VOL. 12, NO. 23, DECEMBER 2017 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2017 Asian Research Publishing Network (ARPN). All rights reserved.

www.arpnjournals.com

ITERATIVE PARALLEL GENETIC ALGORITHM FOR DETECTING COMMUNITIES IN SOCIAL NETWORKS

Nikhil K. S., Ambika B. and M. V. Judy
Departmentof Computer Science and IT, School of Arts and Sciences, Amrita University, Kochi, India
E-Mail: prodigionikhil@gmail.com

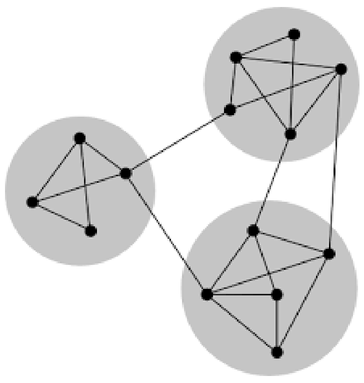
ABSTRACT

A social network is basically a graph where nodes or vertices represent users/actors and links or edges represent the relationship among the actors. Analysis of social networks, especially community detection, is a continuously evolving research area. Genetic algorithms have been proven to be a fruitful method for detection of communities in social networks but the user time taken to detect these communities in large scale networks is quite considerable. In this paper, we enhance a simple genetic algorithm with optimum population size, mutation rate and selection strategy by parallelizing with MapReduce architecture for detecting quality community structures in a shortened time frame. We have used an enhanced framework for MapReduce which increases the performance of the genetic algorithm in a distributed environment. The result shows that the iterative parallel genetic algorithm (IPGA) converges to the optimized solution faster than the traditional method.

Keywords: community detection, Hadoop, MapReduce, parallel genetic algorithm.

1. INTRODUCTION

Social networking is one of the most widely used applications within the domain of the internet in the present day. Previously, users of the internet were consumers of information, but after the dawn of social networking, users are both the producers and consumers of information. Researchers are increasingly interested in the analysis of social networks due to different features of social networking data: the popularity of social networks, the ability to represent social networks as graphs, the availability of large volumes of data, and commercial interest. To analyze a social network, we have to represent the network as a graph G(V,E), where V is a set of nodes or actors and E is a set of edges or links that define the relationship between the actors. An edge represents an interaction (like, retweet, share, friendship) between two nodes. A feature of social networks is community structure and detection of communities in these networks is an important problem. In social networks, collections of actors are considered as communities. The characteristic feature that differentiates these communities is a high number of associations or interconnections among actors within a single community. Nodes within a community are far densely interconnected with each other and have a lower number of connections with nodes belonging to another community. In the graph below, the shaded portions are communities.



Parallel Genetic algorithms have proved to be a successful methodology for detecting communities because the population based characteristic of GA allows the computation of fitness function of each individual in parallel. Hadoop MapReduce is a framework for developing applications that rapidly process vast amounts of data in parallel on large clusters of computing nodes. Therefore, it is an ideal candidate for high scalable parallelization of GAs. In this paper, we demonstrate the how a normal GA can be converted into map and reduce primitives. We implement the non-conventional MapReduce program and demonstrate its scalability to detect communities in large social networks.

2. RELATED WORKS

A. Community detection

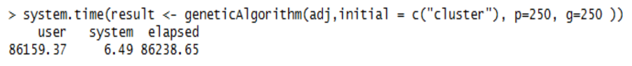
The community detection method proposed by Girvan and Newman became the base for further research in this field [2]. It was in this paper the concept of Modularity (Q) was defined. Modularity is popular community quality detection measure to measure the quality of communities detected. Many researchers proposed various algorithms such as fast greedy [3], label propagation [4], leading eigenvector [5], multilevel [6], optimal, spinglass [8], walktrap [9], infomap [10] etc. that are based on the concept of modularity optimization with different or altered approaches. However, some of these algorithms require prior knowledge of the network (number of edges, vertices, communities etc.) whereas others performed poorly against large networks.

B. Genetic algorithm

Genetic Algorithm is a search heuristic optimization technique that mimics the manner of natural evolution [7]. The principle that governs GA is 'select the best and discard the rest'. It is a methodto be considered especially when the solution space of the problem is vast and an exhaustive search for the most optimum solution is unfeasible. The algorithm repetitively modifies a population (all possible solutions to the problem) of individual solutions called chromosomes. At each step, the genetic algorithm randomly chooses the fittest individuals from the current population and uses them as parents to produce the offspring for the next generation. Fittest individuals are determined by comparing the individual with a certain value derived from the fitness function of the GA. Over successive generations, the population "evolves" to form an optimal solution for that particular problem.

C. Traditional GA for community detection

GA is quite effective in detecting communities [1]. In the traditional GA for community detection, network modularity value is used as the fitness value for each solution member. The algorithm commences by initializing a population. Each "chromosome" is represented by an array that has n elements (corresponding to the number of nodes). Each chromosome represents a community structure. Several chromosomes holding different community configurations form the initial population. Genetic operations are performed for many iterations on the initial population to obtain the optimal solution. Each iteration involves evaluation of fitness value of each chromosome, cross-over between individual members and mutation to form a new population ready for the next iteration. In every iteration, chromosomes are sorted based on their fitness values and the fittest chromosomes are retained for the next generations. This way (survival of the fittest), a good chromosome is never lost. In [1], the authors tested the accuracy of the GA on the well-known Zachary Karate Club dataset and obtained promising results. The algorithm ran successfully dividing the dataset into two communities just like the division within the club in real. Although the GA delivered acceptable results, it took a considerable amount of time to generate the exact result.



```
> system.time(result <- geneticAlgorithm(adj,initial = c("cluster"), p=250, g=250 ))
    user   system  elapsed
86159.37     6.49 86238.65
```

Above is the result of the simulated GA on RStudio. The underlying hardware consisted of Intel® Core™ i5-490S CPU @ 3.00Ghz and 8 GB RAM. This is of course not the best system configuration in the world, nor is it the worst. This configuration is what can be expected on most users' systems. Therefore, a speedup of the GA would be largely desirable. We also observed that the detected communities were not exactly like the real-world division in the Karate club dataset.

D. Parallel community detection on large graphs

You Limitations in application of pre-existing community detection algorithms came into the light when these algorithms were used to detect communities on large scale networks. There are several studies on parallelizing the algorithms for identifying community structures in networks. The size of social networks is growing by the minute and the use of the earlier community detection algorithms is almost obsolete. MapReduce models have been used to tackle these large-scale networks [13]. Bahmani *et al.* [14] proposed an algorithm for finding the densest subgraph and implemented it using the MapReduce model. Li *et al.*

[15] proposed MR-LPA,a parallel version of the label propagation algorithm using the MapReduce model. Yang and Lonardi [16] presented a parallel implementation of the Girvan-Newman algorithm. The evaluation results showed that the time taken to detect optimal communities decreased almost linearly as the number of reducers increased with negligible errors.

3. IMPLEMENTATION USING ITERATIVE PARALLEL GA WITH NON-CONVENTIONAL MAPREDUCE

The strategies suggested in the literature to parallelize Genetic Algorithms are discussed from [11]. Different approaches can be used to parallelize genetic algorithms. The most popular are: the fitness evaluation level (i.e. global parallelization model), population level (i.e. coarse-grained parallelization or island model) and the individual level (i.e. fine-grained parallelization or grid model). In the global parallelization model, a node acting as a master, manages the population (i.e. applying genetic and selection operators) and distributes the individuals among slave nodes which compute the fitness values of the individuals. The key advantage of using this model is that it does not require any change to the design of traditional GA since the individual fitness evaluation is independent from the rest of the population. In the island model, the population is divided in several subpopulations of relatively large sizes which are in several islands (i.e. nodes). A Genetic Algorithm is executed on each subpopulation and subpopulations exchange information by allowing some individuals to migrate from one island to another per given temporal criteria. The main advantages of this model are: (i) different subpopulations explore different portions of the search-space; (ii) migrating individuals inject diversity into the converging population. Finally, in the grid model each individual is placed on a grid (i.e. each individual is assigned to a node) and all GA operations are performed in parallel, evaluating simultaneously the fitness value and applying locally the selection and genetic operations to a small neighborhood.

The main drawback of this approach is the overhead due to the numerous communications between grid nodes.

A. Enhanced MapReduce framework

MapReduce is a smart and flexible paradigm that enables the development of large-scale distributed applications. It is expressed in terms of two distinct functions, Map and Reduce, which are combined in a divide-and-conquer manner. The Map function's responsibility is to handle the parallelization while the Reduce function gathers and merges the results. A master node splits the initial input into several pieces where each piece is identified by a unique key, and allots them via the Map function to several slave nodes (i.e., Mappers) which work independently but in parallel, performing the same task on a different piece of input. As soon as a Mapper finishes its own job, the output is identified and collected via the Reducer function. Each Mapper emits a set of intermediate key/value pairs which are used by the Reducers to group all the intermediate values associated to the same key and to compute the list of output results. The program automatically invokes and allocates several distinct Reducers that correspond to the number of distinct intermediate keys. Several different implementations of MapReduce are available for use today. Our algorithm is implemented using the non-conventional Hadoop MapReduce [17] where the sorting mechanism is bypassed and invocation of the reduce function is modified so that it can be called with a single record. Reducers no longer wait to remotely read from Mappers and then to be grouped. Due to this, performance is improved as Reducers need not wait until the Mappers complete their entire work and shuffling gets completed. Intermediate results foreach key is not stored. Separate threads are maintained for every Mapper. A single buffer retrieves all records. A separate thread executes the Reduce function and is passed one record at a time from the buffer in a first in first out manner.

B. Proposed iterative parallel GA based on MapReduce

A hybrid model by combining the first and second parallelization strategies using the enhanced MapReduce is the basic idea behind the algorithm. The distributed model design [18] based on the polytypic concept of a species being represented by several types that are capable of mating and producing promising offspring is used. Breeding and evaluation are typically carried out in isolation on each island. To be consistent with biological motivations, it was noted that migration should occur after a period of stasis. However, difficulty in defining stasis or equilibrium meant that migration occurred after G generations.

In the sequential version, all processing is performed on a single processor, whereas in the parallel version, the processing load is divided among several processors/mappers (slaves), under the direction of a master processor. An iteration of the GA is encapsulated as a separate MapReduce job and the chromosome fitness evaluation task is parallelized among several *Mappers*. *Reducers* collect the results concerning each island based on the key value and perform genetic operations such as parent selection, crossover and mutation, survival selection and migration process which are necessary to produce a new generation following a global parallelization model. Migration is done after G generations during which the best individuals in each island are sent to each neighbour thereby replacing the worst individuals.

Figure-1 shows the proposed architecture based on Hadoop MapReduce and is composed of the following main components: A Parallel Genetic Algorithm, a Master, many Mappers and Reducers (considered as an island), together with two other units, namely Input Stream and Output Stream, which are responsible to split the data for the Mappers and to store the *Reducer* output into the Hadoop Distributed File System (HDFS) respectively.

These components communicate with each other exploiting the HDFS distributed file system provided by Hadoop, while the communications within the Hadoop framework (i.e., those between the master and slave nodes) are carried out via socket using SSH (Secure SHell).The Parallel Genetic Algorithm module takes a sequence of community structures as input. Once the GA terminates, it returns a predicted structure as output.

Split phase: In this phase the *InputFormat* module gets the current population (i.e., the sequence of community configurations composing the current population) from the HDFS and processes it to split it in crunch of data (i.e., input split) to be distributed among the *Mapper* modules. The number of input splits is dynamically computed based on the number of available *Mappers*.

The *Master* module is responsible to coordinate and supervise the assignment of resources and the computations, taking care also of the load balancing aspects. Once the *InputFormat* begins to emit the *<key, value>* pairs - exploiting the RecordReader component of Hadoop - the underlying Hadoop framework is automatically notified and the *Master* component is invoked to assign the input split produced by the *InputFormat* to the available *Mappers*

Map phase: In this phase, each Mapper carries out its task on the received input split in a parallel and independent way. Each Mapper performs the genetic operations, once such evaluation is completed, each Mapper generates a new pair <key, value>, where value is a pair <chromosome, fitness value>. The key is generated by the Master module based on the number of islands and is assigned to Reducers.

Reduce phase: As soon as a Mapper evaluates a chromosome, the corresponding data (i.e., key (island number), chromosome, and fitness value) is sent to the Reducer. Once the entire population corresponding to the island is available to the Reducer, it can perform the survival selection and apply on the new generation the crossover and mutation operators to produce new offspring to be evaluated in the next MapReduce job. To obtain the entire population, the Reducer must wait until all Mappers have replied although genetic operations can be performed as and when it receives a pair of chromosomes from the single buffer.

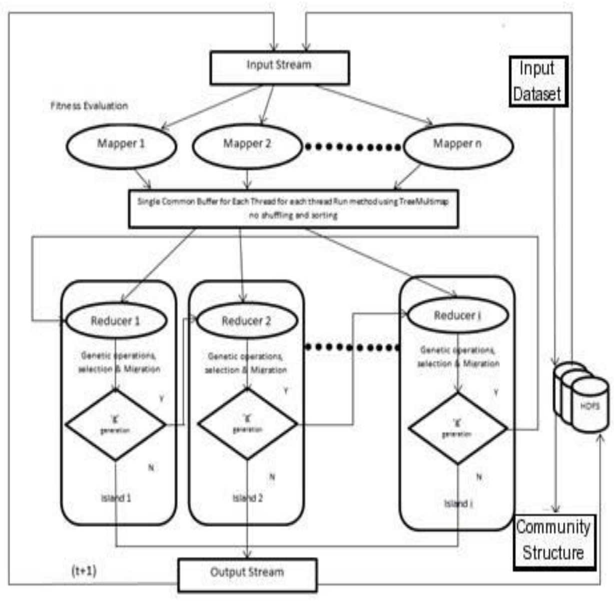
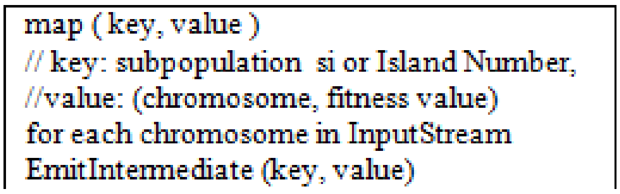


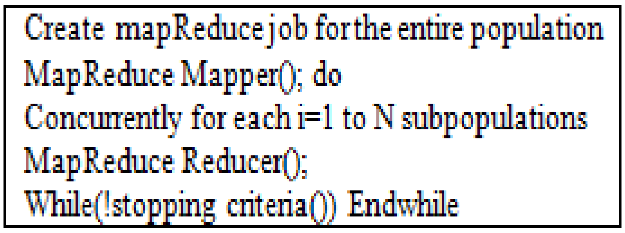
Figure-1. Architecture of the proposed GA with enhanced MapReduce framework.

Table-1. Pseudocode for parallel GA.



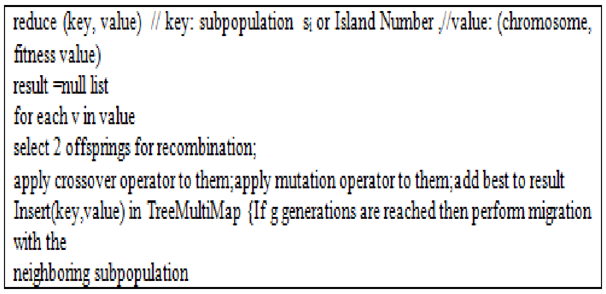
```
map ( key, value )
// key: subpopulation  si or Island Number,
//value: (chromosome, fitness value)
for each chromosome in InputStream
EmitIntermediate (key, value)
```

Table-2. Pseudocode for MapReduce Mapper ().



```
Create mapReduce job for the entire population
MapReduce Mapper(); do
Concurrently for each i=1 to N subpopulations
MapReduce Reducer();
While(!stopping criteria()) Endwhile
```

Table-3. Pseudocode for MapReduce Reducer ().



```
reduce (key, value) // key: subpopulation si or Island Number ,//value: (chromosome,
fitness value)
result =null list
for each v in value
select 2 offsprings for recombination;
apply crossover operator to them;apply mutation operator to them;add best to result
Insert(key,value) in TreeMultiMap {If g generations are reached then perform migration
with the
neighboring subpopulation
```

However, selection can be done only after all chromosomes undergo genetic operations. Also, each Mapper evaluates approximately the same number of chromosomes and if they are executed in parallel, they require approximately the same time. Finally, the data regarding the new offspring is saved by the Output Format - using the Record Writer - into the HDFS, allowing the Parallel Genetic Algorithm module to verify whether the termination criteria holds. The Master module in this phase is responsible to notify the Parallel Genetic Algorithm to restart computation by invoking the MapReduce Job for new offspring created by Reducer.

Comparison between traditional GA and iterative parallel GA

Table-4 shows the results from running a GA to find the optimum community structure in the Zachary's Karate Club dataset and best structure for community detection problem. It is observed that the Parallel GA finds optimum solution much faster than traditional GA.

Table-4. Comparison between traditional GA and parallel GA.

Algorithm	Modularity	Community sizes	Time taken
GA	0.42	2	86238
IPGA	0.42	2	10770

The Table shows the variations in performance of compared methods. We used higher mutation rates in combination with crossover. The population size was 250.With PGA we could speed up convergence of GA and detect good quality communities.

RESULTS AND FURTHER DISCUSSIONS

Parallel genetic Algorithm outperforms the traditional GA significantly on community detection problems, especially as the problem size increases in terms of time and optimality. It is also seen that IPGA too detects quality community structures in social networks.

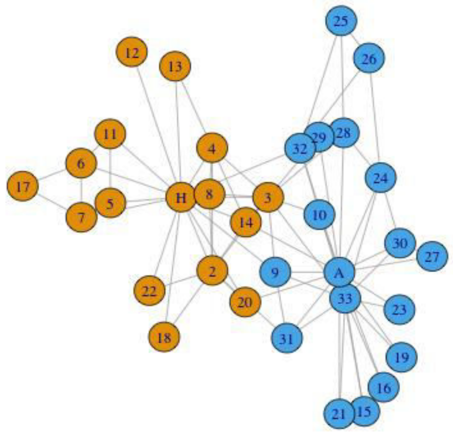


Figure-2. Communities detected in Zachary's Karate Club.

Results also showed that a random migration selection was more effective than aggressive selection strategy which caused the population to get stuck in local optima. Memory management is one of the main problems that could arise for large sets of data as obtained partial results are stored after the Map Stage in memory only. This could result in overflow. A suggested solution to this problem could be to move the contents which are least recently used into files. A Hash table could be used to keep the track of files which have been moved on to the file and for faster access. Various other hybrid models of parallelization can be tested on benchmark problems like community detection.

REFERENCES

[1] Mursel Tasgin, Amac Herdagdelen and Haluk Bingol. 2008. Community Detection in Complex Networks Using Genetic Algorithms. Department of Computer Engineering Bogazici University, Turkey (Dated: February 2).

[2] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. Department of Physics and Center for the Study of Complex Systems, University of Michigan, Santa Fe Institute, Department of Physics, Cornell University.

[3] Newman M. 2004. Fast algorithm for detecting community structure in networks. Phys. Rev. E. 69, 066133.

[4] Raghavan U.N.; Albert R.; Kumara S. 2007. Near linear time algorithm to detect community structures in large-scale networks. Phys. Rev. E. 76, 036106.

[5] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. Department of Physics and Center for the Study of Complex Systems, University of Michigan.

[6] Bruce Hendrickson and Robertleleand. A Multilevel Algorithm for Partituoning Graphs. Sandia National Laboratories.

[7] Holland J.H. 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan.

[8] Jorg Reichardt1 and Stefan Bornholdt1. Statistical Mechanics of Community Detection. 1Institute for Theoretical Physics, University of Bremen, German.

[9] Pascal Pons and MatthieuLatapy. Computing Communities in Large Networks Using Random Walks. LIAFA - CNRS and Universit´e Paris 7

[10] Martin Rosvall, Carl Bergstrom. Maps of random walks on complex networks reveal community structure. Proceedings of the National Academy of Sciences.

[11] Di Geronimo L.; Ferrucci F.; Murolo A.; Sarro F.A. 2012. Parallel Genetic Algorithm Based on Hadoop MapReduce for the Automatic Generation of JUnit Test Suites. 2012 IEEE Fifth International Conference on, Issue Date: 17-21.

[12] W N Martin, Jens Lienig and James P Cohoon. Population Structures Island (migration) models: evolutionary algorithms based on punctuated equilibria. Handbook of Evolutionary computation, May 97 Release.

[13] Seunghyeon Moon a, Jae-Gil Lee b, MinseoKangb. Scalable Community Detection from networks by computing edge betweenness on MapReduce. KAIST Institute for IT Convergence; Department of Knowledge Service Engineering, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea.

[14] B. Bahmani R., Kumar S., Vassilvitskii. Densest subgraph in streaming and MapReduce.

[15] Q. Li, Z. Wang, W. Wang, Y. Liu, P. Wang, T. Yu. 2011. LI-MR: a local iteration map/reduce model and its application to mine community structure in large-scale networks. Proceedings of the 2011 IEEE International Conference on Data Mining Workshops (ICDMW). pp. 174-179.

[16] Q. Yang, S. Lonardi. 2007. A parallel edge-betweenness clustering tool for protein–protein interaction networks. Int. J. Data Min. Bioinform. 1(3): 241-247.

[17] Ashutosh Rajan and M V Judy.2013. An Enhanced Map Reduce Framework for Improving the Performance of Massively Scalable Private Clouds. International Journal of Computer Applications (IJCA), Proceedings on Amrita International Conference of Women in Computing - 2013 AICWIC(3): 24-26. Published by Foundation of Computer Science, New York, USA.

[18] Wiese K. and Goodwin, S.D. 1998. Parallel Genetic Algorithms for Constrained Ordering Problems. Proceedings of the 1lth International Florida Artificial Intelligence Research Symposium, FLAIRS'98, pp. l0l-105.