Distributed Genetic Algorithm to Big Data Clustering
A Novel Distributed Encoding Techniques

Mustafa H. Hajeer
Department of computer science
The University of Memphis
mhhajeer@memphis.edu

Dipankar Dasgupta
Department of computer science
The University of Memphis
ddasgupt@memphis.edu

Abstract - Clustering algorithms have emerged as a powerful learning tool to accurately analyze the massive amount of data generated by current applications and smart technologies. Precisely, their main objective is to categorize data into clusters such that objects are grouped in the same cluster when they are similar according to specific metrics. There is a wide and diverse body of knowledge in the area of clustering and there has been attempts apply these algorithms and scale it to adopt today’s data. However, one major challenge in using clustering algorithms is scalability of such algorithms in a way that faces the challenges and computational cost of clustering big data. In this paper, we are describing a mapping between graph clustering problem and data clustering. Using genetic algorithms and multi-objective optimization as well as distributed graph stores, the proposed algorithm (1) transform big data into Distributed RDF graphs. With (2) a novel distributed encoding techniques. The algorithm (3) scales to deal with big RDF graphs to (4) produce clusters in using clustering algorithms is scalability of such algorithms in a way that faces the challenges and computational cost of clustering big area of clustering and there has been attempts apply these algorithms and scale it to adopt today’s data. However, one major challenge provided such data and (5) produce comparative results compared to other peers of clustering algorithms by maximizing graph modularity as a main objective. The results on LUBM generated big data shows the (5) ability to deal with the challenges of different genes “Biology” … etc.

As much as it is beneficial to build a science out of data many challenges appear while dealing with today’s data. Known as the five V's, Velocity, Variety, Veracity, Value and Volume. In other words, today’s hardware isn’t suitable enough to unlock the full potential of today’s data, and it has never been. The growth of data has a higher rate than the growth in possible computing power at a time.

One of the most widely investigated areas in data mining and machine learning communities is data clustering “unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters)”. The importance of such problem comes along with the importance of its applications in many fields such as; social network analysis, search results grouping, market research, recommendation systems, field robotics…etc. the applications can even be on a lower level in computer science field such as job scheduling, resource optimization …etc. Though; in most fields there is a limitation of applying clustering algorithms on their big data.

A very important instruments used to represent data in various domains are graphs. Graphs seem such a natural way for us to represent so many of the complicated biological, economic, social and technological systems we find in the real world. Graph clustering is the partitioning of a network into groups of nodes, called communities or clusters or modules, having compact intra-connections, and dispersed inter-connections. This is done by analyzing those graphs. Discovering this structure of corresponding graphs is a major task in the previously mentioned fields. In recent years, the scale of these graphs has increased to millions of vertices and billions of edges, making this discovery increasingly difficult and costly.

The analogous problem of data clustering in graph theory is graph clustering. Graph clustering problem differs from data clustering; since data clustering clusters are groups of points with respect to a distance or similarity measure, however; in graphs clusters are based on edge density. The definition of community in a network, yet, is not accurately defined, and the application domain of concern influences its definition. Thus, we can infer that the number of edges inside the same community in the graph (Intra-cluster edges) should be much higher than the number of edges connecting it to the remaining nodes outside the community (inter-cluster edges). Two main objectives can be derived from this definition: maximizing the number of community-internal links and, simultaneously, minimizing the number of external links that act as bridges between the communities. However, as mentioned before scaling such problems to adapt with the five V’s of Big Data is difficult and costly.

Since graph theories are well studied, experiments of mapping the two problems worth considering. By mapping data points and their properties into nodes with edges connecting them; data clustering can be transformed into graph clustering problem.

Modularity [1] is one of the most popularly used metrics for concluding the quality of non-overlapping graph clustering, particularly in the network analysis community [2], [3], [4], [5]. The problem of discovering a clustering with maximal modularity is NP-Complete [6]. As a result, much polynomial time heuristic algorithms have been developed [7], [8] [9] [10].

In this study we emphasis on the difficulties of applying clustering methods to big Graph data due to new challenges that are raised by big data. As Big Data is referring to millions and billions of data points and clustering algorithms are come with
high computational costs, it is extremely important to study how to cope with this problem and how to develop and extend clustering techniques to (1) be able to operate on big data and (2) get the results in a reasonable time.

In the next sections we discuss some definitions and notations, describe modularity and the relation to graph-based clustering and multi-objective paradigm. Then we introduce our distributed genetic algorithm where we developed using open sources for multi-objective optimization and genetic algorithms Jmetal [11] and apache Jena Elephas [12] to store and manage RDF data. Later we validate it on well-known datasets and experiment it on big RDF graph generated by (LUBM) Lehigh University Benchmark.

II. DEFINITIONS & NOTATIONS

A. Community and Community Detection Definition

Real world 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 help in areas such as social behavior, online marketing, and studies about web characteristics. Recently they have attracted much attention among the research field and research groups also in areas concerned with security issues. Understanding how these groups are formed and how they change over the time can help in applying theories and techniques to improve these fields. Networks, such as social networks, are a combination of interconnected distinct groups. These distinct groups, need to be extracted from the single large set of profiles and their connections, as modeled in the site ontology of a particular social media site. The study of inferring these groups is called graph clustering; which can show the real clusters (groups) within any dataset, such as social network data.

B. Network Clustering

In this work we referred to graph of vertices \( V \) and edges \( E \) as \( G(V,E) \), as an undirected graph. Let number of vertices \(|V|=m\), number of edges \(|E|=n\) and clustering \( C = \{ C_1, C_2, C_3, \ldots, C_j \} \) as a partition of \( V \) as 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 \( C_1 \) as a subgraph of \( G \). The graph \( G[C_1] := (C_1, E(C_1)) \), where \( E(C_1) = \{(V,W) \in E : V,W \in C_1\} \). Then \( E(C) = \bigcup_{k=1}^{m} E(C_k) \) is the set of intra-cluster edges and \( E(C) \) 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.

C. RDF Graph Stores

A triple store or RDF store is a graph based database for the storage and retrieval of triples by semantic queries. A triple is a data entity composed of subject-predicate-object.

Very much like a relational database, triple stores save information as triples and retrieve it via a query language; yet, there are some key differences, mainly that a triple store is optimized for the storage and retrieval of triples. In addition to queries, triples can be imported/exported using Resource Description Framework (RDF) and other formats.

Rohloff et al. [13] explained how to store graph data in Hadoop using a representation of vertex-edge-vertex format for what is referred to as triples, as illustrated in FIGURE 1. There has been some progress in research made towards clustered RDF database systems. Clustered RDF database that are currently available, such as SHARD [13], YARS2 [14], Jena and Jena Elephas [12] and Virtuoso [15], generally hash partition triples across multiple machines and parallelize access to these machines at query time.

II. DEFINITIONS & NOTATIONS

A. Community and Community Detection Definition

Real world 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 help in areas such as social behavior, online marketing, and studies about web characteristics. Recently they have attracted much attention among the research field and research groups also in areas concerned with security issues. Understanding how these groups are formed and how they change over the time can help in applying theories and techniques to improve these fields. Networks, such as social networks, are a combination of interconnected distinct groups. These distinct groups, need to be extracted from the single large set of profiles and their connections, as modeled in the site ontology of a particular social media site. The study of inferring these groups is called graph clustering; which can show the real clusters (groups) within any dataset, such as social network data.

B. Network Clustering

In this work we referred to graph of vertices \( V \) and edges \( E \) as \( G(V,E) \), as an undirected graph. Let number of vertices \(|V|=m\), number of edges \(|E|=n\) and clustering \( C = \{ C_1, C_2, C_3, \ldots, C_j \} \) as a partition of \( V \) as 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 \( C_1 \) as a subgraph of \( G \). The graph \( G[C_1] := (C_1, E(C_1)) \), where \( E(C_1) = \{(V,W) \in E : V,W \in C_1\} \). Then \( E(C) = \bigcup_{k=1}^{m} E(C_k) \) is the set of intra-cluster edges and \( E(C) \) 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.

C. RDF Graph Stores

A triple store or RDF store is a graph based database for the storage and retrieval of triples by semantic queries. A triple is a data entity composed of subject-predicate-object.

Very much like a relational database, triple stores save information as triples and retrieve it via a query language; yet, there are some key differences, mainly that a triple store is optimized for the storage and retrieval of triples. In addition to queries, triples can be imported/exported using Resource Description Framework (RDF) and other formats.

Rohloff et al. [13] explained how to store graph data in Hadoop using a representation of vertex-edge-vertex format for what is referred to as triples, as illustrated in FIGURE 1. There has been some progress in research made towards clustered RDF database systems. Clustered RDF database that are currently available, such as SHARD [13], YARS2 [14], Jena and Jena Elephas [12] and Virtuoso [15], generally hash partition triples across multiple machines and parallelize access to these machines at query time.

III. MODULARITY BASED GRAPH CLUSTERING & MULTI-OBJECTIVE PARADIGM

Studying complex networks usually involves detection of community structure [16], and sometimes it is referred to as clustering [17]. Graph clustering problem differ from data clustering, since data clustering clusters are groups of points close with respect to a distance or similarity measure clusters, while in graphs clusters are based on edge density. The definition of community in a network, however, is not accurately defined and the application domain of interest influences its definition. Thus, we can infer that the number of edges inside the same community in the graph (Intra-cluster edges) should be much higher than the number of edges connecting it to the remaining nodes outside the community (inter-cluster edges). Two main objectives can be derived from this definition: maximizing the number of community-internal links and, simultaneously, minimizing the number of external links that act as bridges between the communities.

A very interesting problem-solving method called multi-objective optimization that successfully determines a set of solutions where multiple or conflicting objectives must be optimized [18].

From these definitions of communities, multi-objective optimization and Evolutionary Algorithms; community detection problem can be formulated as a multi-objective optimization problem. The framework of Pareto optimality provides a set of solutions resembling the best compromise among the objectives to optimize.

As per Yi et al. [19] “In Genetic Algorithms (GA), a population of chromosomes, which encode candidate solutions/individuals to an optimization problem, evolves toward better solutions. After the solution is genetically
represented in the chromosome format and the fitness functions are defined, GA proceeds to initialize a population of solutions randomly/deterministically. Then, GA aims to improve it through repetitive applications of several genetic operators such as selection, crossover, and mutation. Finally, local search and boundary search operators are applied to fine-tune the results.

In the past few years, many algorithms were introduced to solve such a problem; some are evolutionary in nature, whereas others are not. Nonetheless, the main difficulty we encountered while applying such algorithms to large data sets was scalability. A large number of literature algorithms were replicated to produce results and some of them already had an open-source. Nevertheless, most of them failed to operate on larger datasets. 1.6M nodes and 52M edges were extracted from livejournal.com, and a server installation with 16 processor cores and a 192 GB main memory crashed several times when the software was used to analyze the data [20]. To overcome these issues a novel encoding for the solution space to save memory space was constructed. We also proposed a parallel processing and evaluation for these solutions, thus constructing a scalable framework that provides better quality and scalable community extraction approach than the known ones. This is illustrated in the results section.

One popular community detection algorithm is the Girvan-Newman algorithm [21], where edges having maximal betweenness centrality are consecutively removed from the network until no edges remain. Modularity may be defined as in equation (1), where $n_i$ is a total number of communities, $m$ is the number of edges in the graph, $l_i$ is the total number of edges within community $i$, $d_i$ is the sum of degrees of all nodes in $i$.

$$Q = \sum_{i=1}^{m} \left[ \frac{l_i}{m} - \left( \frac{d_i}{2m} \right)^2 \right]$$

As per S. Fortunato and M. Barthelemy [22] describe resolution limit in modularity-based community detection; on larger networks maximization of modularity does not always resolve smaller scale communities. Empirical comparison of different objective functions is presented in [16]. Survey of community detection methods for directed networks is presented in F. D. Malliaros and M. Vazirgiannis [23].

Application of genetic algorithms for community detection was described in [24], [2], [3], [4], [5], and [25]. M. Tasgin and H. Bingol [24] describes an approach where modularity was used as a fitness function. Chromosomes contains all nodes of the graph, and communities are assigned to them. GA-Net is another work of C. Pizzuti [2] were the Main difference from M. Tasgin and H. Bingol is the fitness function, in GA-Net community score is used. Also, different representation is used: locus-based adjacency representation, however it also stores whole graph in each chromosome. [3] Introduces MOGA-Net, multi-objective optimization algorithm for community detection. Key difference from GA-Net is the fitness function. Presented approach uses two functions, community score and community fitness.

Li and Song work [25] describes extended compact genetic algorithm. Results of modularity maximization and comparison with Girvan Newman, Clauset Newman Moore, and algorithm from most popular evolutionary clustering are presented in

<table>
<thead>
<tr>
<th>TABLE I. MODULARITY MAXIMIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>GN</td>
</tr>
<tr>
<td>Karate</td>
</tr>
<tr>
<td>Dolphins</td>
</tr>
<tr>
<td>Football</td>
</tr>
<tr>
<td>Books</td>
</tr>
</tbody>
</table>

Kajdanowicz, Kazienko and Indyk [26] describe architectures for parallel processing of large graphs. Authors compare MapReduce, map-side join, and Bulk Synchronous Parallel (BSP) by running two graph algorithms: single source shortest path, and relational influence propagation. BSP model was developed in 1990 [27]; recently it proved to be useful for distributed graph algorithms. There is proprietary implementation by Google, Pregel [28] and open-source implementation, Graph Processing System (GPS) [29]. There is also implementation of BSP model from Apache foundation, Giraph [30].

Label propagation community detection algorithm, which is not based on modularity is described in [31]. Clique percolation method is described in [32]. Distributed community detection method based on ensemble learning is described in [33]. High performance parallel community detection, and its implementation in C++ with OpenMP is described in [34].

Another distributed genetic algorithm to solve clustering problem were described in [35] and [36]. Hans et al. [35] formed chromosomes of centroids for each data split input to mappers randomly and merge the centroids in the reduce phase. The limitation in [35] was the need of previous knowledge about the number of clusters. On the other hand, [36] formed a very long chromosomes of each data point, memory needs and limitation of applying GA operators on big-data presented in such technique.

Clustering based on machine learning techniques were presented in [37] and [38].

IV. DISTRIBUTED GENETIC ALGORITHM

A. Encoding & Representation

Most of the evolutionary achievements in the literature tend to use the same encoding routines, where nodes in the graph are placed along with its cluster ID in each solution "chromosome" and adjusts only the objective functions. Thus constructing algorithms that work fast and provide good quality results on small datasets. However, when we tried to apply these algorithms on some larger scale datasets (millions of nodes or more), we encountered a problem of very long chromosomes (solutions). Where each solution contains all graph nodes in it, resulting in a problem applying the evolutionary operations on the solutions (i.e. crossover and mutation). Hence making it impossible to deal with huge datasets or very slow with medium size datasets.

We used encoding from [39] to overcome the big encoding issues found in previous studies and listed in [39]. Such
encoding derives from the definition of clusters. However, even with such encoding in [39], solutions can still have a very large representation as the data scales up. Eventually, the GA client will run out of memory handling solutions itself as the data scales up. Another technique we used to reduce the overhead of manipulating these solutions is to store it as extra information along with graph triples on HDFS. By converting data points from `<Node> <Predicate><Node>` triples as in Rohloff et al. [13] into `<Chromosome_part><Node>` `<predicate><Node>` quadruples. We referred to `<chromosome_part>` as a list of solution IDs that this particular node belongs to in the population. This encoding lead to a population of a fixed size list of Integers on the GA client side called solution IDs. This technique allows the client to scale the clustering GA on larger size datasets that the HDFS can hold.

The idea was to treat solutions as data and to inherit all scalability properties that apply to the graph. Thus, the population of a size X on the client side has a constant size(X) regardless of the data size. We referred to this novel technique as Distributed chromosomes, and as a concept, it is about the distribution of genes from the solutions along with the data. FIGURE 2. explains how graph data were stored in RDF format and how we performed the integration of solution encoding on RDF data.

We used Apache Jena and Jena Alephas and modified these open sources to match our needs. Convert_to_quads Chromo was developed to convert RDF graph Triples to Quads as in FIGURE 2. This class contained Mapper, reducer, combiner, and appropriate writable as well as input and output classes formatted to deal with RDF data. It takes each triple from each block of data and converts it into a quad with a random gene (part of solutions) that it belongs to then stores it back into HDFS.

![Figure 2](image)

**FIGURE 2. RDF TRIPLES TO CHROMOSOMES QUADRUPLES “SOLUTION ENCODING”.

B. **Objective Functions**

In our clustering algorithm, we are using modularity as a fitness measure in Hajeer et al. [39]. As per [40] Modularity is then defined as the fraction of edges that fall within group 1 or 2, minus the expected number of edges within groups 1 and 2 for a random graph with the same node degree distribution as the given network. Hence, the actual number of edges between v and w minus expected number of edges between them is

\[ A_{vw} - \frac{\langle k_v \cdot k_w \rangle}{2m} \]

Thus, modularity can be expressed in equation (2).

\[ Q = \frac{1}{2m} \sum_{vw} [A_{vw} - \frac{K_v \cdot K_w}{2m}] S_v S_w + \frac{1}{2} \]

It is important to notice that equation (2) partition the network only for two groups. To identify multiple communities in a graph this formula has to be generalized as equation (3):

\[ Q = \frac{1}{2m} \sum_{vw} A_{vw} - \frac{K_v \cdot K_w}{2m} \delta(C_v, C_w) = \sum_{i=1}^{c} (e_{ij} - a_i^2) \]

Where \( e_{ij} \) is the fraction of edges with one-end vertices in community i and the other in community j as in equation (4):

\[ e_{ij} = \frac{1}{2m} A_{vw} 1_{v \in i} 1_{w \in j} \]

and \( a_i \) is the fraction of ends of edges that are attached to vertices in community i as in equation 5:

\[ a_i = \frac{k_i}{2m} = \sum_j e_{ij} \]

Note that modularity maximization is not the only objective. Another objective is to minimize the solution length. Considering intra-cluster edges as inter-cluster edges results in some longer solutions with no difference in modularity. Hence, those solutions need to be given a smaller fitness but not totally ignored (a combination with other solution may lead to a better clustering).

V. **ALGORITHM DESCRIPTION**

A. **Population Initialization**

Population initialization is the process of creating a collection of diverse solutions. As described in encoding and representation section, we transform the triples in RDF data into quads, adding the ability to hold a gene “part of the solution” for each data point. Where this gene is a random solution IDs that each data point belongs to” if a data point D have S1 and S5 as genes, that is translated as the solutions S1 and S5 will consider the data point D edge as an inter-cluster edge that connect two separate clusters“.

Considering T is the set of triples represent the graph G and S is the set of solutions in the population; then \( V \neq T \) there is a set of solutions \( S_0 \subseteq S \). This set of solutions \( S_0 \) when combined represent the solution \( t_0 \). It is very important to keep in mind that the maximum size of \( S_0 \) is the integer size of the population. The initialization process for a populations is shown in FIGURE 2. Here we note again that the GA client will only hold a two dimensional array of integers “solution IDs” and floats “Modularity Fitness”, thus allow the client to start the selection process and initiate the distributed GA operators working on a fixed small size two dimensional array, where the real genes are stored in the data blocks in a distributed manner taking advantage of HDFS. Refer to FIGURE 2.

B. **Solutions Evaluation**

The evaluation was done using the objective functions described in the objectives section. Each solution is evaluated by computing modularity on the analogous graph, a graph where edges in the solution are marked as inter-cluster edges. We identified the clusters by removing the marked edges and considering the disconnected graph components as
communities. Then, we computed the modularity considering the marked edges again as inter-cluster edges.

The process of computing the modularity on a large graph is both resource and time consuming, so we decided to improve it using distributed tasks to be run on the quadruple store created with extra data for solutions. Using HDFS and distributing the dataset over multiple machines, we were able to batch process each set of solutions (generation) at once.

After the client side of the algorithm injects current population solutions data into the quadruples stored in HDFS, it sends the list of solutions IDs (list of integers) to be evaluated. FIGURE 3. illustrate the evaluation Map tasks.

```java
Given a Population_ID list S that contain ID’s of solutions to be evaluated
Map (Key index, Value Quad):
ForEach solutionID in S:
  If Quad.GetGenes in solution:
    Quad.marked = True
  Else:
    Quad.marked = False
  Emit (solutionID, Quad)
```

FIGURE 3. DISTRIBUTED EVALUATION MAP TASK.

The map function is called for each Quad in the graph chunk that represents part of the graph. Jena Elephas is used with modified input and output class to use Chromosome quads rather than default graph quads. Each container on the HDFS cluster performs map operation on the graph chunks it has assigned. After mapping all the chunks into pairs of <keys, values> representing solutions IDs and Quad that are part of the corresponding solution, the shuffling task takes place. All values for the same key are grouped together as <key, list of values> that represent each solution and the list of marked and unmarked Quads (Graph where inter-cluster edges are marked). The final stage consists of the reduce tasks that are described in FIGURE 4.

```java
Given a solution S and a set of Quads marked based on S, as mappers outputs and reducer input for a graph G with N Quads
Reduce (Solution S, EdgesQuads [E1,E2,E3,...,EN]):
  ForEach Quad E in QuadsList:
    If E.marked = True:
      MarkedQuads.append(E)
    Else:
      UnMarkedQuads.append(E)
  Endfor
  Communities = FindComponents(MarkedQuads, UnMarkedQuads)
  Modularity = 0
  ForEach Community C in Communities:
    DegreeFraction = (C.InnerEdges *2+ C.OutterEdges)/(2*N)
    Modularity += (C.InnerEdges /N)-(DegreeFraction)^2
  Endfor
  Emit (S, Modularity)
```

FIGURE 4. DISTRIBUTED EVALUATION REDUCE TASK.

The FindComponents function was implemented using a modified linear finding component algorithm to store also the number of intra-cluster edges and the number of inter-cluster edges for each community. When reduce tasks finishes, the results of reducers are written to HDFS, and it contain each solution with its modularity. The results consist of a fixed size two-dimensional array of integer solution IDs and a fitness for each solution. The evolutionary algorithm reads this file and continues working on an evaluated generation ready for selection, crossover and mutation processes. In the last generation, an extra piece of information controlled by a boolean configuration variable is written to HDFS as well; this piece contains the clustering affiliation for each node. The reason they are only written in the last generation is to lower the write overhead on HDFS while affiliations are not needed any time before it.

C. Crossover Mutation and Selection

Since we stored the chromosomes in a distributed manner, we needed to modify the GA operators used in Jmetal open source to be able to run them on the corresponding quadruples that represent the graph. This procedure was done by developing a distributed crossover and distributed mutation modules, which in return creates jobs of crossover and mutations to be performed on the corresponding population.

After evaluating the population, the selection process starts based on each solution ID and its fitness. Tournament selection is the selection used, and the reason is to avoid converging to local optimal solutions - which are a lot based on our encoding technique. By ranking the population and choosing solutions from each rank, a set of parents along with the new offspring IDs were constructed. FIGURE 5. is a diagram showing the steps in which the algorithm creates GA operator’s tasks.

Encoding and storing solutions in a distributed manner delivered the advantage of small and fixed size populations on a client side. However, GA operators in the open source Jmetal needed to be modified as well as NSGAI, which was used in our case. The original NSGAI creates a population’s and offspring’s solutions and evaluates it one solution at a time. Such a case creates an overhead of tasks on HDFS. Rather, we modified NSGAI to DNSGAI (Distributed NSGAI) by creating a set of solutions then performing evaluation and GA operators at once in one MR2 task. FIGURE 6. illustrates the task of distributed crossover and distributed mutation.

The distributed crossover and mutation task takes the population as inputs along with the selection results then, for each quad in the data, changes the partial chromosomes accordingly. The task
removes any solution ID (gene) that does not belong to the current population to save space and computations. Then, as shown in FIGURE 5, the new offspring population is sent to evaluation. Here we have to note that solutions that belong to a previous generation will not be evaluated since they already have fitnesses. This copy technique of fitnesses saved a huge amount of computations when we dealt with big data for a long series of generations.

**To save space and computations, we remove any Solution ID that is not in the current population from records.**

The processes of representation, population initialization, evaluation, selection and offspring evaluation to population are illustrated in FIGURE 7. The numbers represent the processes evaluation, selection and offspring evaluation to population are shown in FIGURE 5. The processes of representation, population initialization, evaluation, selection and offspring evaluation to population are illustrated in FIGURE 7. The numbers represent the processes evaluation, selection and offspring evaluation to population are shown in FIGURE 5.

![FIGURE 6. DISTRIBUTED CROSSOVER AND MUTATION.](image)

The processes of representation, population initialization, evaluation, selection and offspring evaluation to population are illustrated in FIGURE 7. The numbers represent the processes evaluation, selection and offspring evaluation to population are shown in FIGURE 5.

![FIGURE 7. DISTRIBUTED GENETIC ALGORITHM CLUSTERING PROCESS FLOW.](image)

**FIGURE 7. DISTRIBUTED GENETIC ALGORITHM CLUSTERING PROCESS FLOW.**

![TABLE II. MODULARITY MAXIMIZATION COMPARISON](image)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>GN</th>
<th>CNM</th>
<th>L Max</th>
<th>GATHB</th>
<th>MOG</th>
<th>A-Net</th>
<th>Our Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karate</td>
<td>0.4</td>
<td>0.380</td>
<td>0.419</td>
<td>0.4</td>
<td>0.416</td>
<td>0.416</td>
<td></td>
</tr>
<tr>
<td>Dolphins</td>
<td>0.52</td>
<td>0.495</td>
<td>0.523</td>
<td>0.52</td>
<td>0.505</td>
<td>0.528</td>
<td></td>
</tr>
<tr>
<td>Football</td>
<td>0.6</td>
<td>0.577</td>
<td>0.61</td>
<td>0.55</td>
<td>0.515</td>
<td>0.539</td>
<td></td>
</tr>
<tr>
<td>Books</td>
<td>0.51</td>
<td>0.502</td>
<td>0.526</td>
<td>0.52</td>
<td>0.518</td>
<td>0.523</td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 8. SELECTION AND CONVERGING TO MAX MODULARITY, KARATE CLUB NETWORK.**

The results in TABLE II. and TABLE III. prove that our approach provides results that converge to optimal solutions, and the quality of the results, compared to other popular algorithms and framework, are better in most cases. Some cases showed slightly lower modularity. However, the novel encoding technique that we used, along with the objective functions, made it possible to handle larger graphs. Also our approach can apply evolutionary operators efficiently since the encoded solutions are relatively small compared to other methods.

We found that selection plays a role in converging the solutions. Furthermore, for certain datasets, binary selection converges to higher modularity in a smaller amount of generations, whereas, random selection can provide a higher rate of jumps from local optimal fitnesses. See FIGURE 8, below.

![TABLE III. OPTIMAL MODULARITY](image)

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Optimal modularity</th>
<th>DEGA-Gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karate</td>
<td>0.4198</td>
<td>0.416</td>
</tr>
<tr>
<td>Dolphins</td>
<td>0.5285</td>
<td>0.528</td>
</tr>
<tr>
<td>Football</td>
<td>0.6046</td>
<td>0.539</td>
</tr>
<tr>
<td>Books</td>
<td>0.5272</td>
<td>0.523</td>
</tr>
</tbody>
</table>

**VI. EXPERIMENTAL RESULTS**

We validated the correctness of our clustering algorithm and made sure it produced valid and comparable results. We chose some well-known small datasets carefully and made sure they were the same datasets used in previous studies for comparison. These sets are Bottlenose Dolphins network; US Political books; American College football dataset; and the Zachary Karate Club [41].

To analyze the convergence of solutions over generations, the evaluations after each population were reported. We
generated graphs and computed trend models by dumping the population array and using the scatter plot to create a visual representation of the outcomes for each generation. We found a correlation between the distribution of modularities and the number of generations to extract such modularities. FIGURE 9. Shows the distribution of modularity Vs. Generation and FIGURE 10. describes the polynomial correlation between the generation and the achieved modularity.

To scale our approach for big data we used LUBM to generate RDF graph data and deploy on a cluster with the following properties, as in TABLE IV. The configurations we used yield to 87 container each with access to all 48 disks and have 2 CPU cores as well as 5GB of memory.

![FIGURE 9. POPULATION FITNESS VS GENERATION SCATTER PLOT.](image)

**TABLE IV. CLUSTER AND CONFIGURATIONS**

<table>
<thead>
<tr>
<th>Machine</th>
<th>Threads</th>
<th>Memory</th>
<th>Disks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>72</td>
<td>64</td>
<td>10</td>
</tr>
<tr>
<td>Node1</td>
<td>56</td>
<td>64</td>
<td>10</td>
</tr>
<tr>
<td>Node2</td>
<td>40</td>
<td>64</td>
<td>10</td>
</tr>
<tr>
<td>Node3</td>
<td>40</td>
<td>64</td>
<td>10</td>
</tr>
<tr>
<td>Node4</td>
<td>16</td>
<td>96</td>
<td>2</td>
</tr>
<tr>
<td>Node5</td>
<td>16</td>
<td>48</td>
<td>6</td>
</tr>
</tbody>
</table>

![FIGURE 10. KARATE CLUB TREND MODEL DISCRIPION.](image)

We generated multiple datasets with different sizes to compare the behavior of our algorithm. We analyzed the execution time of initializing a population of solutions; the population size is 1000 solutions per generation. FIGURE 11. shows the trend lines model for LUBM 30M.

**TABLE V. POPULATION INITIALIZATION & ALGORITHM RUN TIME**

<table>
<thead>
<tr>
<th>Number of triples</th>
<th>Initialize Population (S)</th>
<th>Algorithm Run Time (Minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8,970,048</td>
<td>13.556</td>
<td>21.7</td>
</tr>
<tr>
<td>20,637,270</td>
<td>19.621</td>
<td>31.6</td>
</tr>
<tr>
<td>30,285,222</td>
<td>28.611</td>
<td>47</td>
</tr>
</tbody>
</table>

![FIGURE 11. MAXIMIZING MODULARITY OVER GENERATIONS LUBM 30M.](image)

It is important to address that the encoding technique we introduced created a solution space with many local optimal solutions. Hence, using a mutation rate of 100% and multiple points crossovers became a must to not to fall in local optimals. Even though the mutation rate was set to 100 percent, old solutions that have a high rank in each group of tournament in selection process was copied and produced siblings via crossover only; also, considering intra-cluster edges as inter-cluster edges results in no difference in modularity for a given solution. Hence, such mutations do not change the solution, and this effect happens with a high ratio (inter-cluster edges/intra-cluster edges for the corresponding graph).

**FIGURE 11. MAXIMIZING MODULARITY OVER GENERATIONS LUBM 30M.**

**TABLE V. POPULATION INITIALIZATION & ALGORITHM RUN TIME**

We generated multiple datasets with different sizes to compare the behavior of our algorithm. We analyzed the execution time of initializing a population of solutions; the population size is 1000 solutions per generation. FIGURE 12. describes the trend lines model for LUBM 30M.

**FIGURE 12. LUBM 30M TREND MODEL DISCRIPION.**

**FIGURE 13.** describes the modularities and its count in all generations. Since most of the intra-cluster edges do not affect the number of communities produced, having such edges in solutions do not affect the total fitness and explains these high

**FIGURE 13.** describes the modularities and its count in all generations. Since most of the intra-cluster edges do not affect the number of communities produced, having such edges in solutions do not affect the total fitness and explains these high
modularity counts for non-maximal modularity. Yet, having these quadruples in the solution did not affect the algorithm’s ability to jump out of such cases.

FIGURE 13. COUNT OF MODULARITY FOR ALL GENERATIONS AS MODULARITY BINS LUBM 30M.

On a larger scale, we used 221M Quads initialized by converting LUBM triples. FIGURE 14. shows the distribution and the correlation between the fitness measure and its frequency across all generations. FIGURE 15. shows statistics for each generation modularity.

FIGURE 14. Converging to Max modularity Over Generations LUBM 242M.

FIGURE 15. Converging to Max modularity Over Generations LUBM 242M.

VII. CONCLUSION

In this work, we proposed an evolutionary based data clustering scheme, that utilize the capabilities of distributed environment to optimize the encoding of solutions in GA based clustering. We developed parallel evolutionary operations of selection, crossover, mutation and evaluation to adapt with the new distributed encoding. This work is an enhancement of our previous study in, [20], [43] and [39]. This work is a step toward another work in progress that uses clustering to optimize HDFS in handling modern big data.

REFERENCES


