7-25-2013

Distributed Multi-Objective Evolutionary Algorithm For Dynamic Multi-Characteristic Social Networks Clustering

Mustafa Hussein Hajeer

DISTRIBUTED MULTI-OBJECTIVE EVOLUTIONARY ALGORITHM FOR DYNAMIC MULTI-CHARACTERISTIC SOCIAL NETWORKS CLUSTERING

by

Mustafa Hussein Hajeer

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Major: Computer science

The University of Memphis

August 2013

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Dipankar Dasgupta, for his unconditional encouragement and support, and for giving me the opportunity to change my life. He guided my first steps in the academic and research world, allowing me to work with him and his team, believing in my work and in me, and advising me like a parent.

I would also like to thank the members of my committee, Dr. Scott Fleming and, Dr. King-Ip Lin. Their comments and feedback helped me to improve the content of this thesis.

I am greatly thankful to my friends and colleagues Alka Singh and Abdullah Abu Hussein, especially for their help in introduceing my work and helping me with the problem I am solving, and for editing this manuscript, and patiently teaching me how to write with clarity. Thanks for believing in me before I believed in myself. My most sincere thanks to both of you.

I am grateful to all my colleagues in our research group at the University of Memphis, especially to Denise Ferebee. Our meetings and discussions opened my mind to new ideas, and thanks to Kul Subedi for helping me setting up the technical environment. Abhijit Nag and Sanjib Shuvro, my thanks to all of you.

I will always remember my professors and colleagues in computer science department for their generous support during my years in the University of Memphis, and for guiding me.

Finally, to my uncle, Samir Hujier, my mother and my father, Muntaha and Hussein Hajeer, my brothers and my sisters who have been my greatest supporter. Without their unconditional encouragement, love and support, I would have never started this life-changing journey. My sincere thanks to you all.

ABSTRACT

Hajeer, Mustafa Hussein. M.S. The University of Memphis. August 2013. Distributed Multi-objective Evolutionary Algorithm for Dynamic Multi-Characteristic Social Networks Clustering. Major Professor: Dipankar Dasgupta

In this information era, social media and online social networks have become a huge data source. The social network perspective provides a clear way of analyzing the structure of whole social entities. These social media and online social networks are a virtual representation of real life as they represent real life relations between social actors (people). The primary focus of this study is to propose an algorithm and its implementation for clustering of multi-characteristic dynamic graphs in general, and multi-characteristic dynamic online social networks in specific. Social networks are typically stored as graph data (edges lists mostly), and dynamically changes with time either by expanding or shrinking. The topology of the graph data also changes along with the values for the relationships between nodes. Several algorithms were proposed for clustering, but only few of them deals with multi-characteristic and dynamic networks. Most of the proposed algorithms work for static networks or small networks and a very small number of algorithms work for huge and dynamic networks. In this study a practical algorithm is proposed which uses a combination of multi-objective evolutionary algorithms, distributed file systems and nested hybrid-indexing techniques to cluster the multi-characteristic dynamic huge social networks. The results of this work show a fast clustering system that is adaptive to dynamic interactions in social networks also provides a reliable distributed framework for BIG data analysis

Table of Contents

LIST OF FIGURES

1. INTRODUCTION

1.1 Statement of problem

Social media and online social networks have become a huge data source and virtual representation of physical life and physical relations between people. These social media and online social networks contains very important information, which helps in studies such as social behavior, online marketing and studies about web characteristics. Recently they have attracted much attention among the research field and research groups.

Communities can be defined as a group of individuals who interact within a group with each other more frequently than with those outside the group. Studies on these communities cannot only help in study of the above mentioned areas, but also in areas concerned with security issues. Understanding how these groups are formed and how it changes over the time, by classifying nodes in a network, based on some characteristics, can help in applying theories and techniques to improve these fields. The social networks and online dataset are a combination of interconnected distinct groups. These distinct groups, needed to be extracted from this one single large group of network. The study of inferring these groups is called network clustering; which can show the real clusters (groups) within any dataset, such as social network data. Social networks are dynamic in nature, which means it changes overtime. In an online social network world scenario, groups are changing dynamically, therefore the new direction of

network and graphs clustering are directed as multi-characteristic dynamic networks clustering.

Graph or network clustering has proved to be NP-complete problem, [1] which means there is no known efficient way to locate a solution, also the time required to solve this problem increases very fast with the input (network). The time required to solve even moderately sized versions of this problems can easily reach into the thousands of hours. But social networks grows really fast and it is huge in size, in addition to that it is dynamic in nature; which means with the time needed to cluster the network, the network has already changed, and the new formed network may be totally different than the recentlyclustered network.

In addition to the above explanation, social networks are multi-characteristic in its nature. There are multiple ways in which two nodes can be connected, for example in case of Facebook(a major social networking site), a node(a person/page) can add another node as a friend which is a connection, however a node can also send a message to another node, and all of this is different type of connection, and it can be combined together to form links with values for each characteristic.

All of the above different types of connections with multiple characteristics result a huge and complex datasets, which is impossible for clustering algorithms to cluster and produce results in reasonable time.

1.2 Indices for Network Clustering

In this work we referred to graph G(V,E), as an undirected graph. Let |V|=m, |E|=n and C = (C_1, C_2, C_3,…, C_j) as a partition of V as a disjoint sets. We call C a clustering of G containing j clusters. The number of clusters j has a minimum of j=1, when C contains only one subset C_1 = V, and a maximum of j=m when every cluster C_k contains only one vertex. We identify the cluster C_k as a subgraph of G. The graph G[C_k]:=(C_k,E(C_k)), where E(C_k)={{V,W} ∈ E:V,W ∈ C_k}. Then E(C) = $\cup_{K=1}^{j} E(Ck)$ is the set of intra-cluster edges and E\E(C) is the set of inter-cluster edges. The number of intra-cluster edges denoted by m(C) and $\bar{m}$(C) is the number of inter-cluster edges.

As an input the social network assumed as a set of graphs SNG=(G_1,G_2,G_3,….,G_Z), and the set of graphs-clustering SGC=(C_{G1},C_{G2},…,C_{GZ}) where each graph G_i has its own clustering C_{Gi} and satisfies all the conditions mentioned for G and C respectively, where Z is the number of characteristics of the network's dataset, graphs (G_1,G_2,G_3,….,G_Z) have the same set of V but different set of E. each C_{Gl} is an objective to achieve in this work.

The goal is to find SGC using multi-objective optimization and to combine them into one clustering SNC =(SNC_1, SNC_2, SNC_3,…..,SNC_X), where SNC:=$\cup_{L=1}^{Z} C_{GL}$. The set of clustering for social network SNC is not necessarily disjoint, but it is a union of sets where each set is a group of disjoint subsets.

Social networks representation with all of its characteristics can lead to a dataset of a huge graph, however, we represented the social network as a set of graphs rather than one graph, each graph represents one characteristic. The set

of graphs have the same set of vertices, but different number of edges. The proposed algorithm takes the social network as a multi-characteristic dataset, then partitions it into set of graphs SGC, where each graph contains edges for only one characteristic. Then each graph G_I is clustered individually by an edge removal algorithm to produce disconnected graph represented by clustering C_{GI}, then by measuring the strength of these clusters. After clustering each graph G_I, we combine elements of each clustering in SGC into one clustering SNC, to produce an overlapped clustering where clusters SNC_1 to SNC_X are not necessarily disjoint.

For the clustering process of each graph, an evolutionary algorithm has been used because of the huge search space for all graphs. On the other hand, Hadoop distributed file system has been used to provide performance and speed by partitioning the large datasets into smaller blocks.

1.3 Literature Review

Several algorithms were proposed as a solution for clustering problem, the most popular algorithms and frameworks were gathered and classified based on their positive and practical aspect, as well as their drawbacks. In our algorithm, we gathered all of the drawbacks of existing solutions and overcame them with a collection of techniques, which are mentioned in the next sections as we continue explain our approach.

The list below shows different clustering and data mining techniques along with their advantages and drawbacks.

❖ In “basic concept of data mining, clustering and genetic algorithm” [24], Tsai-Yang Jea reviewed a basic evolutionary algorithms, whose concept have been analyzed with the following results:

Advantages: Fast results as a clustering algorithm, since it doesn’t search the whole space of solutions.

Drawbacks: Final clusters don’t show global optimization, the chromosomes represent the whole space each time, and need parameters as inputs, like number of clusters to start. Also the search for node’s similarities itself is time consuming process, and it has to be done in $C*N^2$ time, where N is number of nodes and C number of chromosomes. The amount of time taken to produce the result makes it inapplicable for huge datasets.

❖ Petra Kudová developed (CGA) an evolutionary algorithm for clustering in his paper “Genetic algorithm clustering”[20], published from Academy of Sciences of the Czech Republic, ETID, on 2007. After a deep analysis the advantages and drawbacks for his research have been summarized as follows:

Advantages: faster than regular search approach, and looks for global optimization.

Drawbacks: Similar to Tsai-Yang Jea [4], the search for node’s similarities itself is time consuming process, and it has to be done in $C*N^2$ time, where N is number of nodes and C number of chromosomes. Also each chromosome copies the whole search space as a list, and that is C*S where S is the search space size (billions of nodes and connections in real life). That makes the execution time and space for this algorithm impossible for huge datasets processing.

1.4 General Literature

There are many other related works (listed below) which demonstrate different advantages of clustering algorithms; nevertheless, these approaches have almost similar drawbacks as discussed in reference to the algorithms mentioned in above section. The major drawback was that these approaches needed some parameters to feed, for example, number of clusters, size of clusters or number of generations that are needed by evolutionary approaches. It is suggested that these parameters should be found from the dataset, not given as an input parameters. Also these inputs aren't available; hence, it changes the solutions and can result false solutions.

Some other drawbacks which are critical is that these approaches copy the search space many times, thus result in demanding huge space for processing which is an impractical approach in real life. On the other hand, the results are produced slowly and by the time needed for processing the whole network, the network has already changed because of its dynamic behavior. Below is another list of papers that share the same disadvantages:

- Evolutionary Clustering and Analysis of Bibliographic Networks [15]
- Multi-objective Evolutionary Algorithms for Dynamic Social Network Clustering [13]
- A Multi-objective Hybrid Evolutionary Algorithm for Clustering in Social Networks [6]
- A framework for analysis of dynamic social networks. [23]

- An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. [21]
- Genetic algorithm and graph partitioning. [22]
- Multiobjective evolutionary clustering of web user sessions. [18]
- Dynamic algorithm for graph clustering using minimum cut tree. [15]
- Community detection in complex networks using genetic algorithm. [2]
- A new graph-based evolutionary approach to sequence clustering. [16]

Table1 on the next page summarizes the evaluation of previous work along with its characteristics and compares it with our approach.

Table 1: Clustering Algorithms Evaluation

Method Reference	Search overhead	Practical dataset size	Multi-characteristic	Dynamic networks	Evolutionary based	Extra parameters needed
Clustering and Genetic Algorithms [24]	Yes	~1000	No	No	Yes	Yes
Genetic algorithm clustering [20]	Yes	~900	No	No	Yes	Yes
Evolutionary Clustering and Analysis of Bibliographic Networks [15]	Yes	~700	No	No	Yes	Yes
Multi-objective Evolutionary Algorithms for Dynamic Social Network Clustering [13]	Yes	~600	Yes	Yes	Yes	No
A Multi-objective Hybrid Evolutionary Algorithm for Clustering Social Networks [6]	Yes	500 to 700	Yes	No	Yes	Yes
Multiobjective evolutionary clustering of web user sessions [18]	Yes	~700	No	No	No	No
Proposed approach	No	~10000 on each reducer	Yes	Yes	Yes	No

None of the previous works show interest in distributed or parallel approach, thus resulting in clustering and search overhead. The approach in this study uses Hadoop distributed file system and a combination of hybrid hashing and evolutionary algorithms; hence, resulting in a fast, robust and practical solution according to the dataset size.

1.5 Distributed Evolutionary Clustering

Many metrics have been used in traditional social network analysis as a measurement to determine the strength of each group in the network. Some of these metrics have been used in the algorithm itself to find groups (clusters), such as max-clique, k-clique, modularity, k-club etc., where Lei Tang and Huan Liu, Morgan & Claypool explains these metrics in his work “Community Detection and Mining in Social Media”[10] discussed in section 1.6 below.

1.6 Metrics of Measurements

1.6.1 The Maximum Clique

Cliques are the complete graphs where every node is connected to all other nodes in the graph. These are the measurements of how strong the groups are. This technique tries to find the maximum sub graphs that are cliques in nature. The problem itself has been proved to be NP hard as Lei Tang and Huan Liu, Morgan & Claypool discussed in “Community Detection and Mining in Social Media”[10].

Figure 1 below illustrates synthetic network to define the maximum clique. In this figure Nodes 1,2,3 and 4 form a clique, also any three nodes combination of 1,2,3 and 4.

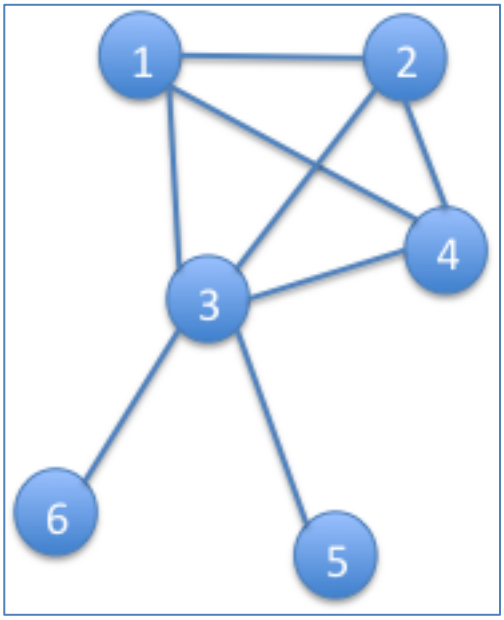


Figure 1: Clique Definition

Nodes 1, 2, 3 and 4 form the maximum clique, node 5 or node 6 can't be added since the group loses its definition. Smaller cliques can be found like 1, 2 and 3; however, the maximum number of nodes, which forms a clique in this network, is 4.

Clustering solutions can be measured based on how many groups of maximum cliques the solutions have. The more number of groups the solution has, the stronger the solution is, which is also known as "*maximization problem*".

1.6.2 K-clique and K-club

As per definition, clique of size k is the graph where each node maintains degree >= k-1, similarly k-clique is defined as the maximal sub graph, where the longest distance between any pair of nodes <= k. The k-club is defined as a substructure of diameter <= k. For the network illustrated in figure 1, it can be said that:

- 2-clique contains nodes {1, 2, 3, 4, 5 and 6}.
- 2-clubs contains nodes: {1, 2, 3, 4 and 5} or {1, 2, 3, 4 and 6}.

To achieve stronger clustering with stronger groups, the objective is to minimize k for the sub graphs and maximize the size of the group at the same time. The overall objective is to combine this relation for all groups in one value, and compare it with other solutions.

1.6.3 Modularity

Lei Tang and Huan Liu, Morgan and Claypool define modularity in "Community Detection and Mining in Social Media"[10] as the strength of a community partition by taking into account the degree of distribution which can be calculated as follows:

The strength of community can be calculated by formula:

$$\sum_{i \in C, j \in C} \left(A_{ij} - \frac{d_i d_j}{2m} \right),$$

Where, $\frac{d_i d_j}{2m}$ = expected number of edges between nodes i and j with the degree d_i for node i and d_j for node j.

A_{ij} is the number of real edges between node *i* and node *j*, and *m* is the number of edges in the network.

The following formula is used to calculate the modularity, and maximize the strength of the communities' structure in the network:

$$Q = 1/2m \sum_{l=1}^{K} communiity\ strength$$

Where, Q = modularity measure and k = number of communities.

By maximizing the objective *Q* a better communities structures achieved as a solution.

1.7 The Motivation

The major problem in the traditional combine and test, for the search algorithms (the sub problem of clustering), is that the search space is too huge and for large networks it is impossible to apply. It was observed that the previous works including evolutionary clustering algorithms, need to search for connections, and this process takes place several times for each node in the network either for the clustering process or for the evaluation process. On the other hand, traversing the network at each node and looking for all neighbors in a huge dataset is an overhead itself.

For the above reasons, a new distributed evolutionary algorithm and a hybrid HashMap technique was developed for a very fast search.

1.7.1 The Evolutionary Clustering

Since the problem have a huge search space, and the clustering problem is proved to be NP-Complete problem, as per Jiri Sima and Satu Elisa Schaeffer proved in their work "On the NP-Completeness of Some Graph Cluster Measures". We chose evolutionary algorithms to find approximation or close to

the optimal solution and because of the dynamic nature of the social networks, it was decided to develop an evolutionary algorithm that clusters the network in a fast way and uses the metrics above as a fitness function and evaluation for solutions. Since social networks are full of noise in terms of data, the chromosome encoding developed as a list of weak and noisy edges to be removed, “edge removal and cut based algorithm”.

Most of traditional evolutionary algorithms were developed in a way that the user should provide some parameters about the network, and the algorithms were processing the data based on these parameters. In this study, it is believed that the user shouldn’t provide these parameters, but it should be extracted for the network itself. Hence an algorithm is developed in such a way that the network –edges list and its characteristics- has to be the only input, and that no interfering or noisy data were read from the user. During the execution of the algorithm, it receives the changes in the network and reflects it on the algorithm inputs-the edges list- then the algorithm produce results and create output solutions based on the most recent network inputs.

Jmetal 4.3 is a powerful object-oriented Java-based framework aimed at multi-objective optimization by using metaheuristics. jMetal provides a rich set of classes which can be used as the building blocks of multi-objective techniques.as per Antonio J. Nebro, Juan J. Durillo “jMetal 4.3 User Manual” [18] In this study jMetal is used to develop the evolutionary algorithm.

1.7.2 The Job Distribution and Parallelism

The clustering problem is an overhead itself, and during the process there is a lot of search jobs. To make the process faster and less memory demanding, a parallel distributed evolutionary algorithm is developed. Such algorithms need synchronization mechanism so the solutions can be produced, In this study, the algorithm is synchronized on a population level (discussed in architecture section):each population will move to the next one after complete evaluation only.

Evolutionary algorithms made the adopted approach work faster and distributed evolutionary computing made it possible to process even faster; hence, resulting in less clustering overhead and practical to cluster huge dynamic datasets.

Hadoop distributed file system HDFS provided a robust platform for our algorithm, where dataset is distributed among multiple computers, each computer works on the data it has, based on the job it receive from the master computer. The master computer is responsible for receiving results from other DaraNodes and combines them into one population, also receiving the next population from client and then submitting jobs to TaskTrackers again.

The master computer is denoted by NameNode or task tracker. Computers in the network denoted by DataNode or job tracker. To make it less confusing in the next sections, the network of computers running HDFS will be referred as “Cluster” and the communities in the dataset will be referred as “Groups”.

Figure 2 illustrates a diagram of (HDFS) Hadoop distributed file system structure.

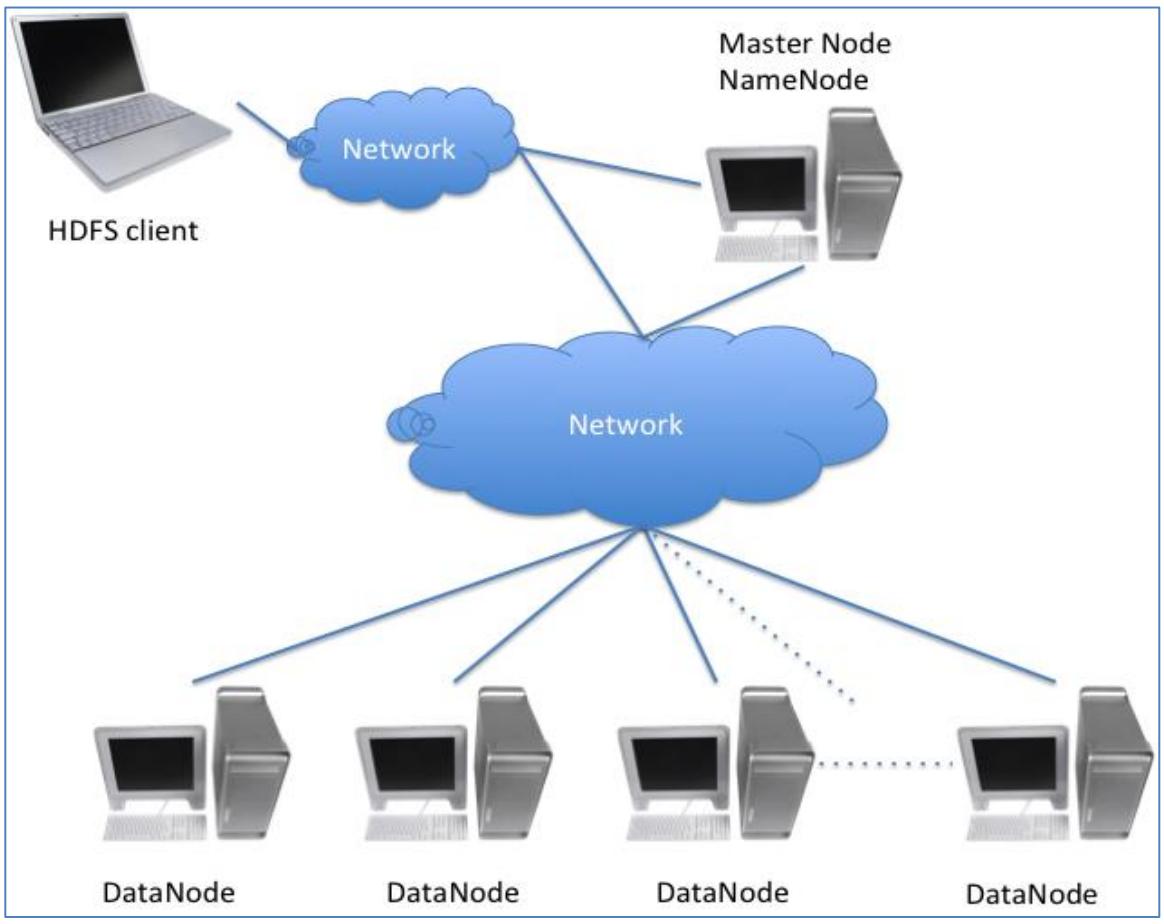


Figure 2: Hadoop Network

1.7.3 Hybrid *Hashmap*

The main overhead in clustering problem is the search in huge datasets, in the process of finding connections, and during the evaluation process. With undirected graphs or network, the problem arises where each connection can be directed in both sides, and the search problem takes double the time of the

regular directed graphs search. The dynamic nature of social networks made faster search impractical for sorting process.

In our approach, a hybrid Hashmap technique is developed which converts the edges list of the network to a hashmap matrix that represents it. Each node was given an ID as a search index, where the search value is a small Hashmap representing all connected nodes with characteristics values as a string. This Hybrid Hashmap made it possible to group distinct nodes in a single group and at the same time evaluate the group strength in one cumulative step. A faster way to do depth first search is to traverse the disconnected groups in one network and to evaluate in a very fast time for huge networks. When the search process starts, the next node to add to the group can be found in less than one millisecond rather than taking a long time to search the dataset. The search is an overhead itself taking around $2N^2$ for each solution to cluster and evaluate.

Table 2 below shows a snapshot of a synthetic dataset which contains distinct groups after removing noise connections with the values of characteristics of the network.

Table 2: synthetic dataset edges list

Node A	Node B	Number of emails	Number of posts	Number of comments
1	2	4	4	4
1	3	3	3	3
1	4	4	4	4
1	5	4	5	5
5	4	3	4	4
5	3	4	3	3
2	3	3	3	3
2	4	3	3	3
6	7	5	4	3
6	8	3	5	4
6	9	4	4	4
7	8	4	3	5
7	9	3	4	4
8	9	4	5	5

Figure 3 shows its hybrid Hashmap representation of the same dataset where we converted the edges list to make it fast for search and merge process.

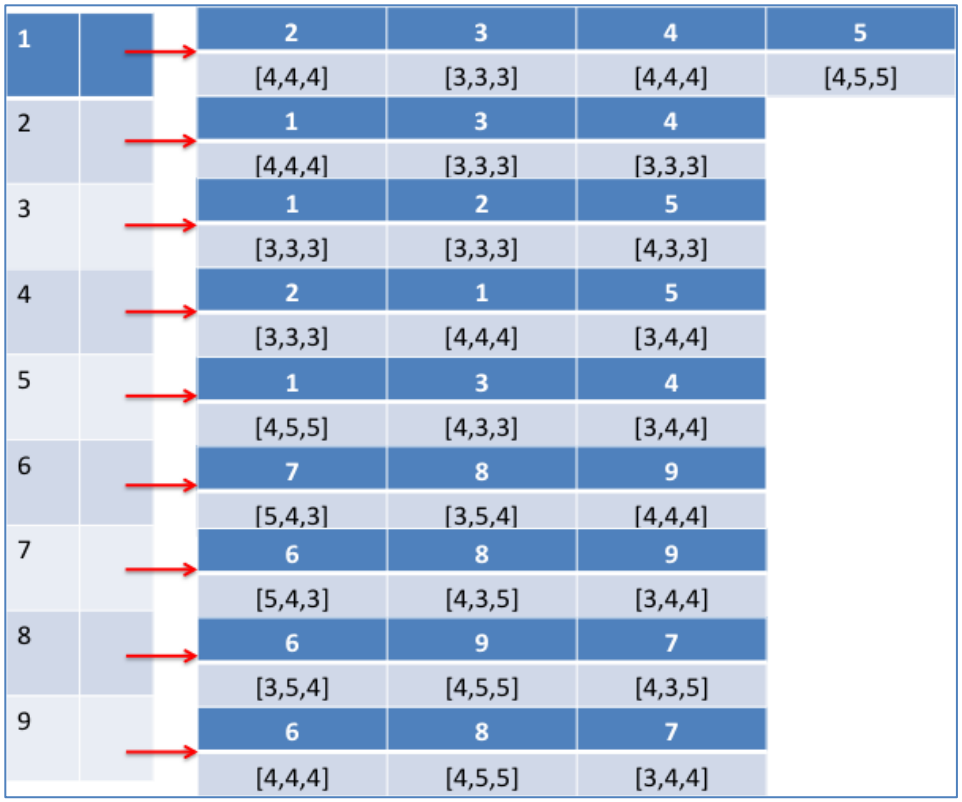


Figure 3: Hybrid Hashmap representation of table 2

1.8 Chapter 1 Summary

The problem statement has been discussed in this chapter and the motivation for the proposed solution. It was shown that it is better to represent the multidimensional space of social networks by multiple graphs, each represent one characteristic, so the solution space becomes smaller to search. It was shown also that the optimization and approximation are good techniques for such problem with large search space, also the distribution of such algorithms reduce the overhead of searching and clustering process.

2. Evolutionary Algorithms

Evolutionary algorithms are a collection of problem solving techniques that provide a very suitable way to search for solutions in large space problems. The idea came from natural selection in biology, where the implementation in computer science is that the good solutions in the search space stays and produce better solutions, and bad solutions fades. The steps of evolutionary algorithm is explained as:

1. The solution of the problem is encoded in a Geno representation called chromosome.
2. Multiple solutions are generated and the group of solutions called population.
3. The best solutions “called parents” is selected and crossover is done to produce child solutions searching for better solutions.
4. Mutation is done for the generated solutions to make the algorithm expand in the search space and look for diversity in the population.
5. New population is generated from the child solutions
6. Step 3 is repeated until criteria are met.
7. Best solution is extracted in the last population.

The evolutionary algorithms were found to solve problems with close to optimal solutions, usually for NP hard problem or NP complete problems, and the result solution is an approximation.

For our clustering problem in this study, Jmetal framework is used because it provides a robust and reliable collection of algorithms. Jmetal is an

easy to use, extensible, and flexible. A problem class and encoding class is created, and then the algorithm class is overridden to match the needs of providing a new distributed way to evaluate the solutions. It is further explained in the architecture section.

To provide an overview of Jmetal, the framework entities and connections between these entities are represented in figure 4 as per Antonio J. Nebro, Juan J. Durillo, jMetal 4.3 User Manual [4].

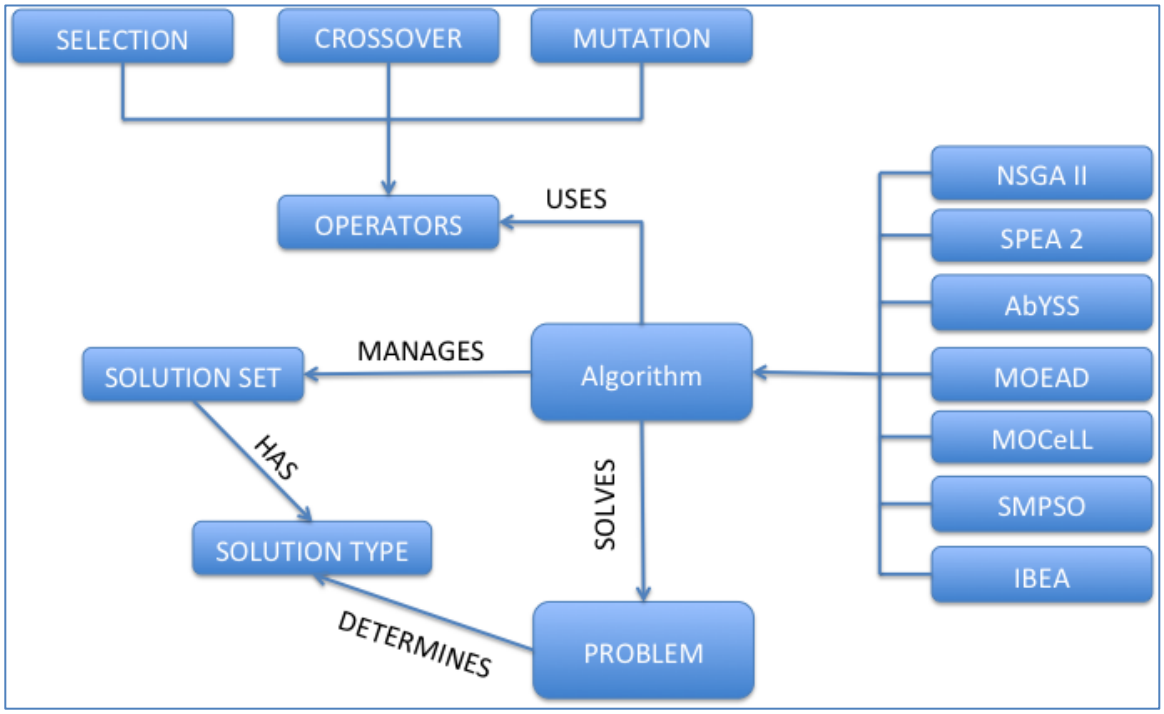


Figure 4: Jmetal architecture diagram

3. Hadoop Distributed File System (HDFS)

3.1 HDFS Architecture

Hadoop distributed file system (HDFS) is an open source file system. It is a file system that can combine multiple computers and show them as one system. Also it allows the user to query and process large datasets in short time. The capabilities of the system increases with the number of computers added to HDFS cluster. It can work with unstructured data as well as collection of structured data. The main idea behind this file system is to split large datasets into smaller ones and spread them over the HDFS cluster, with some redundancy mechanism to provide a strong reliability in the cluster.

HDFS cluster provides a collection of services. The primary service among all is the MapReduce, where any process (also called job) can be submitted to the master computer (called the master node), and the master computer by its turn spread the job into tasks for each computer in the HDFS cluster (called data nodes). Each computer performs mapping and reducing for the task assigned to it on the data it have. After mapping and reducing, the result is returned to the master node, and the master node returns result to the submitting user as files.

For better understanding, figure 5 illustrates a simplified HDFS cluster architecture with the names of its components.

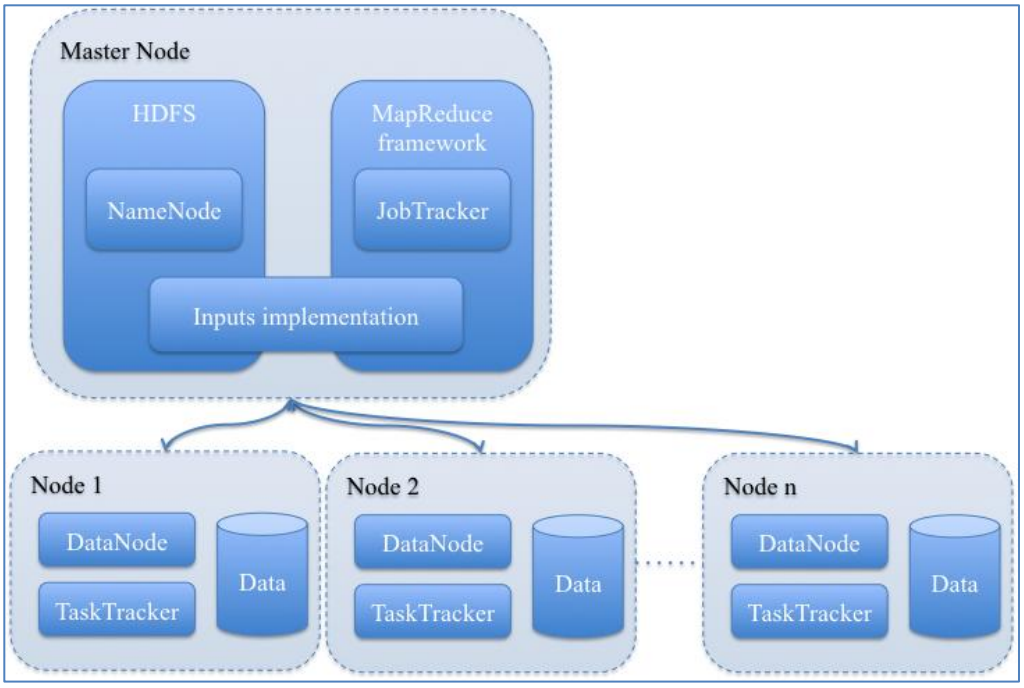


Figure 5: HDFS Architecture

HDFS has two main components:

- The core as the master node:
  - NameNode: Maintains mapping of file names, blocks and data nodes.
  - Job tracker: Tracks resources and schedules jobs across task tracker nodes.
- Nodes:
  - DataNodes: contains the files blocks and continuously sends heartbeat to the master node to confirm its status.
  - Task tracker: Runs tasks (work units) within a job.

3.2 MapReduce

MapReduce is the main service that runs on Hadoop distributed file system HDFS. MapReduce can be used to query from serial files distributed on HDFS cluster. This process made it easier to query huge datasets (petabytes of data) in a faster way than normal indexed databases.

Before initializing any MapReduce process, data have to be ready in the HDFS cluster. To upload the data into the HDFS cluster, the command “put” can be used along with the file name and the destination directory in HDFS file system. When the “put” command is called, a copy of the data file is moved to the HDFS cluster, then the file is divided into blocks across the DataNodes in the cluster. The hadoop file system provides redundancy mechanism to provide reliability and availability in hardware failure situations and data blocks information saved in the cluster. Each DataNode sends heartbeat periodically to the NameNode to confirm the status of the DataNode.

After the file is uploaded, it becomes ready to use. MapReduce operations can occur based on RPC (remote procedure calls) for the user to the Job Tracker. The user defines a MapReduce functions and passes them to the Job Tracker. Then the Job Tracker spreads the map function as tasks to Task Tracker. Each Task Tracker works only on the data blocks it have on its DataNode. The map function reads the data and maps it to pair of <Key, value> and passes it to the reducer function. Before the reducer function performs the reduce operation, shuffle operation exchanges the pairs between TaskTracker, where each reducer takes one Key or more to work on. The reducer then starts

reducing the collection of <key, value> pairs that came from the map function into a new single pair of <NewKey, NewValue>. The reduce operation is user defined which writes these results into files and saves it in the HDFS.

Figure 6 illustrates the previous explanation of MapReduce operation on Hadoop distributed file system –HDFS- cluster.

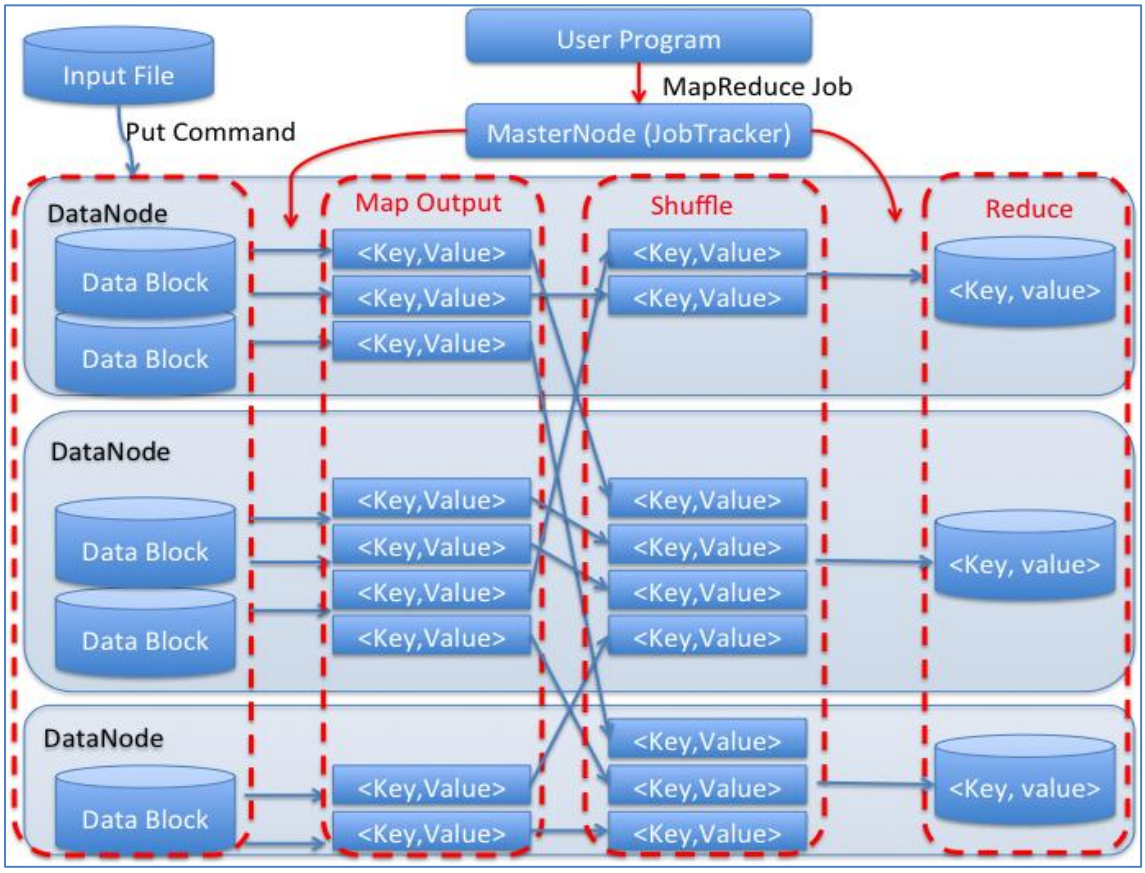


Figure 6: MapReduce operation

Based on figure 6, the pseudo code steps below explain flow of any general MapReduce function call. In the architecture section, it is demonstrated how these steps are mapped to approach adopted in this study by creating new object writer for values.

3.3 MapReduce Steps

- Map and Reduce jobs are received from user program by the JobTraker in the master node of HDFS.
- The JobTraker spreads the MapReduce Tasks to the TaskTracker in all Nodes in HDFS.
- Each TaskTracker reads file blocks contained in its DataNode.
- For each data line read from DataNode, mapper converts it into pair of <Kye, Value> based on user program definition, and writes it into intermediate file.
- <Key, Value> pairs are shuffled and exchanged between Reducers, so each group of the same Key is collected into the same Reducer.
- The Reducer reduces the array of values for each key into one value based on reducer definition from the user program, by emitting the value from the intermediate file and combining it to the NewValue.
- The <NewKey, Newvalue> pairs are written into result files, and saved on HDFS. After this the user can pull it out of the cluster when the user program receives a trigger that the MapReduce operation is completed.

The more nodes in the HDFS cluster, the MapReduce operation can be processed with more speed and efficiency.

3.4 Chapter 3 Summary

Hadoop distributed file system (HDFS) was discussed in details along with its MapReduce process. It was shown that such file systems provide a robust platform for distributed jobs on huge datasets. HDFS provides fast solutions for algorithms distribution with an advantages of handling large processes in a small period of time, which makes it a very suitable file system for clustering algorithms, especially with datasets that have a strong relations between it's components.

The huge amount of queries on the social networks datasets by clustering algorithms made HDFS a very suitable platform that can provide solutions in a practical period of time and without discarding the dynamic nature of social networks.

4. THE Proposed Architecture

4.1 Implementation Details

The proposed framework consists primarily of two main components, the evolutionary component, and the distributed file system. The two components overlaps on each other to provide one framework that takes dynamically changing dataset as an input and splits it into blocks over the distributed file system. The evolutionary part creates chromosomes and is responsible for extracting the parameters, create clustering and evaluating jobs, sending jobs to HDFS cluster, getting the result back to generate new solutions and create new jobs again.

Figure 7 below illustrates the proposed framework architecture on a very high level. The components with its purpose listed after the figure. The rest chapter 4 describes these components in details.

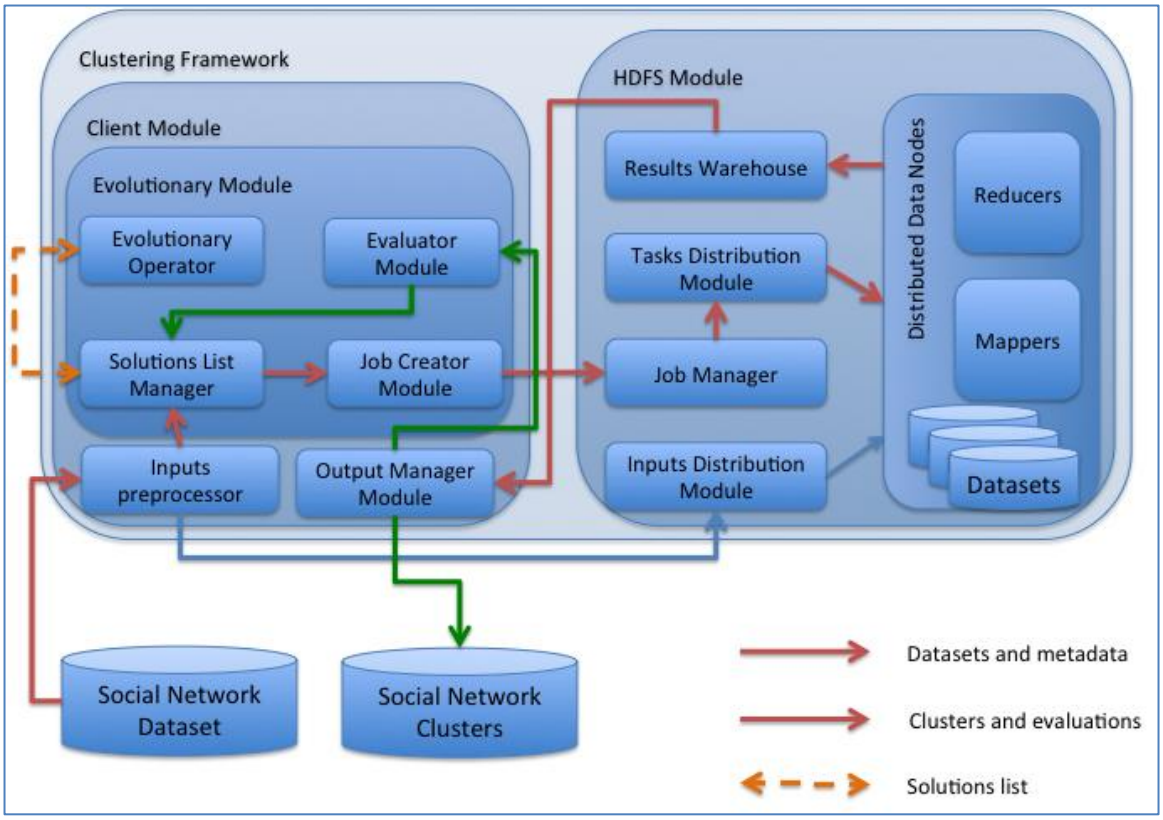


Figure 7: The Proposed framework

The framework composed of two main components:

1. Client Module:
    a. Inputs Preprocessor: process the social network dataset and transform it into multi-dimensional dataset ready for uploading into HDFS.
    b. Output Module: getting evaluation results and sends it to evolutionary module and saves the most recent clusters as clustering results.

c. Evolutionary Module: process and execute the evolutionary algorithm and its operator to find best clustering solutions, it also create clustering and evaluating jobs for HDFS.

2. HDFS module:
    a. Inputs Distribution Module: receives the processed dataset and distributes it over HDFS DataNodes.
    b. Job Manager: receives jobs from evolutionary module and transform it into tasks of Map and Reduce.
    c. Tasks Distribution module: distribute Map and Reduce tasks to TaskTrackers.
    d. Results Warehouse Module: saves clustering results of solutions and its evaluations to be read by output manager module on client module.

Any distributed system needs synchronization. The approach adopted in this study, especially the evolutionary algorithm component, needs synchronization because no solutions can be generated if previous solutions(parents) aren't evaluated. To reduce the overhead in job generation and submission and on HDFS calls, the synchronization is generalized to the simplest level.

The simplest level to which synchronization can be generalized is new population level. The approach can be preceded with the population itself but cannot be moved before evaluating solutions, so the algorithm class in Jmetal is modified to evaluate the population at once rather than solutions level. Each group of solutions is send at once as a clustering and evaluating job to HDFS. The next generation of solutions can only be created after the previous one is already clustered and evaluated. Figure 8 shows a general view of proposed approach and illustrates the synchronization level.

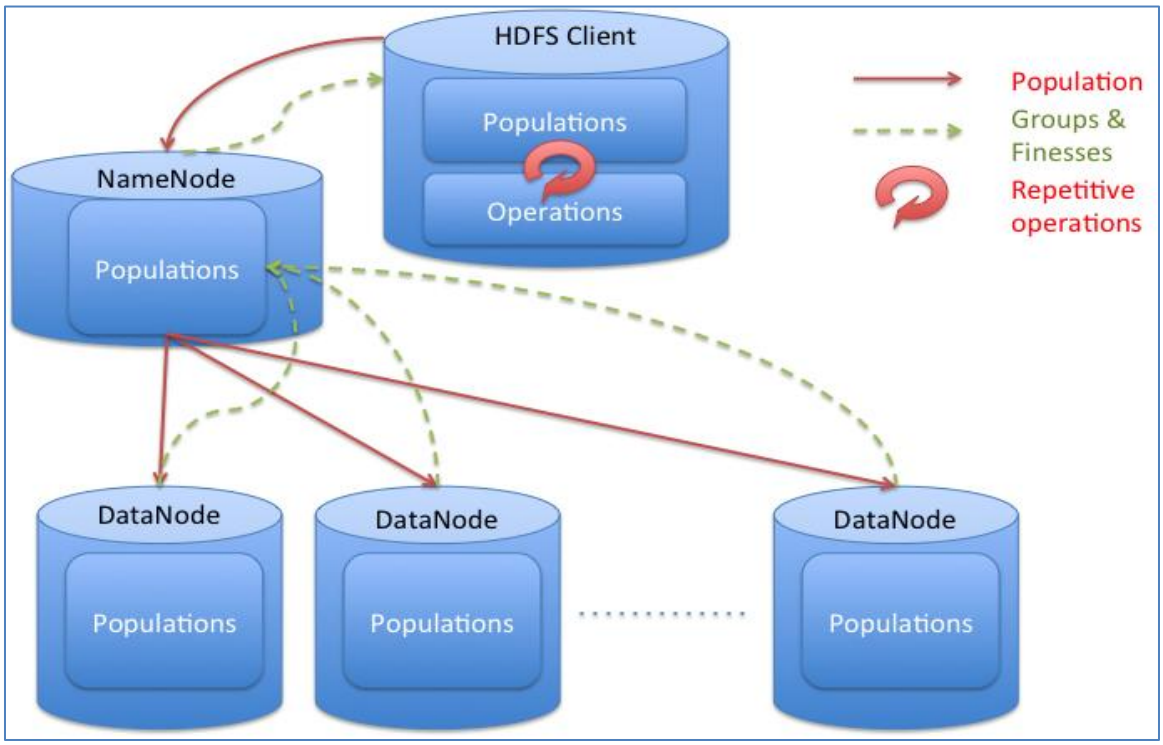


Figure 8: General View of Population-based approach and Synchronization Level

Figure 8 illustrates the pseudo code of the general steps of the algorithm after the file is uploaded in the HDFS and the changes are continuously being uploaded in parallel with the execution of the algorithm. It can be further described as:

1. The problem class generates the inputs for the evolutionary algorithm.
2. User program runs the algorithm by issuing the command execute for the modified algorithm class, and send the problem object with its values to the algorithm object by the execute method.
3. The algorithm class on HDFS client generates the first population chromosomes, and keeps it without fitness values ready for evaluation.
4. The algorithm class running on HDFS client contains the unevaluated population into a job along with number of dataset characteristics, and sends the job as a MapReduce job to the JobTracker on the NameNode.
5. The JobTracker distributes the job to TaskTrakers as tasks.
6. A map and Reduce operation carried out by TaskTrackers and writes the solutions as groups with its fitness's in results files into HDFS.
7. JobTracker triggers the HDFS client running the user program that the job is done and the solutions with evaluations are ready along with group description for each solution.
8. The HDFS client pulls the results form HDFS results files, writes the groups of the best solution into network file as the most recent clustering solution, on the other hand the algorithm takes only the

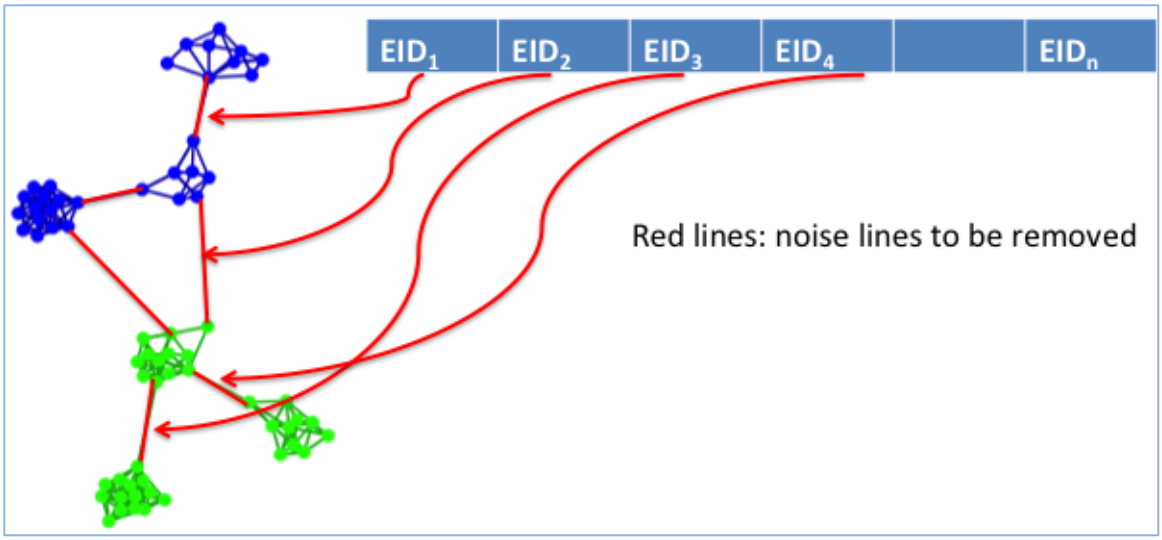
fitnesses along with the solutions as an evaluated population, ready to do GA operations like crossover and mutation, to generate a new unevaluated population. Step 4 is repeated while the program is still under execution.

While the system is running, any changes to the dataset is immediately uploaded and merged with the input dataset on the HDFS. In parallel to the running algorithm, the changes are immediately reflected in new solutions.

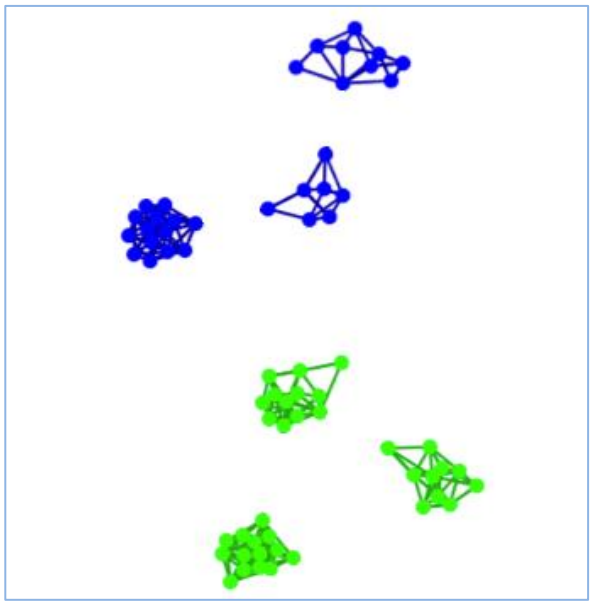
4.2 The Proposed Design of Solution Space

The primary idea of encoding scheme for the chromosome (called solution) is an array of integers that represent the noise edges in the network. We want to find and remove, to create a network of distinct groups without any noisy edges, and then find how strong these groups are as an evaluation for each solution. Each TaskTracker works only on the parts of the chromosome that it have in its block and mark it as removed edge. Each integer is as ID to an edge in the dataset, these IDs created uniquely for each edge before uploading into HDFS.

Figure 9 (A) illustrates the chromosome-encoding scheme where the length of the chromosome is variable. Figure 9 (B) illustrates the network representation for solution in figure 9 (A), since the algorithm is looking for noise edges, its one of the objectives is to find the number of these edges.



(A)



(B)

Figure 9: (A) Chromosome encoding scheme, (B) Network representation of solution (A)

4.3 The Proposed Objective Functions

The algorithm is configured to be parameter-less, while previous work required the user to enter parameters such as number of clusters, clusters size gap, clusters modularity etc. It is believed that these inputs should be derived from the dataset, to make solutions realistic. So these parameters are made as objective functions, and they are added to the problem class in Jmetal framework. The main objective function is composed of an equation, which contains number of groups, number of noise edges removed, and values of each characteristic of the edges itself. Another objective has been added to represent the groups strengths and use these values as multiple fitnesses to evaluate the solution. The formula below shows the main objective:

- $fc = \left(\sum_{groups} \frac{\sum_N Vn}{Gn}\right) - Er$

Where:

- Fc: objective for characteristic C.
- N: edge N.
- Vn: value of characteristic C on edge N.
- Gn: the group size.
- Er: number of edges removed.

N number of objective functions was created based on number of characteristics the network have, and each one of them reflect a different Fc and results different Fc value. Each one of these values are considered to have the

objective to maximize in the problem class in addition to the modularity objective. This formula is developed to remove noise edges. If any edge other than noise is removed, it will results in a lower fitness as a penalty, that lowers number of edges removed and keep the network in the same topology, it also prevents groups of one node to be created.

Number of objectives differs from a network to another, based on what data is obtained from the dataset, especially number of characteristic, that can be gathered about the network.

4.4 Task At TaskTracker Level

Tasks on the TaskTracker level receives a copy of the solutions list (population),then the task is to map the data read from data block into pairs of <Key, Value>, by comparing the data with the solution list received. Each TaskTracker works only on the parts of the solution contained in its data blocks, and we call these parts active parts, the rest parts of the solutions called inactive parts. During reduce step values for each solution collected from all mappers, together makes each solution fully active.

Figure 10 shows the population on TaskTracker level and represents the mapper step where parts of solutions are inactive.

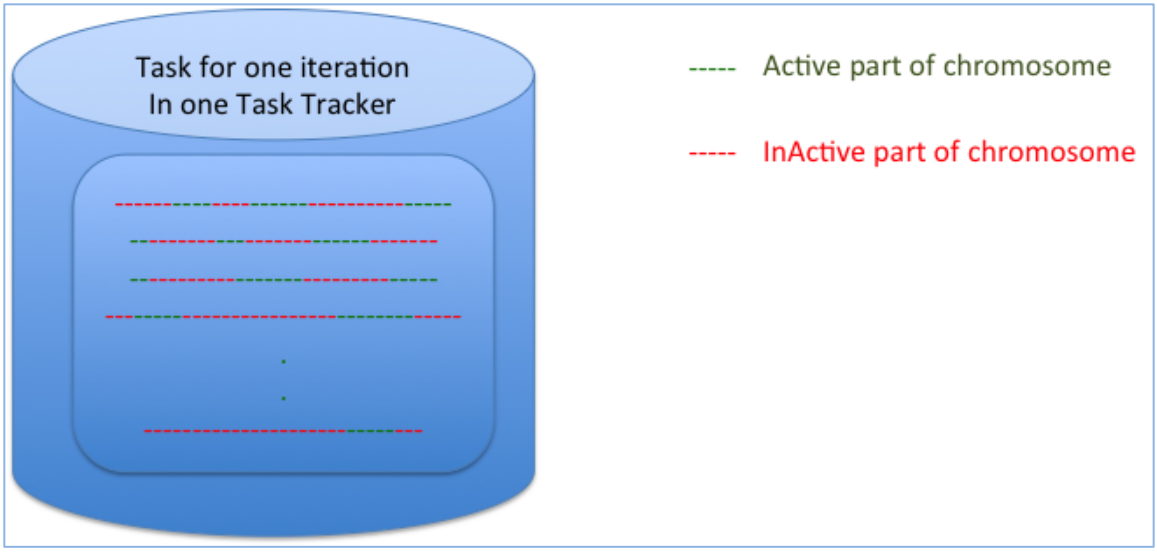


Figure 10: population on TaskTracker level at mapping step

The green dashed line is the active part of the solution, and its data is available in the data block on DataNode where the TaskTracker runs. The red dashed line indicates that these data are available in another data block on another DataNode. The reduce step combines all available solution for one or more Keys, and the main Key becomes all active because of the data shuffle process that collect all data for each solution. After the reduce step is carried out, the result for each solution (NewKey) is a data structure we developed, and it is shown in figure 11. It's content is the list of solutions with its distinct groups and final fitness ready to be written on HDFS, so that HDFS can read it and proceed to the next generation in the algorithm.

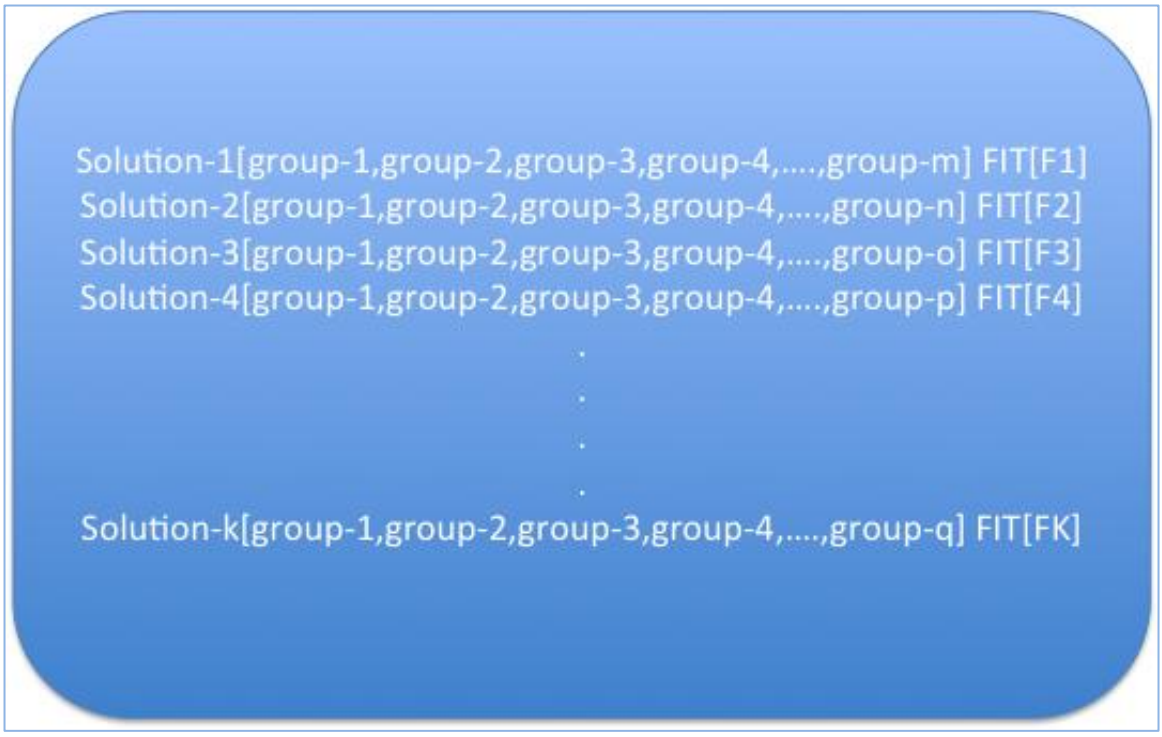


Figure 11: Data file written on HDFS after reduce operation

Each solution after this process have different number of groups based on the solution itself, and the fitness FIT is calculated by the previous mentioned formula (Fc). This data is returned to the HDFS client, and as mentioned before groups are parted for best solution written on most recent result file where the value of FIT returns to the algorithm so it can proceed to the next generation.

General steps of map reduce need definitions of writables, an object that can be written into a file. Since a Key and Value of our object is being used, general writable didn't fit our needs, so new definitions for writable was developed so it can be used as custom objects as Keys and Values. Some operations like equality and relational algebra operations are overridden to allow the TaskTrackers to shuffle Keys and Values between them.

5. Synthetic MapReduce Job Illustration

In this section, a MapReduce job is illustrated on the proposed framework to explain exactly what is happening in each step of the Job on TaskTrackers. Figure 12 (A) shows a synthetic dataset as edges list file and (B) its graph representation before uploading on HDFS. Where N1 is node at one side of the edge, N2 the other node at the other side of the edge.

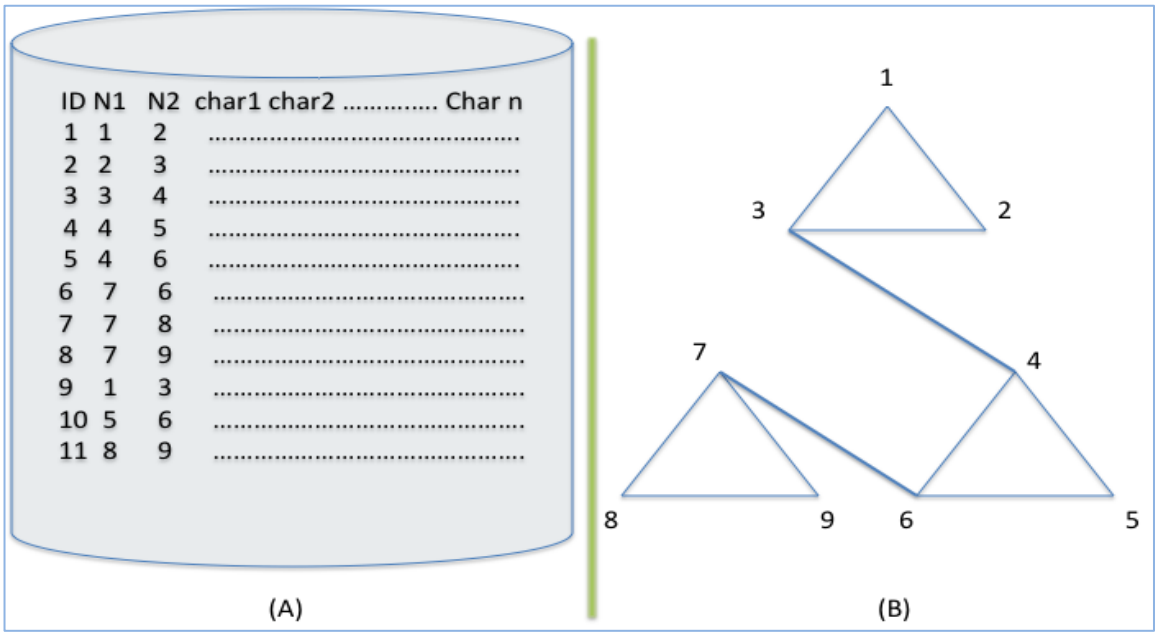


Figure 12: (A) synthetic edge list, (B) Graph representation of list (A)

After uploading the file to HDFS cluster, it gets divided into data blocks each on DataNode, multiple blocks can be on the same DataNode. For illustration purposes, the file is divided into three data blocks, each on separate DataNode as shown in figure 13.

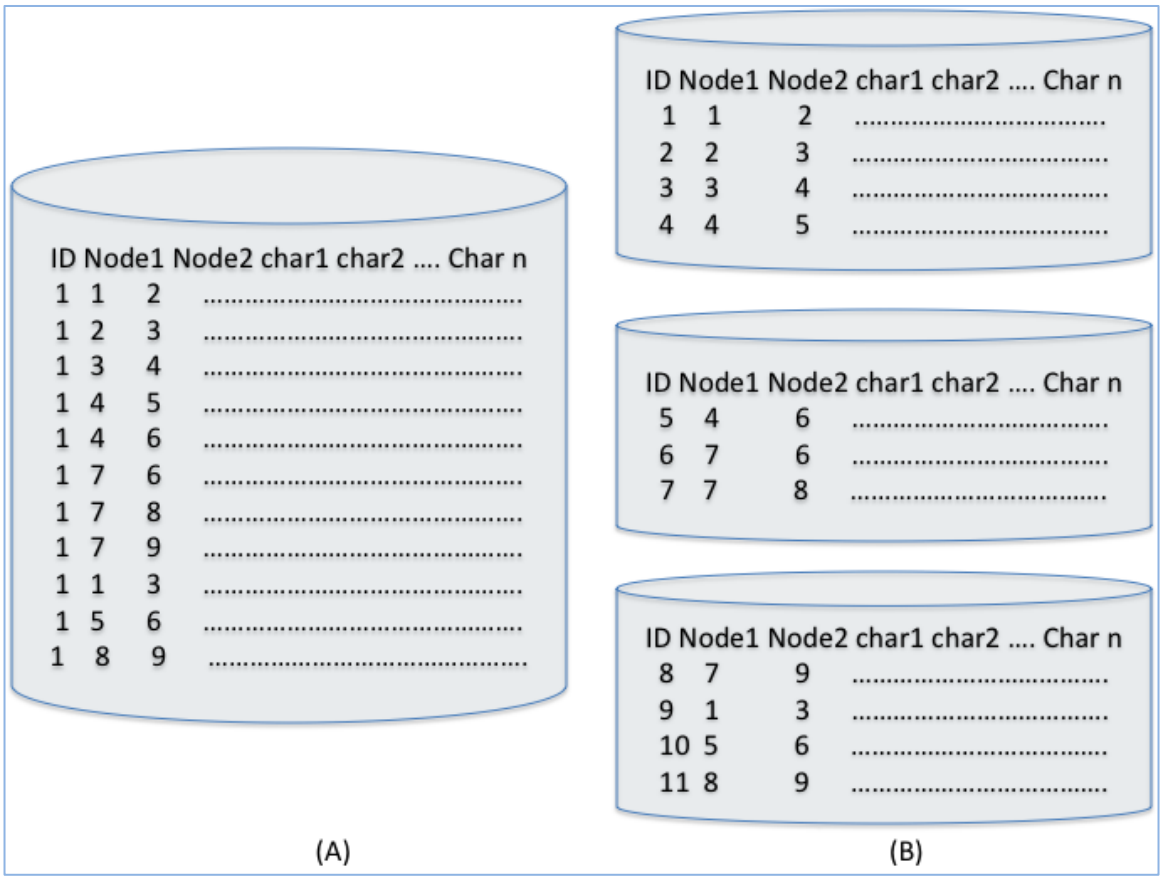


Figure 13: (A) File before uploading to HDFS, (B) Data blocks after uploading to HDFS

Each TaskTracker receives a copy of the Keys-solutions- to the mapper as mentioned in section 4.4, and maps the data into a <Key, Value> pairs for the active part of the solutions. These <Key, Value> pairs are written into intermediate files using the custom writables we have specially designed. Figure 14 shows the map task on a single TaskTracker on one DataNode.

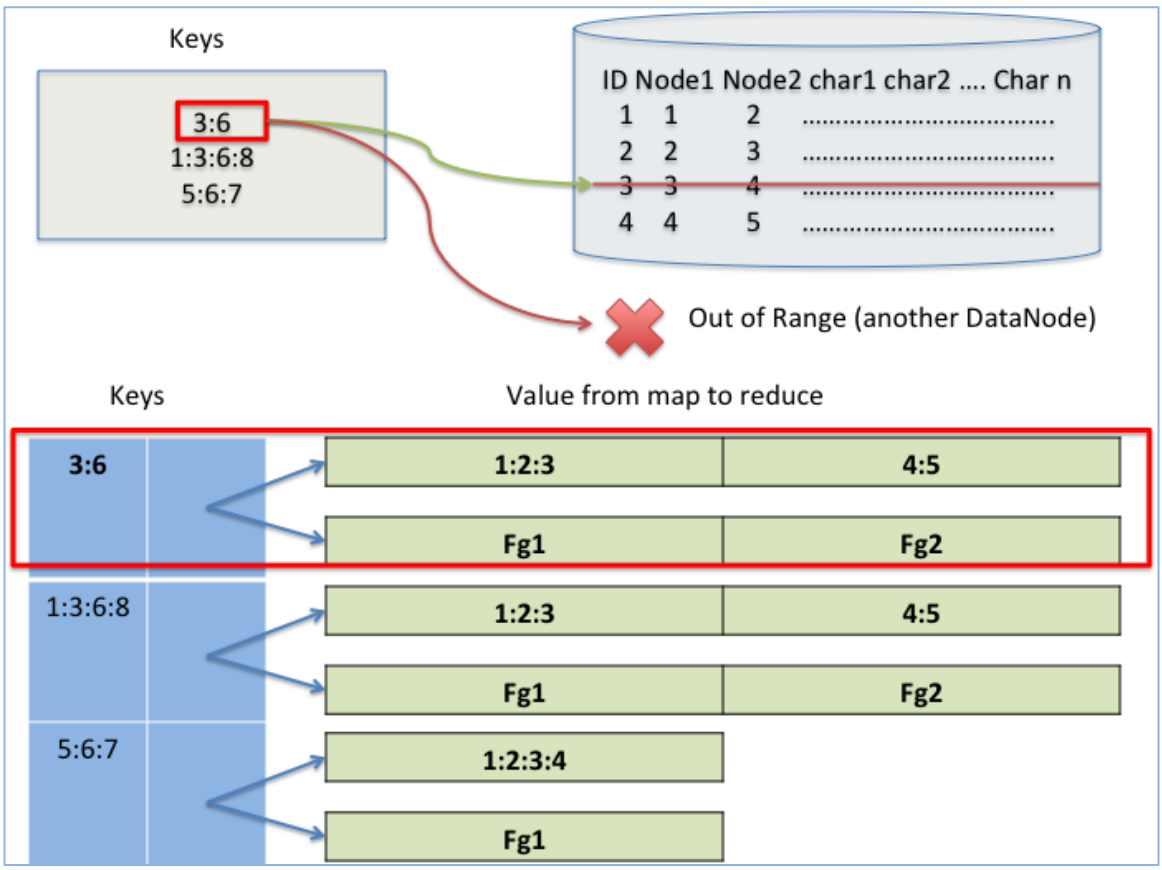


Figure 14: map operation on single DataNode.

Figure 14 shows a data block, which contains edges with IDs 1,2,3 and 4, and the Keys list, which is input to the mapper and consists of three solutions. The first one is responsible for removing noise edges 3 and 6, here edge 3 is considered active part since the data blocks on this DataNode (only one block available) and contains information about it; on the other hand, 6 is inactive since no information about it is in this DataNode. The mapper maps these values after removing the edge 3 for the first solution and writes the values shown in figure 13 as an array list. In figure 14, nodes 1,2,3 and 4,5 are grouped after their fitness is calculated. Other nodes will be combined from other mappers in the reduce phase, and groups can be merged together when there is connections and fitness is recalculated. The same process is continued for solution 2 and 3. Their keys and values list from this mapper is then sent to the reducer.

The reducers shuffle the files so that each Key gets assigned to one reducer along with collection of values as an array, making the whole key active at this stage. The reducer looks for connections between groups, combines groups where there is connection into a single group, then combines multiple array elements into one element and recalculates the fitness by combining subgroups fitness values in the same formula. Figure 15 (A) shows the result received by single reducer for the first solution, and (B) shows the solution after reducing and merging groups that have connections.

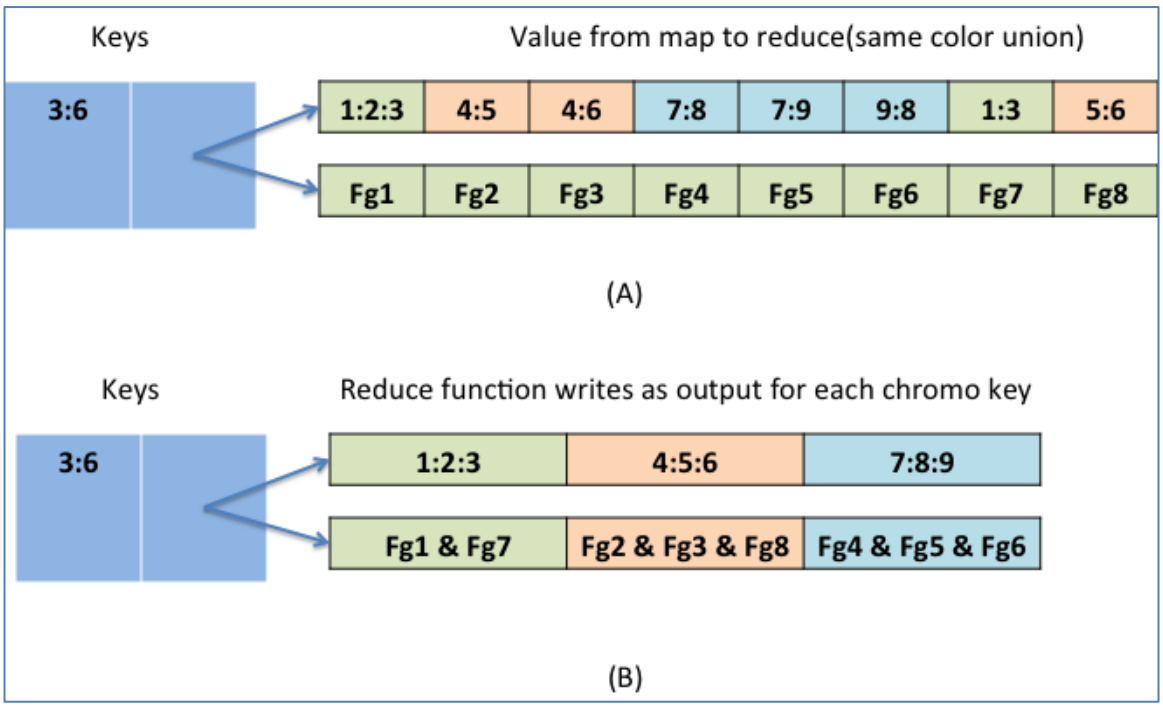


Figure 15: (A) Values list for first solution from mapper collected to reducer, (B) groups merged after reduce process.

In figure 15 (A) each color should be reduced into one group. The reducer merges these groups using the hybrid HashMap we developed and explained in section1.3.3. The same section using the following algorithm is illustrated in table 2 and figure 3:

1. Convert the Keys and Values list to hybrid HashMap.
2. While (HashMap is not empty)
    a. Pop the first element from the map and push into stack
    b. While stack is not empty
        i. Pop node from the stack.
        ii. Add the fitness to the group fitness using the formula in section4.3. and the node to result group

iii. Push all nodes connected to the stack.

iv. Remove the node from the HashMap and the reverse of the edge from connected nodes to it.

v. End while

c. Write the group as finalized to the result writable

d. End while

3. Write the result writable to reduce output file on HDFS

After implementing this algorithm to all reducers, all keys are finalized with values and written into HDFS ready to be downloaded to the HDFS client for most recent results files, values for the algorithm to generate the next generation (population cycle).

6. Experiments and Analysis of Results

6.1 Synthetic Dataset Experiments

Synthetic dataset are used as a starting point for experiment. Table 3 below represents a dataset for small world synthetic dataset, and it consists of one characteristic (number of messages between nodes), 9 nodes and 11 edges. Figure 16 represents a graphical representation of the same network.

Table 3: Small world synthetic dataset

Node1	Node2	Number of messages
1	2	5
1	3	6
3	2	2
3	4	1
4	5	10
4	6	5
5	6	3
6	7	3
8	7	6
7	9	3
8	9	2

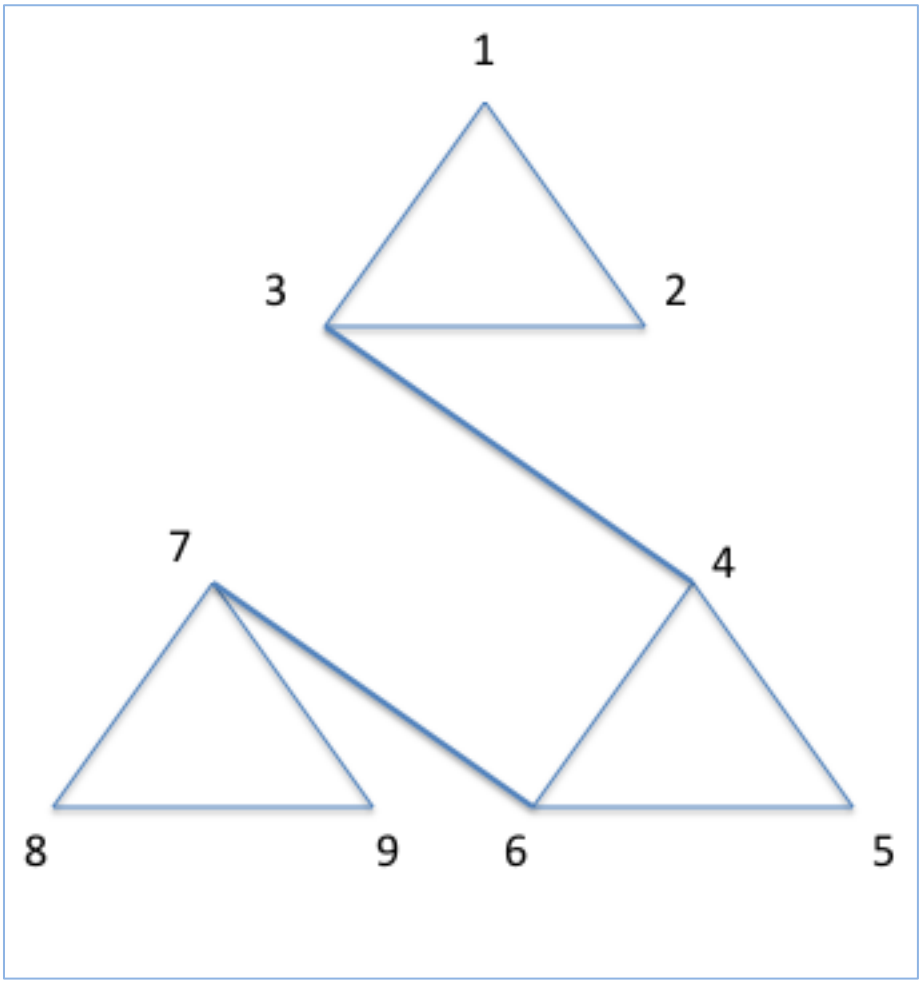


Figure 16: Graphical representation of small world synthetic dataset

The Algorithm was executed for the dataset above with a population size of 20. After the fourth MapReduce operation, the desired results provided by the system are kept steady. The result are as follows: three groups were generated after removing the noise edges 3-4 and 6-7, resulting the groups1 [1,2,3], group2 [4,5,6] and group [7,8,9], with the highest fitness were achieved. The results were

compared manually by generating all possible grouping solutions, the result were totally optimal as expected since the dataset was very small.

To verify the reaction of the algorithm with changes in the network, a new couple of nodes were added, node X and node Y. Node X was added with relations to nodes 2 and 3, where node Y added with only one relation to X, as the result immediately reflected in the network to group1 during the fifth generation, since the edges were not noise edges, then a new relations from y to nodes 4,6 and 5, after 2 generations node Y was removed from the first group and added to the second group. The last experiment was to join multiple groups together by adding new relations from node Y to nodes 2 and 3, again after two generations group1 and group2 were joined together in one single group. After all of these changes the network became 2 groups, group1 [1,2,3,4,5,6,X,Y] and group2 [7,8,9].

After the first experiment verified, the results showed that the algorithm successfully passed the small tests on small world dataset.

6.2 Large Scale Real World Dataset

The next experiments took place on a larger scale, a youtube multi-dimensional dataset available online pulled from youtube servers using an open source API called youtube API. The dataset contains 15,088 nodes and 5,574,249 edges, it also contained the following characteristics:

- Number of shared subscribers between two users.
- Number of shared favorite videos

- Number of shared friends between two users excluding the original nodes.
- Number of shared subscriptions between two users.

These datasets were merged together in on dataset and uploaded into HDFS of 4 nodes, three DataNode and one NameNode, and then the following experiments were performed.

The first experiment consisted of 100 edge, population size 100 chromosomes, each generation execution time were around 6000 ms, after ~50 generations the solutions start to take a steady groups, 17 groups were found. During the run of the algorithm, dataset was modified, 5 arbitrary groups were added totally not connected to any of previous groups, and the results were immediately reflected. After 3 groups were joined together with some noise edges, the algorithm took around 10 generations to find those edges and to separate the groups again and go back to the steady results which was expected.

The same experiment was done with larger datasets, 200, 400, 800, 1600, 3000 and 10000 edges; table 4 shows the results of these experiments.

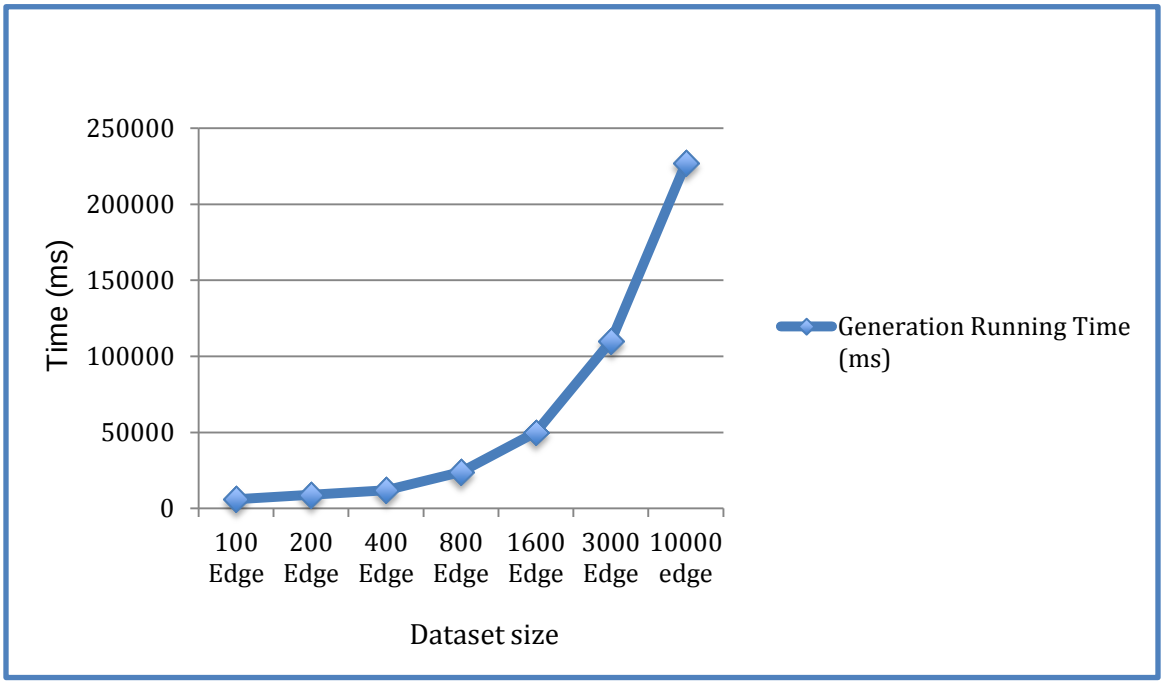
Table 4: experimental Results (2 Nodes HDFS Cluster)

Dataset Size	Average Generation execution time (ms)	Number of groups	Number of generations for steady results
100	~6000	17	~50
200	~9000	26	~130
400	~12000	50	~280
800	~24000	90	~740
1600	~50000	236	~2000
3000	~110000	479	~3200
10000	~270000	4932	~7100

6.3 Real World Dataset Experiments Analysis

As results are shown in table 4, there was almost a polynomial relation between the dataset size and the generation execution time. The time difference caused, because of the dataset size and the communication latency, more time needed for shuffling data between the cluster components extra than the clustering algorithm time.

Graph 1 illustrates the average generation running time vs. dataset size on the same HDFS cluster.

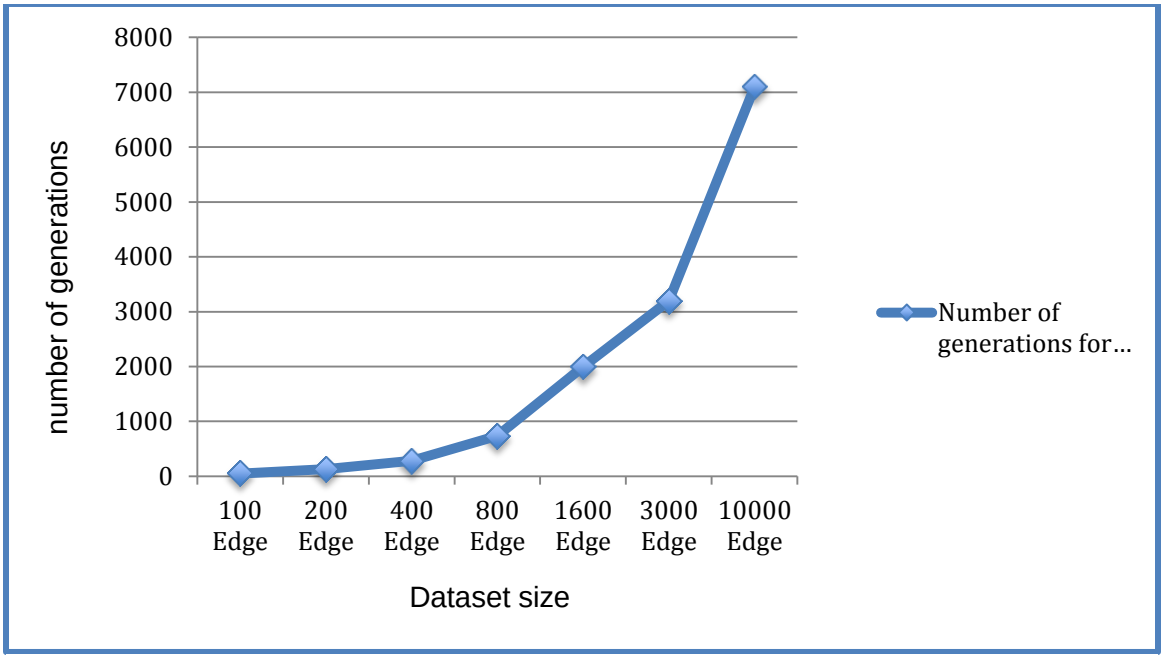


Graph 1: Average generation running time Vs. dataset size

The small curve at the end of the graph is caused by the difference between the last two datasets in size, and the light curve at the left side of the graph is caused by the communication latency at shuffle step between the maps and reduces. The algorithm doesn't affect the number of groups found. Numbers of groups are totally related to the dataset.

Groups strength –cluster coefficient – had no noticeable difference from previous evolutionary work such as Manish Gupta Charu, C. Aggarwal Jiawei, Han & Yizhou Sun, (MOEAs)[4], since same objectives were used. The main differences were at time and size variables; the approach performed faster and on a larger scale.

Another positive impact of the approach that it extract parameters like optimal number of groups and groups sizes, considering it as objectives where previous approaches consider these values as input parameters, and gave our approach an advantage of less number of runs to get the right inputs, which differ from dataset to another.



Graph 2: Number of generation needed for steady results vs. dataset size

The relation between the number of generation and the dataset size is clearly more expensive than polynomial; however, the result shows that it's less expensive than exponential.

A deeper study and analysis of results files produced by the framework showed that after making changes in the dataset, the changes immediately reflected on the dataset of the changes do not result in adding groups, or splitting groups. However if the changes add groups or split current groups into new groups it takes time only to cluster new changes, groups that are not affected do not need to be re-clustered, and that caused by the evolutionary part of the framework, since it keeps a copy of the best solutions and passes it to the next generation.

6.4 HDFS Experiments

The HDFS components are tested on the same YouTube dataset of 10000 nodes. The number of DataNodes involved in the HDFS cluster are changed and the test was run on single node cluster, 2 Nodes HDFS cluster, 3nodes HDFS cluster and 4 nodes HDFS cluster.

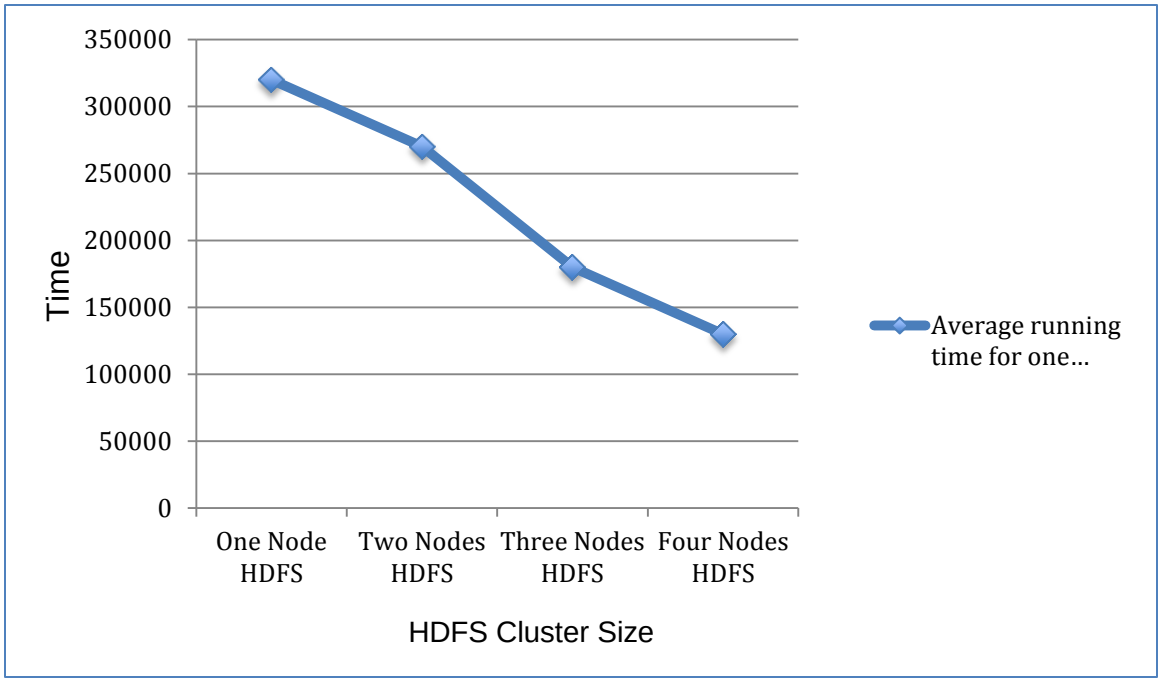
Table 5 illustrates the results on multiple clusters for 10000 edge dataset and 100 solution population size.

Table 5: running time on different HDFS cluster size

HDFS cluster size	**Average Generation running time (ms)**	**Number of generations for steady results**
Stand alone one node	~320000	~7100
Two nodes HDFS cluster	~270000	~7110
Three nodes HDFS cluster	~180000	~7120
Four nodes HDFS cluster	~130000	~7110

Table 5 shows that the HDFS cluster size had negligible effect on the number of generations needed for steady results to start being produced, thus the size of HDFS cluster affected only the running time and had no affect on the results of clustering the dataset.

Graph 3 illustrates the relation between the size of the HDFS cluster vs. the average running time for each generation.

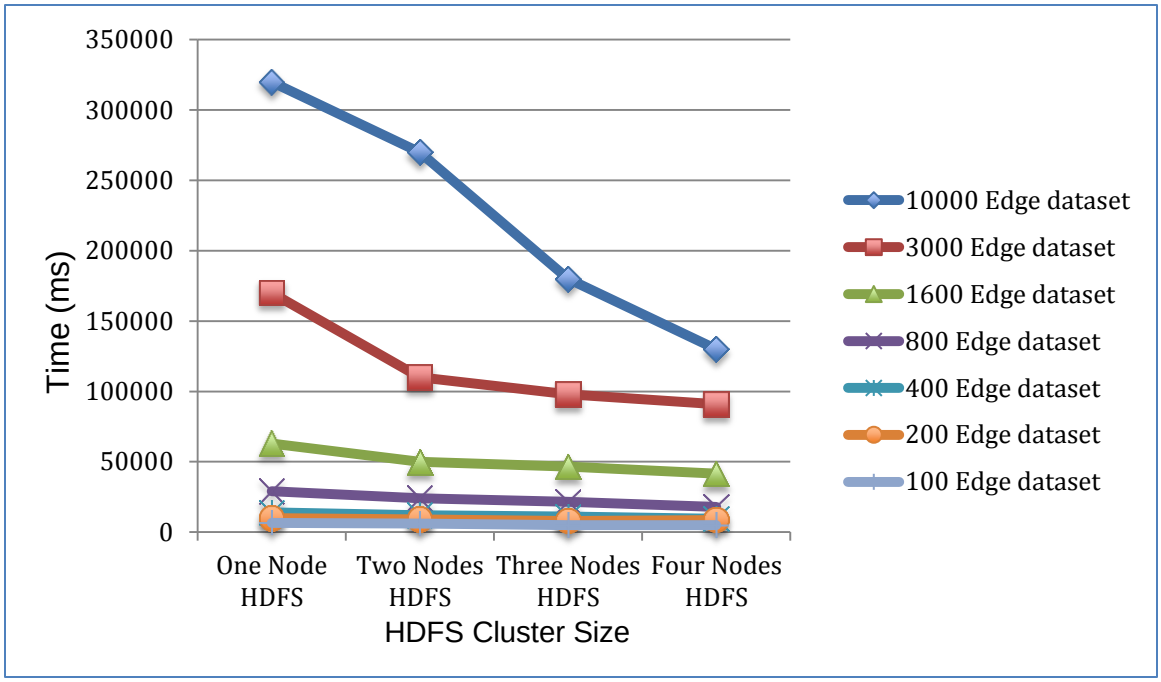


Graph 3: Average generation running time vs HDFS cluster size

The average running time do not decrease in polynomial form since increasing number of nodes in HDFS cluster reduce clustering work;on the other hand, the communication and shuffling latency time increases and communication between nodes takes more time.

6.5 Literature Comparative Results

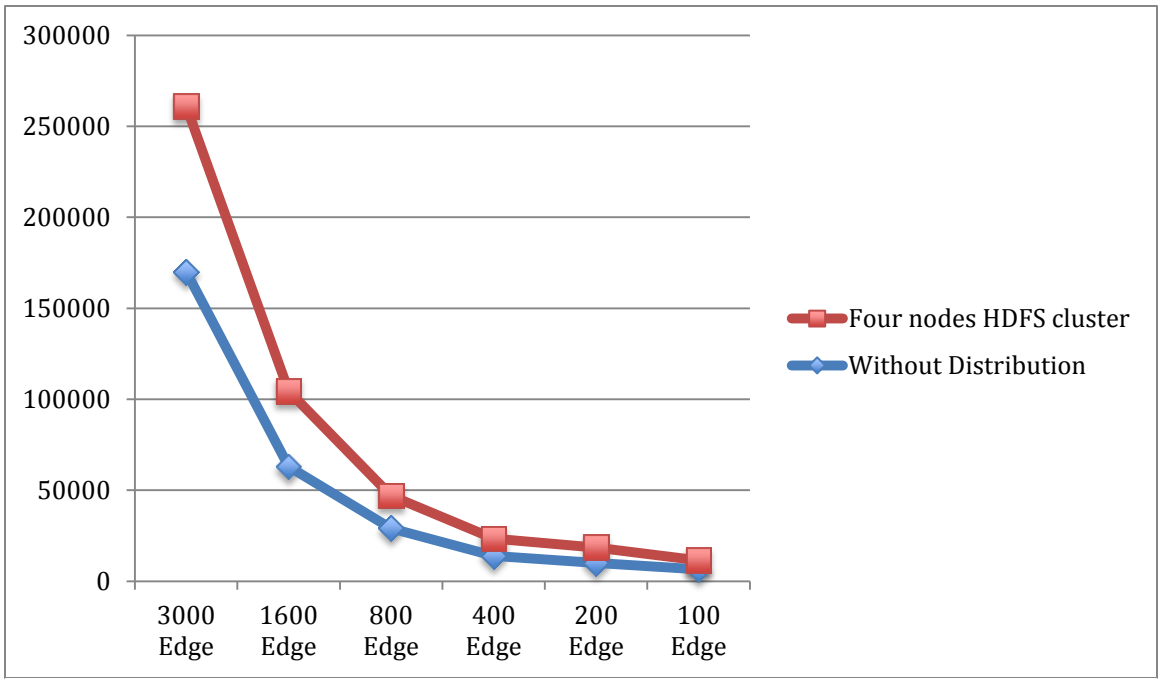
For comparison between average running time on different sizes HDFS cluster and different dataset size, graph 4 illustrates the performance of HDFS cluster and its affect on clustering time.



Graph 4: average generation running time vs. different dataset size on different HDFS cluster size

Graph 4 explains the affect of the HDFS size on clustering performance on time variable. Results on one node HDFS cluster illustrate the execution of the evolutionary algorithm without distribution. By comparing results on graph 4, the results shows that HDFS cluster size have a big affect on the running time, and this effect decreases with the dataset size, on a very small dataset the HDFS size starts to lose it affect, since communication time between HDFS components and the shuffling latency takes more time than clustering on one node HDFS cluster. The almost steady change in average running time for 100,200 and 400 datasets size clearly proves the analysis above.

Graph 5 illustrates the time performance to cluster 3000-edge network, to compare the difference of executing evolutionary clustering algorithm without distribution, with executing the algorithm on HDFS cluster of 4 nodes.



Graph 5: Single algorithm vs distributed

Graph 5 illustrates the difference for evolutionary clustering without HDFS distribution, in comparison with four node HDFS distribution. Results show that there is a slight difference on small datasets; however, the distribution of the algorithm execution provided a noticeable difference for larger datasets. Distributed evolutionary clustering algorithm improved the performance by influencing the computation performance on time variable. The same execution steps with almost half the time. However on small datasets, results show that there is slight difference or almost no difference in execution time. Deep analysis

showed that communication and shuffling step on four Nodes HDFS cluster consumes almost the same time difference the distribution saves in map and reduces steps.

7. CONCLUDING REMARKS and Future Work

Clustering social network and huge datasets is an NP-complete problem, optimization techniques proved efficiency in solving such problems in the past, however combining such algorithms with the new distributed systems, lead to noticeable improvement.

Defining multi-Characteristics social network dataset as a collection of multiple layers graphs, where each have the same set of nodes but different set of edges based on the characteristics, improves the computation and makes less overhead in computation aspects. On the other hand, considering the social network graph as a single layer represents all links and characteristics, leads to more complex graph with multiple links between same end nodes thus resulting in higher complexity for the same algorithm.

Social networks are full of noise, and can lead to improper assumptions. To be able to deal with large datasets, noise has to be eliminated.

Distributed systems do not affect the solutions as much as the primary algorithm does; however, it has a big influence on the performance of the algorithm used, speeds up the process, and reduce work load and memory usage. Distributed computing opens new directions for algorithms to be expanded, and distributed file systems allow computing power to be able to work on a larger scale.

This work showed the importance and the power of combining literature studies with new technologies and opens new areas of concentration where fast and optimized results are needed.

At the end distributed computing should attract more attention, since it allows algorithms to run on a collection of normal resources rather than acquiring huge expansive resources.

As a next step we are looking for more inelegant clustering solutions that provide more adaptability with the dynamic changes in the social networks, and provide the ability of predicting future changes on clusters based on the heuristics of previous clustering solutions for the same social network.

REFERENCES

[1]. A List of NP-complete problems, Wikipedia, http://en.wikipedia.org/wiki/List_of_NP-complete_problems

[2]. A. Sima Uyar & Sule Gunduz Oguducu. A new graph-based evolutionary approach to sequence clustering. In ICMLA ’05: Proceedings of the Fourth International Conference on Machine Learning and Applications, pages 273–278. IEEE, 2005.

[3]. A. Sima Uyar & Sule Gunduz Oguducu. A new graph-based evolutionary approach to sequence clustering. In ICMLA ’05: Proceedings of the Fourth International Conference on Machine Learning and Applications, pages 273–278. IEEE, 2005.

[4]. Antonio J. Nebro & Juan J. Durillo, jMetal 4.3 User Manual, January 3, 2013

[5]. B. Saha & P. Mitra. Dynamic algorithm for graph clustering using minimum cut tree. In Proceedings of ICDM Workshops, pages 667–671, 2006.

[6]. Babak Amiri, Liaquat Hossain & John Crawford, Multiobjective Hybrid Evolutionary Algorithm for Clustering in Social Networks,

[7]. C. A. C. Coello, G. B. Lamont & D. A. V. Veldhuizen. Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation). Springer-Verlag, 2006.

[8]. C.-K. Cheng & Y.-C. Wei. An improved two-way partitioning algorithm with stable performance [VLSI]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 10(12): 1502–1511, 1991.

[9]. Clara Pizzuti. Ga-net: A genetic algorithm for community detection in social networks. In PPSN, volume 5199 of Lecture Notes in Computer Science, pages 1081–1090. Springer, 2008.

[10]. Community Detection and Mining in Social Media. Lei Tang and Huan Liu, Morgan & Claypool, September 2010.

[11]. E. Zitzler & L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, pages 292–304, 1998.

[12]. Jianbo Shi & J. Malik. Normalized cuts and image segmentation. IEEE Trans. on Pattern Analysis and Machine Intelligence, 22(8):888–905, 2000.

[13]. Keehyung Kim, RI (Bob) McKay& Byung-Ro Moon, Multiobjective Evolutionary Algorithms for Dynamic Social Network Clustering

[14]. M E Newman & M Girvan. Finding and evaluating community structure in networks. Phys Rev E Stat Nonlin Soft Matter Phys, 69(2):026113.1–15, 2004.

[15]. Manish Gupta, Charu C. Aggarwal, Jiawei Han & Yizhou Sun, Evolutionary Clustering and Analysis of Bibliographic Networks,

[16]. Mursel Tasgin & Haluk Bingol. Community detection in complex networks using genetic algorithm. In ECCS ’06: Proc. of the European Conference on Complex Systems, Apr 2006.

[17]. Mursel Tasgin & Haluk Bingol. Community detection in complex networks using genetic algorithm. In ECCS ’06: Proc. of the European Conference on Complex Systems, Apr 2006.

[18]. Pasi Fränti & Olli Virmajoki, POLYNOMIAL TIME CLUSTERING ALGORITHMS DERIVED FROM BRANCH-AND-BOUND TECHNIQUE, University of Joensuu, Department of Computer Science, P.O. Box 111, FIN-80101 Joensuu, FINLAND

[19]. Peter Rousseeuw. Silhouettes: a graphical aid to the interpretation & validation of cluster analysis. J. Comput. Appl. Math, 20(1):53–65, 1987.

[20]. Petra Kudová, Clustering Genetic Algorithm, Department of Theoretical Computer Science, Institute of Computer Science, Academy of Sciences of the Czech Republic, ETID 2007

[21]. R. Breiger, S. Boorman, & P. Arabie. An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling* 1. Journal of Mathematical Psychology, 12(3):328–383, 1975.

[22]. T. N. Bui & B. R. Moon. Genetic algorithm and graph partitioning. IEEE Transactions on Computers, 45(7): 841–855, 1996.

[23]. T. Y. Berger-Wolf & J. Saia. A framework for analysis of dynamic social networks. In KDD ’06, Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 523–528, 2006.

[24]. Tsai-Yang Jea, Basic concepts of Data Mining, Clustering and Genetic Algorithms, Department of Computer Science and Engineering, SUNY at Buffalo.

[25]. X. Cheng, C. Dale, & J. Liu. Statistics and social network of YouTube videos. In IWQoS ’08: Proceedings of the 16th International Workshop on Quality of Service, pages 229–238, 2008.