Vol. 5(15) Apr. 2015, PP. 2198-2206

Contents list available at IJMEC

International Journal of Mechatronics, Electrical and Computer Technology (IJMEC)

Journal Homepage: www.aeuso.org

PISSN: 2411-6173 - EISSN: 2305-0543

Comparing performance of parallel grouping genetic algorithm with serial grouping genetic algorithm for clustering problems

Javad Vahidi[1,*], Seyed Saeed Mirpour Marzuni[2], Sara Farzai[3]

[1]Department of Applied Mathematics, Iran University of Science and Technology, Tehran, Iran

Corresponding Author's E-mail: jvajidi@iust.ac.ir

Abstract

In this paper we compare parallel grouping genetic algorithm with serial grouping genetic algorithm for data clustering. According to our experimental results, the proposed parallel grouping genetic algorithm (PGGA) considerably decreases the CPU time without inversely influence on the answer.

Keywords: grouping genetic algorithm, clustering

1. Introduction

Clustering is an unsupervised pattern classification method that partitions the input space into clusters. The goal of a clustering algorithm is to perform a partition where objects within a cluster to be similar and objects in different clusters to be dissimilar. Clustering has been applied to a wide variety of problems in many different fields such as pattern recognition, bioengineering, image quantization, renewable energy prediction, etc. Once a clustering algorithm has processed a dataset and obtained a partition of the input data, a relevant question arises: How well does the proposed partition fit the input data? This question is relevant for two main reasons. First, an optimal clustering algorithm does not exist. In other words, different algorithms — or even different configurations of the same algorithm — produce different partitions and none of them have proved to be the best in all situations [8]. Thus, in an effective clustering process we should compute different partitions and select the one that best fits the data. Secondly, many clustering algorithms are not able to determine the number of natural clusters in the data, and therefore they must initially be supplied with this information—frequently known as the k parameter. There are numerous algorithms for clustering problem. In the last few years, evolutionary computing algorithms (EAs) have been widely applied to very different problems with very few changes, and also because these algorithms are able to manage constraints in an efficient way.

Genetic algorithms (GAs) are powerful search techniques that are used to solve difficult problems in many disciplines. Unfortunately, they can be very demanding in terms of computation load and memory. Parallel genetic algorithms (PGAs) are parallel implementations of GAs which can provide considerable gains in terms of performance and scalability. PGAs can easily be implemented on networks of heterogeneous computers or on parallel mainframes. This paper presents grouping genetic algorithm for clustering problem in second section. In third section parallel genetic algorithm are presented and in last section result are presented.

2. Grouping genetic algorithm [2]

The grouping genetic algorithm (GGA) is a class of evolutionary algorithm especially modified to tackle grouping problems, i.e., problems in which a number of items must be assigned to a set of predefined

Article History:
Received Date: Dec.01, 2014
Accepted Date: Mar. 29, 2015
Available Online: Apr. 19, 2015

groups. It was first proposed by Falkenauer (1992, 1998), who realized that traditional genetic algorithms had difficulties when they were applied to grouping problems. Thus, in the GGA, the encoding, crossover and mutation operators of traditional Gas are modified to obtain a compact algorithm, with a high performance in grouping-based problems.

2.1. Encoding

The encoding is carried out by separating each individual in the algorithm into two parts: $c = [l \mid g]$, the first part is the element section, whereas the second part is called the grouping section of the individual. As an example to clarify the GGA encoding in clustering, let us suppose the following individual:

1 2 1 4 5 4 3 2 1 5 5 3 4 1 5|1 2 3 4 5

This individual represents a solution with 5 clusters, and the following partition of input data: $\{x_1, x_3, x_9, x_{14}\}, \{x_2, x_8\}, \{x_7, x_{12}\}, \{x_4, x_6, x_{13}\}$ and $\{x_5, x_{10}, x_{11}, x_{15}\}$.

2.2. Selection

When we generated the initial population, chromosomes are selected from the population to be parents to crossover. GGA use a rank-based wheel selection mechanism [2], similar to the on described in James et al. (2007). First the individuals are stored in a list based on their quality. The position of the individuals in the list is called rank of the individual, and denoted R_i, $i = 1,2,..., \xi$, with ξ number of individuals in the population of GGA. We consider a rank in which the best individual x is assigned $R_x = \xi$, the second best y, $R_y = \xi - 1$, and so on. A fitness value associated to each individual is then defined, as follows:

$$f_i = \frac{2.R_i}{\xi(\xi+1)} \qquad (1)$$

Note that these values are normalized between 0 and 1, depending on the position of the individual in the ranking list.

The process carried out in GGA consists of selecting the parents for crossover using this selection mechanism. This process is performed with replacement, i.e., a given individual can be selected several times as one of the parents, however, individuals in the crossover operator must be different.

2.3. Crossover

The crossover operator implemented in GGA selects two parents and render one offspring with following steps:

1. First, two individuals are randomly selected, and two crossing points are chosen in their group part.
2. Insert the elements belonging to the selected groups of the first individual into the offspring, if they have not been assigned by the first individual.
3. Insert the elements belonging to the selected groups of the second individual into the offspring, if they have not been assigned by the first individual.
4. Randomly complete the elements not yet assigned with elements from the current groups.
5. Remove empty clusters, if any.
6. Modify the labels of the current groups in the offspring in order to numerate them from 1 to k.

2.4. Mutation

Mutation caused to search unknown areas of problem space. One can deduce that the most important task of mutation is escaping from local optimum. In GGA, this operator was implemented for two different mutation operators adapted to the clustering problems:

- Mutation by cluster splitting: it consists of splitting a selected cluster into two different ones. The samples belonging to the original cluster are assigned to the new clusters with equal probability. Note that one of the new generated clusters will keep its label in the group section of individual, whereas the other will be assigned a new label (k+1). The selection to the initial cluster to be split is carried out depending on the clusters' size, with more probability of split given to larger clusters.
- Mutation by clusters merging: it consists of merging two existing clusters, randomly selected, into just one. As in mutation by cluster splitting, the probability of choosing the clusters depends on their size.

The general form of a flowchart of the GGA algorithm is as follows offered.

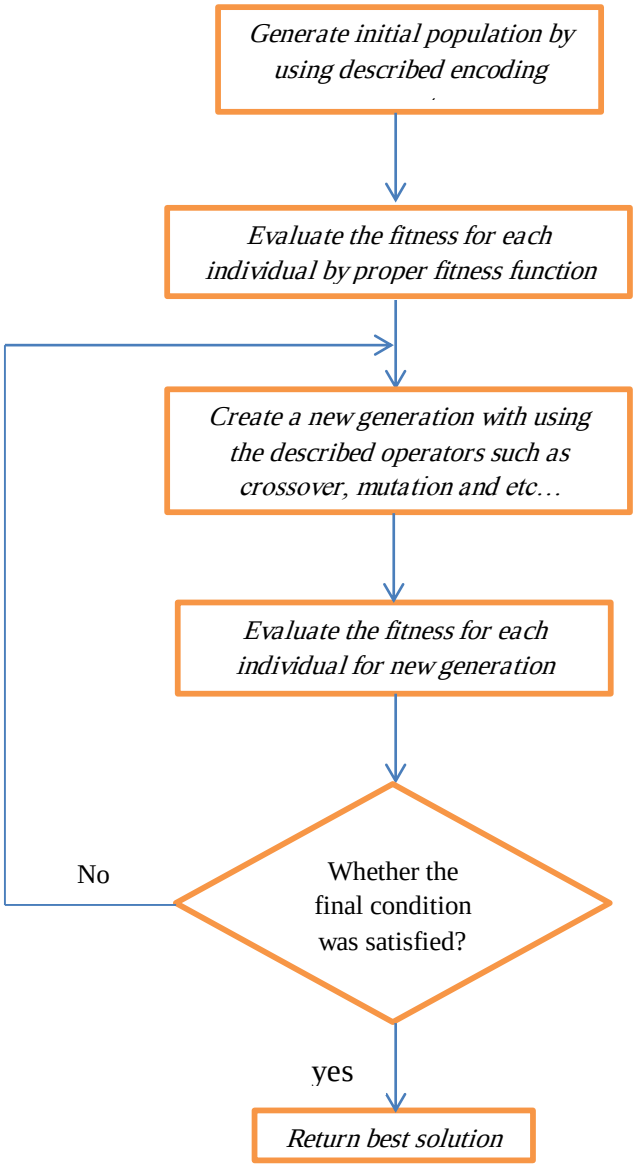

Fig 1: Flowchart of grouping genetic algorithm

3. Parallel genetic algorithm

Parallel computing, or the concurrent operation of separate processing units, has for a long time been used as a technique to derive better performance from a given technology. Over the years various types

of concurrency within a computer system evolved to enable several operations to occur at once. Flynn made an early attempt to divide the various types of computer systems into four major categories based on the number of instruction and data streams that are processed simultaneously [6,7]. Flynn's categories are as follows:

- Single instruction stream, single data stream machines (SISD).
- Single instruction stream, multiple data stream machines (SIMD).
- Multiple instruction stream, single data stream machines (MISD).
- Multiple instruction stream, multiple data stream machines (MIMD).

The SISD architecture executes a single program as one sequential procedure using a single set of data and is still today's most used type of computer (e.g. Intel 80486). SIMD machines may invoke a plurality of data operations to occur concurrently during one instruction cycle. Generally a SIMD machine is massively parallel, with hundreds or thousands of processor elements. Systems with this architecture are restricted in the problems that they can solve, but are particularly suited to operations on large matrices. MISD machines allow actions within an instruction cycle to overlap with different actions of consecutive instruction cycles on a single data stream to achieve a higher rate of instruction execution. However, machines with these architectures are very rare. Lastly, the MIMD organization allows multiple sets of possibly different instructions to execute separately and concurrently on multiple sets of data. Processors can communicate with each other via messages.

Genetic algorithms lend themselves excellently for effective parallelization, giving their inspiring principle of evolving in parallel a population of individuals.

The different models used for distributing the computation and to ease parallelization, cluster around three main approaches which will be explained in the next sections. First, a parallel GA similar to the traditional GA model will be reviewed. Next two approaches are described which decompose the population as a particular model of a GA. The first decomposition approach explained is the island model which divides a population into several equal subpopulations, each of which runs a GA independently and in parallel in respect to the other subpopulations. Occasionally individuals are migrated to another subpopulation. The second decomposition approach explained is the cellular genetic algorithms. This parallel variant of GA divides a population in such a way that each processor typically only has one individual. Each processor then uses GA operators only on individuals in a bounded region. Fit individuals propagate throughout the whole population because all regions overlap each other.

3.1. Global populations with parallelism

In these models one global population is kept, on which the traditional genetic operators are performed. Selection is done on the global population, and then the selected individuals undergo crossover and mutation in parallel. Grefenstette mentions a synchronous master-slave model [8]. Here a single master process keeps the whole population in its own memory, performs selection, crossover and mutation on the individuals, but leaves the calculation of the fitness of the new individuals to k slave processes. The problem with this model is that a lot of time is wasted if some slave processes finish their evaluation significantly quicker than others. Another problem is that much is depending on the master process. Grefenstette then mentions the semisynchronous master-slave model, where an individual is inserted when a slave process is done, and that same process gets a new individual as soon as possible. Again the problem of dependency on one master process is still there. Lastly Grefenstette considers the asynchronous concurrent model. Here the individuals of the population are kept in a common shared memory, which can be accessed by k concurrent processes. Each process performs function evaluations,

but also genetic operations. Each process works independently of the others. Whitley states that the only difference between this last model and the traditional (serial) genetic algorithm is a change in the selection mechanism [9]. Selection can best be done by tournament selection. Assume that one has a population of N individuals and N/2 concurrent processors. Twice each processor randomly selects two individuals from the shared memory, and keeps the best. The two selected strings are then subjected to crossover, mutation and evaluation. The resulting children are put back in the population in the shared memory. This all goes in parallel.

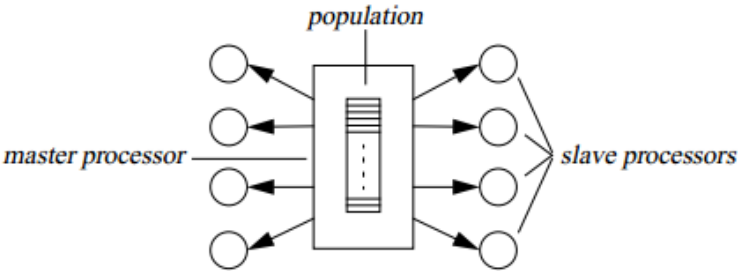

Fig 2: Global population with parallelism

3.2. Island models

Looking at nature again one could say that the world of humans consists of one big population. Another view is to say that it's actually a collection of subpopulations which evolve independently from each other on isolated continents or in restricted regions. Once in a while some individuals from one region migrate to another region. This migration allows subpopulations to share genetic material. If we apply this to genetic algorithms we could parallelize it by letting each processor run its own (sequential) genetic algorithm with its own subpopulation, but each trying to maximize the same function. If a neighborhood structure is defined over the set of subpopulations, and once in a while each subpopulation sends its best individuals to its neighbors, we say we're running a distributed genetic algorithm. If no swapping of individuals to neighbors is done we have a special case of the distributed model, which we call the partitioned genetic algorithm. These models are most suited for MIMD machines.

Since each processor starts with a different initial population genetic drift will tend to drive these populations into different directions. By introducing migration the island model is able to exploit differences in the various subpopulations; this variation represents a source of genetic diversity. However, migrating a large number of individuals too often may drive out any local differences between islands, thus destroying global diversity. On the other hand, if migration occurs not often enough, it may lead to premature convergence of the subpopulations. When dealing with an island model the main issues to be considered are [10]:

- The processors with which each processor exchanges individuals with.
- Migration frequency or how often a processor exchanges individuals.
- Migration rate or the number of individuals exchanged between processors.
- The individuals chosen to exchange.
- The individuals deleted after having received individuals from others.

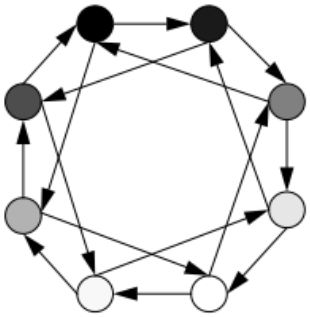

Fig 3: Example of an island model

3.3. Cellular genetic algorithm

Genetic algorithms implemented on a SIMD machine typically have one individual residing at each processor element (cells). Individuals select mates and recombine with other individuals in their immediate neighborhood (e.g. north, south, east and west). This class of genetic algorithms is in fact a subclass of cellular automata. Thus, the term *Cellular Genetic Algorithm* is proposed to describe this class of parallel algorithms. Each individual selects an individual in its local neighborhood as its mate. This can be done by selecting the best individual from among the neighbors or by some local random selection scheme.
The selected individual is then mated with the individual residing in the cell. One offspring is produced and may or may not replace the individual in the cell depending on the replacement scheme chosen. The model is thus fully distributed with no need of any central control. When dealing with a cellular model the main issues to be considered are [10]:

- the neighborhood structure
- the selection scheme
- the replacement scheme

4. Experiments

in this section, first we test parallel grouping genetic algorithm on data set and show the result then each of algorithm (parallel and serial) is run on data set for 5 times and average time of iterations is showed. Reason of 5 repetitions is that GGA is based on random. For this reason, CPU time is different in each iteration. Of course, fluctuation of changes is up to 5 seconds.
In this experiment with artificial dataset, we have tried to create data points with respect to averages and covariance matrices and cluster this data set with PGGA. We evaluate the performance of the PGGA in 2D clustering problems using 200 data points which have been generated randomly by a Gaussian distribution of 9 clusters with equal probability of occurrence and average points of each class and covariance matrices were as follows respectively:

$$\mu_1 = (1,-1), \mu_2 = (-1.5,0), \mu_3 = (0,1), \mu_4 = (-1,1), \mu_5 = (2,-1), \mu_6 = (-2,-1), \mu_7 = (-0.5,2),$$
$$\mu_8 = (-1,-1), \mu_9 = (1.5,0)$$

$$\Sigma_1 = \ldots = \Sigma_8 = \begin{bmatrix} 0.2^2 & 0 \\ 0 & 0.2^2 \end{bmatrix}$$

Fig 4 shows the observations which are randomly generated under mentioned statistical distribution.

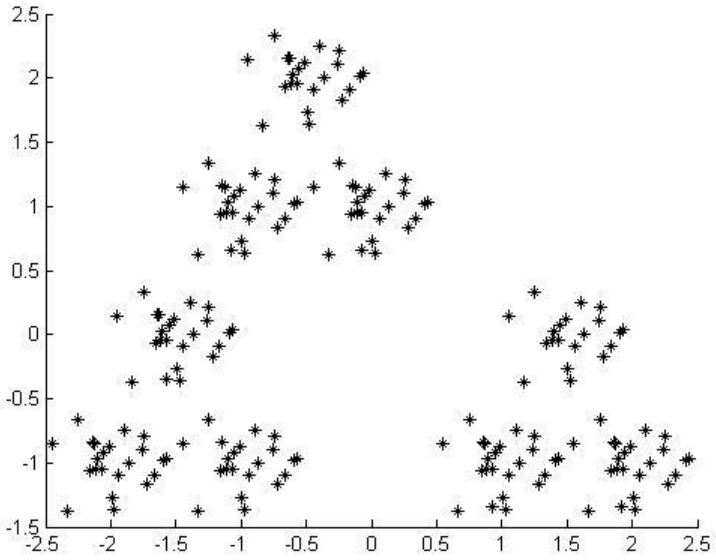

Fig 4: the artificial dataset generated using statistical distribution

Fig 5 shows the clustering results of randomly generated data points using parallel grouping genetic algorithm. As we observe, four clusters are created.

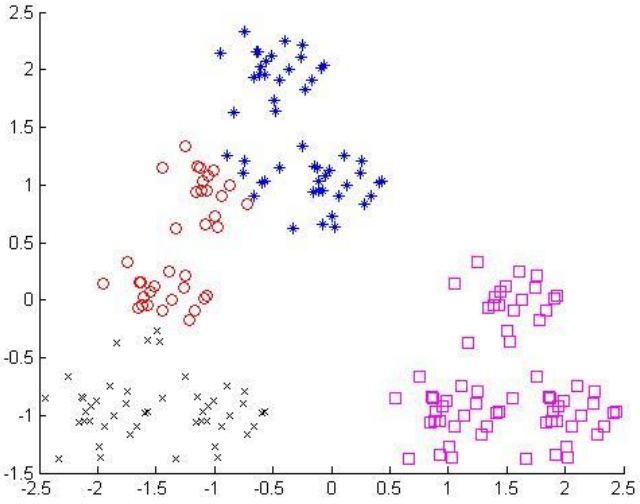

Fig 5: the best clustering result for artificial dataset generated using statistical distribution

Now we simulate parallel grouping genetic algorithm on Intel Core I3 with four cores. We compare CPU time in serial algorithm and parallel algorithm with two and four cores. We also run algorithm in different number of population and different number of generation then show the results in table.

In genetic algorithm we first generate initial population, as we showed in fig 1 in this paper. For creating initial population, we divide number of population to number of CPU then run the function of creating initial population on each CPU. In fact, we run single instruction on all CPU. At the next step we allocate individual to each CPU and reach the fitness of chromosome in parallel.

According to the flowchart of GGA, next step is using genetic operators for creating new generation. Note that in the select operator we need to sort the population based on fitness. To improve CPU time, we apply parallel sort for sorting the population. After that we use master-slave model, as we said in third section. This means we have one shared memory that all population is placed there then we select parents and allocate them to each slave. Each slave carries out its task and replaces the new chromosome instead old one.

Table 1 shows the result of our experiment that we show it as follows:

Table 1: result of experiments

	Number of data = 200		
P=Population G=Generation	Parallel algorithm with 4 cores	Parallel algorithm with 2 cores	Serial algorithm
P=20, G=10	33.26 sec	41.63 sec	58.55 sec
P=30, G=20	67.70 sec	91.00 sec	135.15 sec
P=50, G=30	110.50 sec	149.22 sec	221.26 sec
P=100, G=50	324.66 sec	430.55 sec	731.12 sec
	Number of data = 50		
P=20, G=10	14.05 sec	15.72 sec	19.75 sec
P=30, G=20	27.52 sec	31.53 sec	46.54 sec
P=50, G=30	42.11 sec	52.25 sec	78.32 sec
	Number of data = 10		
P=10, G=10	6.83 sec	6.37 sec	6.30 sec

We tested algorithm on different number of data and different generation and population as we showed in table 1. We first tested algorithm on 200 data point. As results showed, performance of parallel algorithm is better than serial algorithm, if we have more steps in algorithm. Nevertheless, if we watch results, we see that to having four CPU, the CPU time is not quarter serial algorithm's time. Reason of this matter is communication of CPUs that squandered time of the CPU.
We also result of algorithm on 10 data points because we wanted to show that parallel algorithm is not always work better.

5. Conclusions

In this paper, we have presented a parallel grouping genetic algorithm for clustering problem. We compared performance of PGGA with serial GGA and have showed when problem is short, performance of parallel algorithm will decrease. However, parallel algorithm for long issue has the best performance.

6. References

[1] Falkenauer, E. (1992). The grouping genetic algorithm–widening the scope of the GAs. In Proceedings of the Belgian Journal of operations research, statistics and computer science (Vol. 33, pp. 79–102).
[2] L.E. Agustin-Blas, S. Salcedo-Sanz, S. Jimenez-Fernandez, L. Carro-Calvo, J. Del Ser "A new grouping genetic algorithm for clustering problems" Expert Systems with Applications 39 (2012) 96959703.
[3] Falkenauer, E. (1998). Genetic algorithms for grouping problems. New York: Wiley. Gomez-Munoz, V. M., & Porta-Gandara, M. A. (2002). Local wind patterns for modeling renewable energy systems by means of cluster analysis techniques. Renewable Energy, 2, 171–182.
[4] James, T. L., Brown, E. C., & Keeling, K. B. (2007). A hybrid grouping genetic algorithm for the cell formation problem. Computers & Operations Research, 34, 2059–2079.

[5] James, T., Vroblefski, M., & Nottingham, Q. (2007). A hybrid grouping genetic algorithm for the registration area planning problem. Computer Communications, 30(10), 2180–2190.
[6] M.J. Flynn; 'Very high speed computing systems', *Proc. IEEE*, vol.54, 1901-1909, 1966.
[7] M.J. Flynn; 'Some computer organizations and their effectiveness', *IEEE Trans. Comp.*, vol. C-21, 948-960, 1972.
[8] J.J. Grefenstette; *Parallel adaptive algorithms for function optimization*, (Technical Report No. CS-81-19), Nashville: Vanderbilt University, Computer Science Department, 1981
[9] D. Whitley; *A Genetic Algorithm Tutorial*, (Technical Report CS-93-103), Colorado State University, 1993.
[10] Jan Pit L.(1995). "Parallel Genetic Algorithm".Master's Thesis,Department of Computer Science Leiden University.