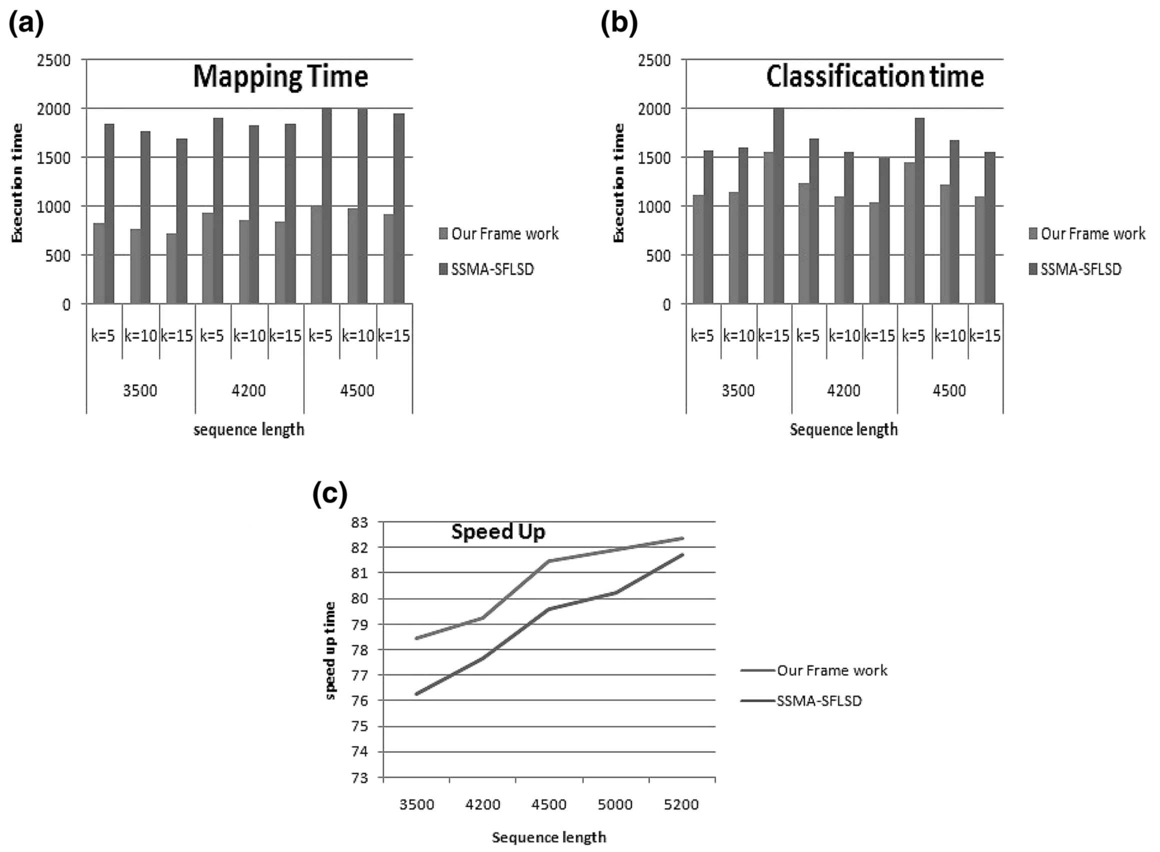

Fig. 12 Average running time for the proposed framework and SSMA-SFSD. **a** Comparison based on mapping time, **b** classification time in reduction phase, **c** speedup measurement for reduction operation

thus the SSMA-SFSD achieved the same performance as the proposed framework, when the data size is small. In the reduction phase, the proposed framework performed better, however it required more time than the mapping phase. Similarly, the SSMA-SFSD required more time in the reduction phase as demonstrated in Fig. 12b. The speedup for both algorithms is computed from the reduction steps average time when several elements are classified in parallel. When the classification instances increased, the speed up value is increased. The results depicted that both algorithms have similar speedup, however the proposed frame work performed faster than SSMA-SFSD when classified elements are increased.

Consequently, the preceding results established the superiority of the proposed approach over the other algorithms. Nonetheless, it is essential to synthesis the convinced research topic more comprehensively to address the following issues: (1) identify overlapping sequences and (2) use noisy datasets. Thus, in future demonstration, dynamic and extensive reasoning are mandatory to address the challenges mentioned above in order to cover the larger datasets and scopes. This will result better performance and applicability of the system.

6 Conclusion

In the current work, De-Bruijn-based-MapReducing approach and classification technique provided significant results for big data analysis. It reduced the error rate, system fault and increase speedup and accuracy in both mapping and reducing phase. A large number of metagenomic data are manipulated in map reducing approach. It executed in parallel and reduced the execution time and space. With the increased number of sequence of species, the proposed system efficiently handled the datasets and accurately classified. The results depicted that the proposed approach is capable to handle and classify large metagenomic data sets.

The experimental results ensured that the proposed map reducing approach with De-Bruijn graph and Jaccard similarity can easily handle the large scale data with low space, well speedup and less execution time.

References

1. Wooley JC, Godzik A, Friedberg I (2010) A primer on metagenomics. PLoS Comput Biol 6(2):e1000667

2. Ley RE, Hamady M, Lozupone C, Turnbaugh PJ, Ramey RR (2008) Evolution of mammals and their gut microbes. Science 320(80):1647–1651
3. Rusch DB, Halpern AL, Sutton G, Heidelberg KB, Williamson S (2007) The Sorcerer II Global Ocean Sampling expedition: northwest Atlantic through eastern tropical Pacific. PLoS Biol 5:e77
4. Huttenhower C, Gevers D, Knight R, Abubucker S, Badger JH (2012) Structure, function and diversity of the healthy human microbiome. Nature 486:207–214
5. Besemer J, Borodovsky M (1999) Heuristic approach to deriving models for gene finding. Nucleic Acids Res 27(19):3911–3920
6. Greenblum S, Turnbaugh PJ, Borenstein E (2009) Metagenomic systems biology of the human gut microbiome reveals topological shifts associated with obesity and inflammatory bowel disease. PNAS 109:594–599
7. Qin J, Li Y, Cai Z, Li S, Zhu J (2012) A metagenome-wide association study of gut microbiota in type 2 diabetes. Nature 490:55–60
8. Handelsman J (2007) Committee on metagenomics: challenges and functional applications. The National Academies Press, Washington
9. Pevzner P, Tang H, Waterman M (2001) An Eulerian path approach to DNA fragment assembly. Proc Natl Acad Sci USA 98:9748–9753
10. Miller J, Koren S, Sutton G (2010) Assembly algorithms for next-generation sequencing data. Genomics 95:315–327
11. Compeau P, Pevzner P, Tesler G (2011) How to apply de Bruijn graphs to genome assembly. Nat Biotechnol 29:987–991
12. Peng Y, Leung HCM, Yiu SM, Chin FYL (2011) T-IDBA: a de novo iterative de Bruijn graph assembler for transcriptome. In: Bafna V, Sahinalp SC (eds) Research in computational molecular biology. RECOMB 2011. Lecture notes in computer science, vol 6577. Springer, Berlin, Heidelberg
13. Zerbino DR, Birney E (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res 18:821–829
14. Simpson JT, Wong K, Jackman K, Schein JE, Jones SJ, Birol I (2009) ABySS: a parallel assembler for short read sequence data. Genome Res 19(6):1117–1123
15. Butler J, MacCallum I, Kleber M, Shlyakhter IA, Belmonte MK, Lander ES, Nusbaum C, Jaffe DB (2008) AllPaths: de novo assembly of whole-genome shotgun microreads. Genome Res. 18:810–820
16. Namiki T, Hachiya T, Tanaka H, Sakakibara Y (2012) MetaVelvet: an extension of Velvet assembler to de novo metagenome assembly from short sequence reads. Nucleic Acids Res 40(20):e155
17. Grabherr M (2009) Full-length transcriptome assembly from RNA-Seq data without a reference genome. Nat Biotechnol 29:644–652
18. López V, del Río S, Benítez J, Herrera F (2014) Cost-sensitive linguistic fuzzy rule based classification systems under the MapReduce framework for imbalanced big data. Fuzzy Sets Syst 258:5–38
19. Miner D, Shook A (2012) MapReduce design patterns: building effective algorithms and analytics for Hadoop and other systems. O'Reilly Media, Inc., Sebastopol, CA
20. Dean J, Ghemawat S (2003) MapReduce: simplified data processing on large clusters. In: Proceedings. of Symposium on opearting systems design and implementation, vol 6, pp 1–10
21. Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: OSDI 2004
22. Yinan W, Renner DW, Albert I, SzparaL ML (2015) VirAmp: a galaxy-based viral genome assembly pipeline. GigaScience 4:19
23. Chang Z, Li G, Li J, Zhang Y, Ashby C, Liu D, Cramer C, Huang X (2015) Bridger: a new framework for de novo transcriptome assembly using RNA-seqdata. Genome Biol 16:30
24. Hernandez D (2008) De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. Genome Res 18:802–809
25. Wang S, Cho H, Zhai CX, Berger B, Peng J (2015) Exploiting ontology graph for predicting sparsely annotated gene function. Bioinformatics 31:i357–i364
26. Yuzhen Y, Haixu T (2016) Utilizing de Bruijn graph of metagenome assembly for metatranscriptome analysis. Bioinformatics 32(7):1001–1008
27. Christopher BB (1997) dentification of genes in human genomicdna. Ph.d. Thesis. Stanford University, Stanford, CA,USA
28. Gens P, Enrique B, Roderic G (2000) Geneid in drosophila. Genome Res 10:511–515
29. Arthur D, Kirsten B, Edwin P, Steven S (2007) Identifying bacterial genes and endosymbiontdna with glimmer. Bioinformatics 23:7
30. Ewan B, Michele C, Richard D (2004) Gene wise and genome wise. Genome Res 14:988–995
31. Leila T, Oliver R, Saurabh G, Alexander S, Michael B, Serafim B, Burkhard M (2003) Agenda: homology-based gene prediction. Bioinformatics 19:1575–1577
32. Green P, Lipman D, Hillier L, Waterston R, States RD, Claverie JM (1993) Ancient conserved regions in new gene sequences and the protein databases. Science 259:1711–1716
33. Noguchi H, Park J, Takagi T (2006) MetaGene: prokaryotic gene finding from environmental genome shotgun sequences. Nucleic Acids Res 34(19):5623–5630
34. Hoff KJ, Lingner T, Meinicke P (2009) Orphelia:predicting genes in metagenomic sequencing reads. Nucleic Acids Res 37:W101–W105
35. Besemer J, Borodovsky M (1999) Heuristic approach to deriving models for gene finding. Nucleic Acids Res 27(19):3911–3920
36. Yang B, Peng Y, Leung H, Yiu SM, Qin J, Li R, Chin FYL (2010) Metacluster: unsupervised binning of environmental genomic fragments and taxonomic annotation. In: Proceedingsof the first ACM international conference on bioinformatics and computational biology, pp 170–179
37. Yang X, Zola J, Aluru S. (2011) Parallel metagenomic sequence clustering via sketching and maximal qQuasi clique enumeration on map-reduce clouds. In: Parallel and distributed processing symposium (IPDPS), 2011 IEEE International, pp 1223–1233
38. Olston C, Reed B, Srivastava U, Kumar R, Tomkins A. Pig latin: a not-so-foreign language for data processing. In: SIGMOD pp 1099–1110
39. Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Zhang N, Anthony S, Liu H, Murthy R (2010). Hive-a petabyte scale data warehouse using hadoop. In: ICDE, pp 996–1005
40. Chaiken R, Jenkins B, Larson PA, Ramsey B, Shakib D, Weaver S, Zhou J (2008) Scope: easy and efficient parallel processing of massive data sets. PVLDB 1(2):1265–1276
41. Río S, López V, Benítez J, Herrera F (2014) On the use of MapReduce for imbalanced big data using Random Forest. Inf Sci 285:112–137
42. Birney E, Zerbino DR (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res 18:821–829
43. Pevzner PA, Tang HX, Waterman MS (2001) An Eulerian path approach to DNA fragment assembly. Proc Natl Acad Sci USA 98(17):9748–9753
44. Zerbino DR, Birney E (2008) Velvet: algorithms for de novo short read assembly using de bruijn graphs. Genome Res 18(5):821–829
45. Limasset A, Cazaux B, Rivals E, Peterlongo P (2016) Read mapping on de Bruijn graphs. Bioinformatics 17(1):237

46. Myers EW (2005) The fragment assembly string graph. Bioinformatics 21:ii79–ii85
47. Myers EW, Sutton GG, Delcher AL et al (2000) A whole-genome assembly of *Drosophila*. Science 287:2196–2204
48. Gross JL, Yellen J (2004) Handbook of graph theory. CRC Press LLC, Boca Raton
49. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51:107–113
50. Dean J, Ghemawat S (2010) Mapreduce:a flexible data processing tool. ACM 53:72–77
51. Benkrid K, Liu Y, Benkrid A (2009) A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment. IEEE Trans Very Large Scale Integr Syst 17(4):561–570
52. Edgar RC (2010) Search and clustering orders of magnitude faster than blast. Bioinformatics 26(19):2460–2461
53. Li W, Godzik A (2006) Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. Bioinformatics 22(13):1658–1659
54. Broder AZ, Charikar M, Frieze AM, Mitzenmacher M (1998) Min-wise independent permutations. In: Proceedings of the thirtieth annual ACM symposium on theory of computing pp 327–336
55. Zhao Y, Karypis G (2001) Criterion functions for document clustering: experiments and analysis. Technical report, Department of Computer Science, University of Minnesota, Minneapolis
56. Needleman S, Wunsch C (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol 48(3):443–453
57. Smith T, Waterman M (1981) Identification of common molecular subsequences. J Mol Bwl 147:195–197
58. Hugenholtz P, Tyson GW (2008) Microbiology: metagenomic. Nature 455(7212):481–483
59. Chatterji S, Yamazaki I, Bai Z, Eisen J (2008) Compostbin: a dna composition-based algorithm for binning environmental shotgun reads. In: Annual international conference on research in computational molecular biology, Springer, pp 17–28
60. Khan I, Kamal S, Chowdhury L (2015) MSuPDA: a memory efficient algorithm for sequence alignment. Comput Life Sci 8(1):84–94
61. García S, Cano JR, Herrera F (2008) A memetic algorithm for evolutionary prototype selection: ascalingupapproach. Pattern Recognit 41(8):2693–2709
62. Price KV, Storn RM, Lampinen JA (2005) The differential evolution algorithm. In: Differential evolution: a practical approach to global optimization, pp 37–134. ISBN 978-3-540-31306-9
63. Neri F, Tirronen V (2009) Scale factor local searching differential evolution. Memet Comput 1(2):153–171