Hadoop MapReduce for Parallel Genetic Algorithm to Solve Traveling Salesman Problem

Entesar Alanzi[1], Hachemi Bennaceur[2]
Department of Computer Science
Al Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, Saudi Arabia

Abstract*—Achieving an optimal solution for NP-complete problems is a big challenge nowadays. The paper deals with the Traveling Salesman Problem (TSP) one of the most important combinatorial optimization problems in this class. We investigated the Parallel Genetic Algorithm to solve TSP. We proposed a general platform based on Hadoop MapReduce approach for implementing parallel genetic algorithms. Two versions of parallel genetic algorithms (PGA) are implemented, a Parallel Genetic Algorithm with Islands Model (IPGA) and a new model named an Elite Parallel Genetic Algorithm using MapReduce (EPGA) which improve the population diversity of the IPGA. The two PGAs and the sequential version of the algorithm (SGA) were compared in terms of quality of solutions, execution time, speedup and Hadoop overhead. The experimental study revealed that both PGA models outperform the SGA in terms of execution time, solution quality when the problem size is increased. The computational results show that the EPGA model outperforms the IPGA in term of solution quality with almost similar running time for all the considered datasets and clusters. Genetic Algorithms with MapReduce platform provide better performance for solving large-scale problems.

***Keywords*—Genetic algorithms; parallel genetic algorithms; Hadoop MapReduce; island model; traveling salesman problem**

I. Introduction

Genetic algorithms (GAs) are stochastic search methods that have been successfully applied in many searches, optimization, and machine learning problems [1]. GAs are used to find approximate solutions in a reasonable time for combinatorial optimization problems. One of the main features of genetic algorithms is that they are inherently parallel. This makes them the most suitable for parallelization [2]. Parallel genetic algorithms (PGAs) can improve GAs to search in a huge solution space and reduce the total execution time. In general, there are three main models of parallel GAs: master-slave model, fine-grained model and coarse-grained also called island model.

The island model is a popular and effective parallel genetic algorithm because it does not only save time but also improves global research ability of GA [3]. Recently, the increasing volume of data requires high-performance parallel processing models for robust and speedy data analysis. Thus, the use of large-scale data-intensive applications has become one of the most important areas of computing.

Several technologies and approaches have been implemented to develop parallel algorithms. Hadoop MapReduce represents one of the most mature technologies. MapReduce programming model, proposed by Google [4], has become the prevalent model for processing a vast amount of data in parallel especially on a large cluster of computing nodes. Due to massive parallelization and scalability of MapReduce, it is used to develop parallel algorithms. It provides a ready-to-use distributed infrastructure that is scalable, reliable and fault-tolerant [4], [5]. The power of the MapReduce comes from the fact that it splits the data into smaller chunks processed in parallel by the mappers and merged by the reducers [6]. MapReduce aims to help programmers and developers to primarily focus on their applications on large distributed clusters, and hide the programming details of load balancing, network communication, and fault tolerance. Hadoop is the latest buzzword in cloud computing which implements the MapReduce framework. Hadoop is an Apache open-source software project designed for distributed parallel processing. It is designed to run applications on a big cluster of commodity nodes in a reliable, scalable and fault-tolerant manner. It is also designed to scale up from single servers to thousands of nodes. Each node in the cluster is a machine offers local computation and storage [7]. Hadoop deploys a master-slave architecture for computation and storage.

The basic design idea of MapReduce is inspired by two functions: Map and Reduce. Both Map Tasks and Reduce Tasks, which are written by the user, work on key/value pairs. A MapReduce application is executed in a parallel manner through two phases. In the first phase, all Map tasks can be executed independently. In the second phase, each Reducer task depends on the output generated by any number of Map task. Then, all Reducer tasks start executing their tasks independently [8]. The architecture of the MapReduce framework is shown in Fig. 1.

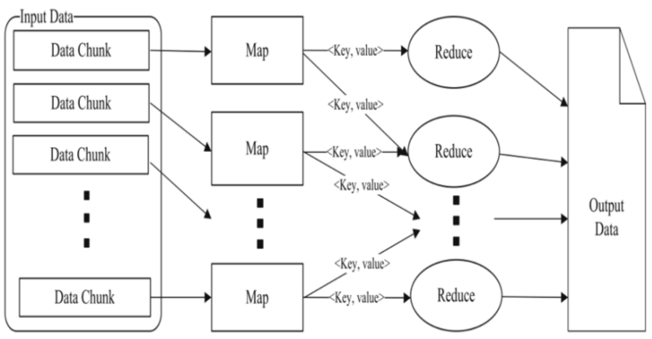


Fig. 1. The Architecture of the MapReduce Framework [8].

Traveling Salesman Problem (TSP) is one of the most common and combinatorial optimization problems in computer science and operations research. Given a set of cities and distances between them, the TSP goal is to find the shortest tour that visits all cities exactly once and returns to the starting city. TSP is easy-to-state but a difficult-to-solve problem since it is an NP-complete problem. TSP is considered enormously important because it can model a large number of real-world problems. Some of the applications include industrial robotics [9], job scheduling [10], computing wiring, DNA sequencing [11], [12], vehicle routing [13], and so forth.

Many methods have been developed to solve the TSP problem. These can be classified into two main categories; exact and heuristics. Exact methods guarantee to find the optimal solution of the problem whereas heuristic methods attempt to provide a good solution in a reasonable time [14]. Genetic Algorithms (GAs) are found to be one of the best metaheuristic algorithms for the TSP problems and yield approximate solutions within a reasonable time [15].

The parallel GAs using Hadoop MapReduce are not always guaranteed better performance than the sequential versions in term of execution time, that because of the overhead produced by the use of Hadoop MapReduce. One of the aims in this work is to understand if and when the parallel GA solutions show better performance.

For this work, two versions of parallel genetic algorithms are implemented using MapReduce to solve the TSP, a Parallel Genetic Algorithm with Island Model (IPGA) proposed in [16], and our proposed algorithm named Elite Parallel Genetic Algorithm (EPGA). We empirically assessed the performance of the two aforementioned PGA models with respect to a sequential GA on TSP problems, evaluating the quality of the solution, the execution time, the achieved speedup and the Hadoop overhead. The experiments were conducted by varying the problem size such as five TSP instances were exploited to differentiate the computation load. Additionally, varying population size and the cluster size were configured based on 4 and 8 parallel nodes. A total of 20 runs was executed for every single experiment of 3000 generations each.

The rest of the paper is organized as follows: Section 2 presents the related works. The third section presents the sequential genetic algorithms. Then, the proposed approach is described in Section 4. Afterward, we present the experiments, findings, and discussions in Section 5. Finally, Section 6 concludes the paper.

II. Related Works

In the last decade, there has been an increasing amount of literature on parallelizing genetic algorithms using MapReduce framework. The first work was an extension of MapReduce called MRPGA (MapReduce for Parallel GAs). Jin et al., [17] claimed that GAs cannot be directly expressed by MapReduce. They extended the original MapReduce by adding a second reduce phase at the end of each iteration to perform a global selection. The mapper nodes evaluate the fitness function. A local reducer for selecting the local optimum individuals and a second reducer produces the global optimum individuals as final results. Verma et al. [18] identified several shortcomings in the previous approach. Firstly, the mapper node performs the evaluation and the ReducReduce does the local and global selection, the bulk of the work—crossover, mutation and the convergence criteria are carried out by a single container. Hence, their approach decreased the scalability due to the sequential part of the coordinator. Secondly, mapper, reducer, and final reducer emitted “default key” with the value 1. In this respect, they changed the MapReduce model, and they did not apply any standards of the model whether grouping by keys or the shuffling.

Verma et al., [18] proposed a GA based on the traditional MapReduce model to solve ONEMAX problem. They considered one MapReduce job for each GA iteration. The Mapper nodes calculate the fitness values. Then, the Reducers implement selection and crossover operations. The default partitioner was overridden by a random partitioner in order to shuffle individuals randomly across different reducers to avoid overloading the Reducers. They confirmed that the GA can scale on multiple nodes with large population size. However, their model had a big IO footprint because the full population is saved to HDFS after each generation. Hence, big performance degradation was caused [16].

Huang and Lin [10] implemented a MapReduce framework to scale up the population for solving the Job Shop Scheduling Problem (JSSP) using GA. The authors used a large population size (up to 10^7) with fewer generations in order to reduce the overall MapReduce overhead for every generation. This study revealed that the GAs with larger populations were more likely to find good solutions as well as converge with fewer generations. Also, the effect of clusters size is presented, that show the speedup by increasing nodes in the cluster.

In [19], Subasi and Keco developed a Hadoop MapReduce model for parallelizing GA using one MapReduce phase for all generations of the genetic algorithm. Most of the processing was transferred from the reduce phase to map phase. This change reduced the amount of IO footprint because all processing data are kept in a local memory instead of HDFS. However, having a different population for each node leads to a species problem in the algorithm. To solve this problem, Enomoto et al., [20] applied migration strategy to improve population diversity in parallelization a GA using MapReduce, by exchanging individuals among subpopulations during the Shuffle phase. They utilized an ID for each island as a key to assign individuals to their sub-populations. Furthermore, they suggested a method to reduce unnecessary network IO in Shuffle tasks by reducing the number of individuals during migrations. This method eliminated half of the worst individuals in each sub-population after completing the map tasks (after GA- convergence). To maintain search efficiency, a number of individuals are recovered and created by applying a mutation operator at the beginning of each map phase. The results showed a significant improvement in the solution quality and execution time. In [21] the authors proposed a MapReduce hybrid genetic algorithm approach to solve the Time-Dependent Vehicle Routing Problem. The island model has been used for parallelizing the algorithm. The migration process has been carried out by changing the key (island ID) with a certain probability. They observed form the experiments that a large-scale problem with hundreds or thousands of nodes can be solved easily by adding more resources without any change in the algorithm implementation which is impossible in a single machine. Although this approach is interesting, it suffers from an overhead due to launching a MapReduce job for each GA iteration.

Apostol et al., [6] proposed new models for two well-known GA implementations, namely island and neighborhood models. They implemented the two models with two methods of handling sub-populations: island model with isolated subpopulation and neighborhood model with overlapping sub-population. The authors tested the algorithms on two optimization problems: the job shop scheduling problem and the traveling salesman problem with an instance of the problem of size 38 cities. The results showed that there was no significant difference between the two models in execution time, but the solution quality was higher for the neighborhood model over the island model. They proved a fact that the correct handling subpopulations formed by MapReduce can significantly improve the obtained results, but their work suffered from IO overhead between Mappers and reducers.

Ra et.al, [2] solved the TSP using GA on Hadoop MapReduce. They used multiple static populations with migration parallelization method. Iterative MapReduce jobs were used to implement Parallel GA. Each generation implemented a single MapReduce job. In the first job, the map tasks created an initial population and wrote them back to the HDFS. Then, the evaluation process started until a maximum number of generations, map tasks read populations from HDFS and sent them to reducers according to their Population Identifier. The reducers applied the GA operators, i.e. rank selection, greedy crossover and mutation with probability 2.1%, the population was evolved for a specific iteration number. All reducers wrote the best individuals of their populations and wrote the new population into HDFS. In order to share the best individuals, the best individuals were written with a different population identifier to migrate them to another sub-population in the next iteration. They measured the performance of their Parallel GA by comparing the sequential version of it (SGA) with the following algorithms-Sequential Constructive crossover, Edge Recombination crossover, and Generalized N-Point crossover. The results showed that the SGA came up with better solutions than other algorithms, but the SCX and SGA took almost the same time when the problem size increased. Moreover, they compared their SGA with MapReduce parallel GA, the MapReduce GA found better solutions. However, the SGA obtained solution faster than MapReduce GA for the small-sized problem because creation time for the map and reduce tasks impacts the solution time. However, when the problem size increased, SGA solution time increased, and MapReduce has almost the same run-time for all problem sizes.

Khalid et al.,[3] proposed a MapReduce framework implementation for GA with a large population using island parallelization technique. TSP was used as a case study to test the algorithm. A single MapReduce job was assigned to each generation. The Map task was responsible for fitness evaluation, crossover and mutation operations whereas the selection and re-population were done in reduce phase. The key represented the fitness of an individual while the value contained the individual itself. Accordingly, the intermediate pairs (key, value) were sorted and grouped according to fitness value. All individuals with the same fitness value were grouped into the same reducer. However, all copies of this individual were sent to an overloaded single reducer. When the GA converges, all the individuals were processed by that single reducer, so the parallelism would decrease as the GA converges and it would take more iterations. Furthermore, a single job was required for each generation which creates an overhead in term of execution time.

Rao and Hegde [22] proposed a novel method to solve TSP using the Sequential constructive crossover (SCX) on Hadoop MapReduce framework to deal with larger problem size. Iterative MapReduce was applied to specify a single job for each generation. The initial population was generated by the master node. Map tasks read the population from the HDFS and calculate the fitness function and then send them to reduce tasks with their population identifier. Afterward, the partitioner shuffled the individuals based on their sub-population identifiers. The remaining GA operators were performed in the Reduce phase. Upon the completion of iteration, all reducers wrote their best individuals and saved the new population into HDFS. An input file with 20 cities was used for the analysis purpose, and a hundred populations were initially created. They used a single-node Hadoop cluster on a single machine. The virtual machine and Cloudera open source Hadoop platform were used to deploy Hadoop. The results showed better performance after the various evolution of the genetic algorithm. However, using a single MapReduce job for each generation increased the overall overhead.

Ferrucci, Salza and Sarro [16] proposed a parallel genetic on Hadoop MapReduce platform based on three models, namely the global, grid and island models, they were used as a benchmark problem, the software engineering problem of configuring the Support Vector Machines (SVM) for inter-release fault prediction. They assessed the effectiveness of these models in terms of execution time, speedup, overhead and computational effort. The results revealed that the island model outperformed the use of Sequential GA and the PGAs based on the global and grid models. Furthermore, the overhead of the HDFS accesses, communication and latency impaired the parallel solutions based on global and grid models when executed on small problem instances. To speed up the execution of tasks, it was useful to reduce datastore operations as it happened with the island model where data store access was limited to the migration period only.

In this paper, we proposed a new MapReduce model to parallelize GAs named an Elite Parallel Genetic Algorithm using MapReduce (EPGA) in order to improve the population diversity. The Elite technique is inspired by the work of [23]. To the best of our knowledge, no literature proposes the Elite migrating based on Hadoop MapReduce to migrate the individuals between the master node and mapper/reducer node during the GA iterations.

III. Sequential Genetic Algorithms

The parallel adaptations are built on the base of the following SGA implementation, which is composed of a sequence of genetic operators repeated generation by generation, as described in Algorithm 1. The algorithm starts by generating randomly an initial population. It uses the tournament selection technique to select parents for the crossover operation. And it uses the inversion mutation operation which consists to swap randomly two nodes of the individual. The SURVIVAL selection is to select fitter individuals. After performing crossover operation survivor selection is used for selecting a next-generation population, as described in Algorithm 1.

ALGORITHM 1 SEQUENTIAL GENETIC ALGORITHM (SGA)

```
1:   population ← INITIALIZATION (populationSize)
2:   for i←1, MaxGenerationNumber do
3:   for individual ∈ population
4:   FITNESSEVALUATION (individual)
5:   elite ← ELITISM (population)
         population ← population− elite
        parent1 ← TOURNAMENTSELECTION (population)
6:   parent2 ← TOURNAMENTSELECTION (population)
7:   child ← ESCX (parent1, parent2)
8:   offspring ← offspring ∪ {child}
9:   for individual ∈ offspring
10:  INVERSIONMUTATION (individual)
11:  for individual ∈ offspring
12:  FITNESSEVALUATION (individual)
         population ← SURVIVALSELECTION (population,
     offspring)
13:  population ← population ∪ elite
```

IV. PARALLEL GENETIC ALGORITHMS USING MAPREDUCE

Island model is a popular and effective parallel genetic algorithm [3]. It reduces the communication overhead which is an eminent drawback in distributed computing and improves the global search ability of evolutionary algorithms [1]. When dealing with island models some aspects need to be considered:

- The migration interval: how often individuals are exchanged.
- The migration rate: the number of migrant individuals between sub-populations.
- The individual is chosen for migration.
- The individual replaced after the new individuals are received [6].

The following subsections describe in detail the algorithms used in this paper and how they are implemented with MapReduce.

A. *Island Parallel Genetic Algorithm using MapReduce (IPGA)*

The parallel genetic algorithm for the island model on MapReduce divides the population into several sub-populations. Each sub-population executed on a node called an island. Each island executes its sub-population a period of iterations independently from the other islands until a migration occurs Fig. 2. This period of consecutive generations before migration is defined as “migration period”. A MapReduce job is needed for each migration period. In this model, the numbers of Mappers and Reducers are coupled, each couple represents as an island. Each island has a specific identifier number.

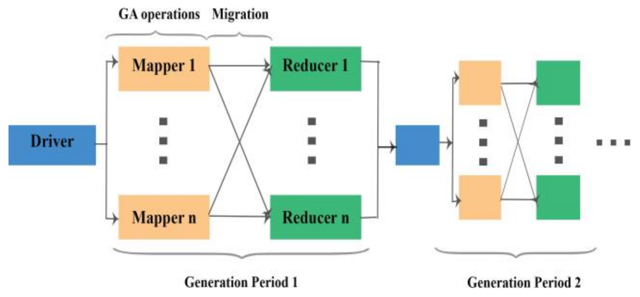


Fig. 2. The Flow of Hadoop MapReduce Implementation for IPGA.

The Mapper: As shown in the algorithm in Algorithm 2, mapper node is used to execute the generation periods, and at the end of the map phase, the migration function is applied. The Mappers output the subpopulation as records in the form (key, value). The key is distention island number, while the value contains the individual and its fitness value. Migration function selects best p % individuals (Migrant Individuals) from island i and migrates them to the next island (i+1) by changing their keys (island distention number).

ALGORITHM 2 MAP PHASE OF IPGA Map(key,value):

```
1: if population not initialized
2:        population ← INITIALIZATION(populationSize)
3:  else
4:       Read population from HDFS
3:  for i←1, GenerationPeriod
4:      for individual ∈ population
5:            FITNESSEVALUATION(individual)
6:       elite ← ELITISM(population)
7:       population ← population – elite
8:       parent1 ← TOURNAMENTSELECTION(population)
9:       parent2 ← TOURNAMENTSELECTION(population)
10:        child    ← ESCX (parent1, parent2)
11:      offspring ← offspring ∪ { child }
12:       for individual ∈ offspring
13:              INVERSIONMUTATION(individual)
14:       for individual ∈ offspring
15:              FITNESSEVALUATION(individual)
16:        population ← SURVIVALSELECTION(population, offspring)
17:        population ← population ∪ elite
18:  end for
19:   for i←1, MigrantIndividuals
20:          selectedIndividual ← GetBestIndividual (population)
21:          NextDestination ← islandNumber % totalNumberOfIslands
22:          remove worstIndividual from population
23:          EMIT  (selectedIndividual, NextDestination)
24:  end for
25:  for individual ∈ population
26:          EMIT (individual, islandNumber)
27:  end for
```

The partitioner sends the individuals to the correspondent island (i.e., the reducer). While the reducer is used only to writes the sub-population received for its correspondent island into HDSF as shown in Algorithm 3.

ALGORITHM 3 REDUCE PHASE OF IPGA Reduce(key, values):

```
1: for individual ∈ population
2:     EMIT (individual, NullWritable.get())
```

B. *Elite Parallel Genetic Algorithm using MapReduce (EPGA)*

Migration Period is realized to propose MapReduce iterations in the previous subsection, but we must consider MapReduce overhead. MapReduce has processing overhead at the start and end times, overhead related to I/O to the data store (i.e., Hadoop Distributed File Systems (HDFS)), and communication overhead in Shuffle tasks. In IPGA, the data store access is limited to the migration phase only. In this study, we aim to reduce MapReduce jobs without decreasing the search efficiency. We propose to apply an Elite migration method in order to reduce the migration frequency without affecting the performance. In this model, the master node (Driver) will read all best individuals from each island at the end of each migration period, sort them by fitness order and share best %p individuals from the top of the list among all islands. The outline of the algorithm is as follows:

1) Each Mapper receives a sub-population to which it applies the GA from the HDFS.

2) Each Mapper performs the GA for a period of generations. An identification number associated with the island (island id) is assigned as a key. Then, a pair of the id and an individual is combined as a (key, value) pair respectively, and outputted to partitioner. If the current period of generations is not the first period, each Mapper reads the Elite individuals received from the master node and replaced them with the %p worst individuals. Elite individuals are added and executed within the current sub-population. Algorithm 4 shows the pseudo-code for the map phase.

3) Each partitioner receives the island id and individual given by the corresponding Map task. The partitioner assigns individuals to the Reducer by referring to the id of the island.

4) Each Reducer receives a subpopulation from its correspondent mapper, and selects best %p individuals, writes them into a separated file to HDFS. Also, outputs all other individuals to HDFS. Algorithm 5 shows the pseudo-code for the reduce phase.

5) If the maximum generation number not exceeded, the Master node reads the best individuals of all islands, sort them and selects best %p individuals and launches the next job.

6) If the maximum generation is achieved, then this process returns the global optimum individual and terminates. Otherwise, it repeats steps 1 to 5.

The flow of EPGA using MapReduce approach is shown in Fig. 3.

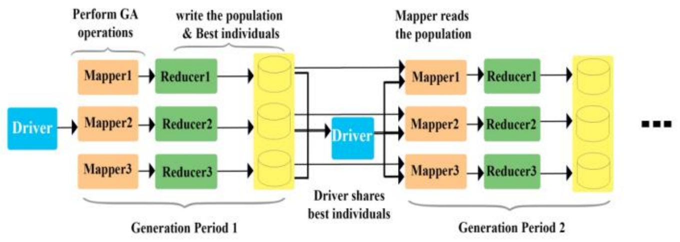


Fig. 3. The Flow of Hadoop MapReduce Implementation for EPGA.

ALGORITHM 4 MAP PHASE OF A GENERATION PERIOD OF EPGA
Map(key, value):

```
1: if population not initialized
2:      population ← INITIALIZATION(populationSize)
3: else
4:      Read population from HDFS
4:      Read EliteIndividuals list from Configuration
2:      Add EliteIndividuals to population
3: for i←1, MigrationPeriod
4:     for individual ∈ population
5:            FITNESSEVALUATION(individual)
6:      elite ← ELITISM(population)
7:      population ← population – elite
8:      parent1 ← TOURNAMENTSELECTION(population)
9:      parent2 ← TOURNAMENTSELECTION(population)
10:      child    ← ESCX (parent1, parent2)
11:     offspring ← offspring ∪ { child }
12:     for individual ∈ offspring
13:            INVERSIONMUTATION(individual)
14:     for individual ∈ offspring
15:            FITNESSEVALUATION(individual)
16:      population ← SURVIVALSELECTION(population, offspring)
17:      population ← population ∪ elite
18: end for
25: for individual ∈ population
26:        EMIT (individual, islandNumber)
27: end for
```

ALGORITHM 5 REDUCE PHASE OF EPGA
Reduce(key, values):

```
1: sort population
2: select best individuals.
3: for individual ∈ population
4:    EMIT (individual, NullWritable.get())
5: write BestIndividuals to HDFS
```

V. Experiments and Results

The experimental evaluation of the proposed MapReduce algorithm is performed on the famous TSP problem. We conducted our experiments on the TSP Data sets provided by Andre Rohe [24]. The problems range in size from 131 cities up to 744,710 cities. Thus, we retained five datasets for a total 10 releases: ft70 (n=70, where n is the problem size)[25], xqf131 (n= 131), xqg237 (n= 237), bcl380 (n= 380), and rbu737 (n= 737). We chose these datasets because they are representing different degrees of the computational load for the fitness evaluation: small (ft70), medium (xqf131, xqg237, bcl381) and large (rub737).

We executed the two PGAs on two different cluster configurations (i.e., C4 and C8) characterized by a different number of nodes. For each PGA model (IPGA and EPGA), dataset (ft70, xqf131, xqg237, bcl380, and rbu737), population size (500, 1000, 2000, 5000 and 10000), and cluster configuration (4 nodes, and 8 nodes), we executed 20 runs. Thus, we executed a total of 2500 runs consisting of 2 × 5 × 5 × 2 × 20 = 2000 runs for PGAs and 5 × 5 × 20 = 500 for SGA.

The experiments are performed in an environment employing a private cloud platform of nine machines which compose the Hadoop cluster. We used a private Hadoop Cluster available at Computer Science department, Al-Imam Muhammad Ibn Saud Islamic University. All nodes have the same configuration to run a fair experiment as shown in Table I.

Table II summarized two different types of Hadoop clusters used in our experiments. SGA was executed on a single node, while for PGAs we exploited C4 and C8 clusters.

We employed the following settings for both SGA and PGAs:

1) Population varies in size 500, 1000, 2000, 5000 and 10000.
2) 3000 generations.
3) The elitism of 1 individual.
4) Tournament Selection for parent selection of size Enhanced Sequential Constructive crossover operator (ESCX) [14], with probability 1.
5) Inversion Mutation, with a probability of 0.5.
6) Survival Selection.

For the PGAs, we used the number of islands equal to the cluster size. We tuned the number of migrant individuals to best 10% of sub-population per island, and the migration period to 3 periods. The performance of the PGAs is measured with respect to execution time, solution quality, speedup, and overhead.

A. Execution Time

The execution time was measured in milliseconds (ms) using the system clock. We compared the computation time achieved by executing all generations of SGA and PGA. Fig. 4 shows the achieved execution times obtained over 20 runs for each dataset and with different population sizes (500, 1k, 2k, 5k, and 10k). We can observe that the PGAs (IPGA and EPGA) outperforms the SGA for the large datasets xqg237, bcl380 and rbu737 (Fig. 4(c), (d), and (e)), regardless of the number of parallel nodes used. And for the ft70 and xqf131 datasets, PGAs are better only when executed using more than 1k population (a and b, Fig. 4). This can be explained by the fact that, for small instance problems, the overhead due to communication between nodes is higher than the computational time. However, when the problem size or/and population size increases, the SGA execution time increases dramatically. We can observe from Fig. 4 that the execution time of the two PGA models using a C8 cluster, is better than using C4 cluster on all the datasets, so the use of more nodes allowed to further reduce the execution time. And the execution time of IPGA and EPGA is very similar time.

B. SpeedUp

The speedup is the ratio of the sequential execution time to the parallel execution time [16]. The speedup is calculated based on the following equation:

$$\text{Speedup} = \frac{\text{SGA time}}{\text{PGA time}} \qquad (1)$$

We compared the achieved speedup with respect to the ideal speedup. The ideal speedup is equal to the number of parallel nodes and corresponds to the situation when the SGA execution time is split among multiple nodes. Fig. 5 shows the speedup obtained by PGAs for all considered datasets. Both PGAs speed up the execution time with respect to SGA over all datasets of mean 7.2 × times by exploiting IPGA usingC8 cluster, 3.9 × times by exploiting IPGA using C4cluster. And 6.7 × times by exploiting EPGA using C8 cluster, 3.8× times by exploiting EPGA using C4 cluster. It is clear from the figure that, both PGAs tend to the ideal speedup value.

C. Solution Quality

The solution quality of the TSP problem was measured by calculating the error (Error%) of approximation of the best individual's fitness value and the TSPLIB optimum found on the website [24]. The error of the best path found with regard to the optimal tour in the TSPLIB is calculated as the given formula:

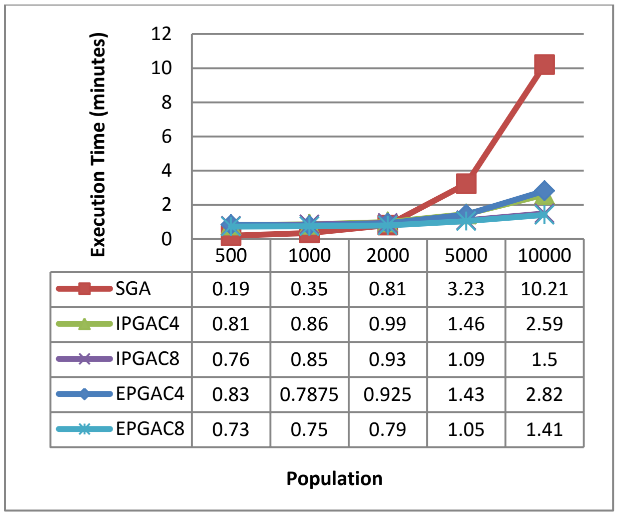
$$\text{Error (\%)} = \frac{\text{Best Solution} - \text{OptimumTSBLIB}}{\text{OptimumTSBLIB}} \times 100$$

TABLE I. MACHINES CONFIGURATION

Feature	Value
Architecture	64 bit
CPUs	4 cores
RAM	8 GB
Storage	500 GB
Operating System	Linux

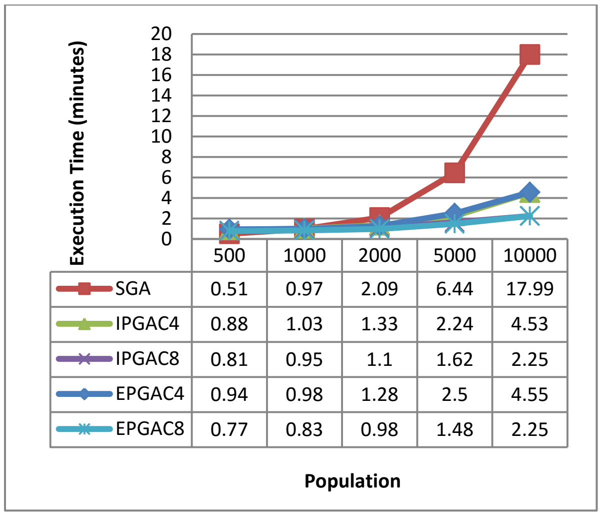
TABLE II. CLUSTER CONFIGURATION EXPLOITED BY PGAS.

Name	Master nodes	Slave nodes	Total nodes
C4	1	4	5
C8	1	8	9



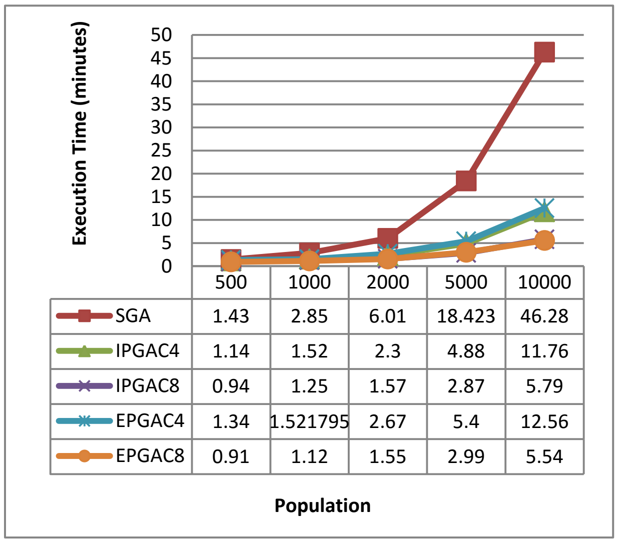
	500	1000	2000	5000	10000
SGA	0.19	0.35	0.81	3.23	10.21
IPGAC4	0.81	0.86	0.99	1.46	2.59
IPGAC8	0.76	0.85	0.93	1.09	1.5
EPGAC4	0.83	0.7875	0.925	1.43	2.82
EPGAC8	0.73	0.75	0.79	1.05	1.41

(a) Fv70



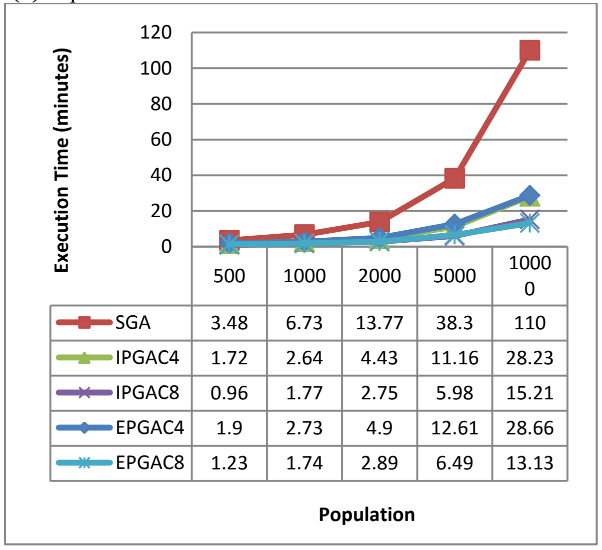
	500	1000	2000	5000	10000
SGA	0.51	0.97	2.09	6.44	17.99
IPGAC4	0.88	1.03	1.33	2.24	4.53
IPGAC8	0.81	0.95	1.1	1.62	2.25
EPGAC4	0.94	0.98	1.28	2.5	4.55
EPGAC8	0.77	0.83	0.98	1.48	2.25

(b) Xqf131



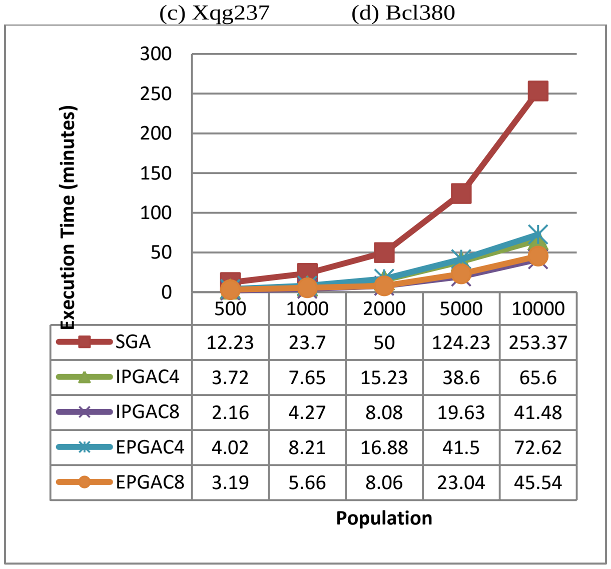
	500	1000	2000	5000	10000
SGA	1.43	2.85	6.01	18.423	46.28
IPGAC4	1.14	1.52	2.3	4.88	11.76
IPGAC8	0.94	1.25	1.57	2.87	5.79
EPGAC4	1.34	1.521795	2.67	5.4	12.56
EPGAC8	0.91	1.12	1.55	2.99	5.54

(c) Xqg237



	500	1000	2000	5000	10000
SGA	3.48	6.73	13.77	38.3	110
IPGAC4	1.72	2.64	4.43	11.16	28.23
IPGAC8	0.96	1.77	2.75	5.98	15.21
EPGAC4	1.9	2.73	4.9	12.61	28.66
EPGAC8	1.23	1.74	2.89	6.49	13.13

(d) Bcl380



	500	1000	2000	5000	10000
SGA	12.23	23.7	50	124.23	253.37
IPGAC4	3.72	7.65	15.23	38.6	65.6
IPGAC8	2.16	4.27	8.08	19.63	41.48
EPGAC4	4.02	8.21	16.88	41.5	72.62
EPGAC8	3.19	5.66	8.06	23.04	45.54

(e) Rbu737

Fig. 4. Execution Times Achieved by SGA and PGAs on the five TSP Datasets.

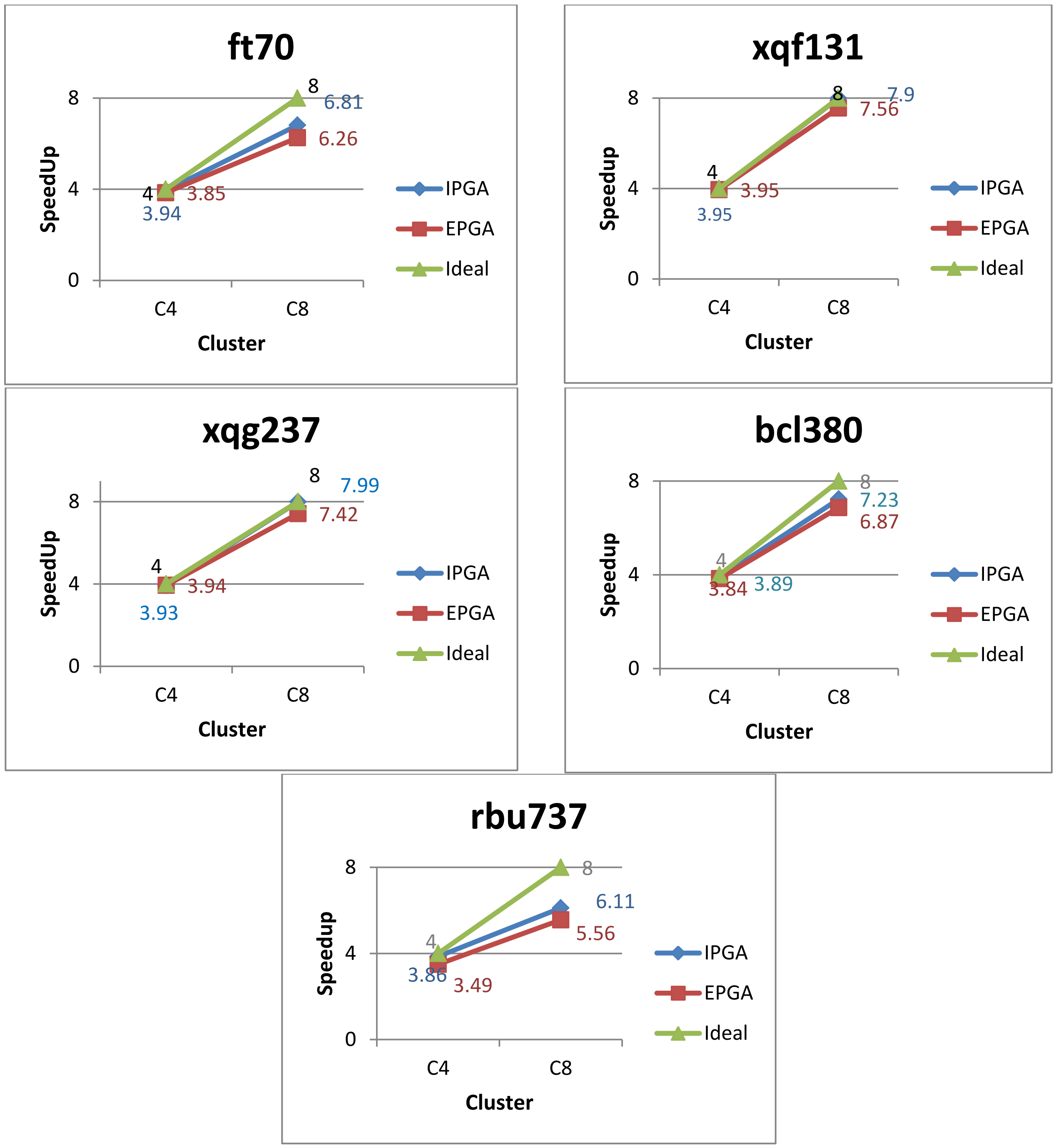
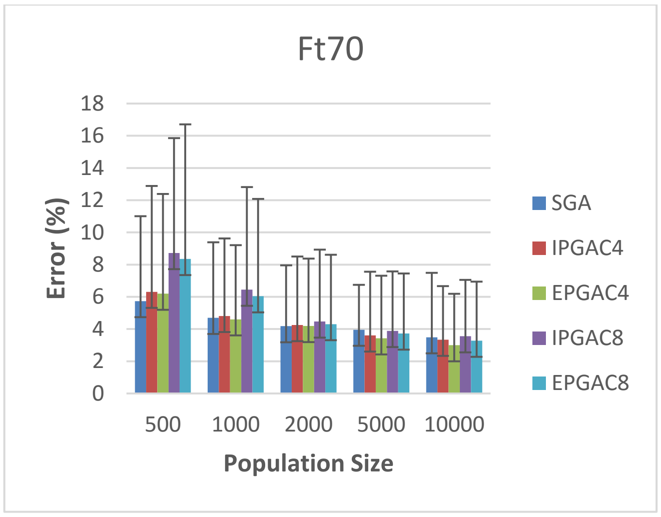
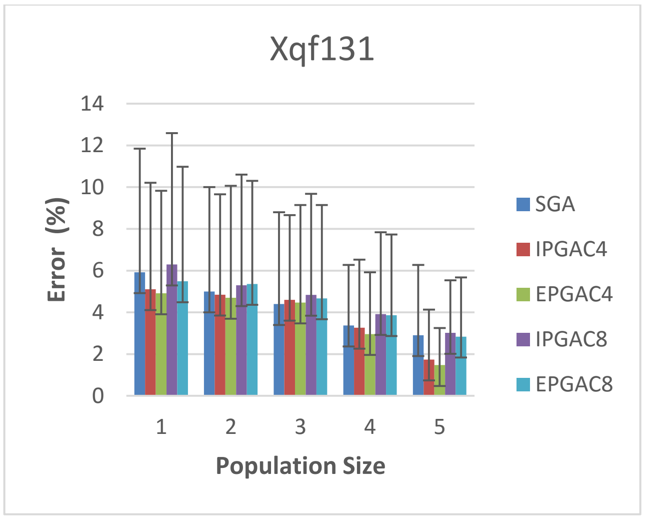


Fig. 5. IPGA and EPGA Speedup Per Dataset with Population Size 10000.

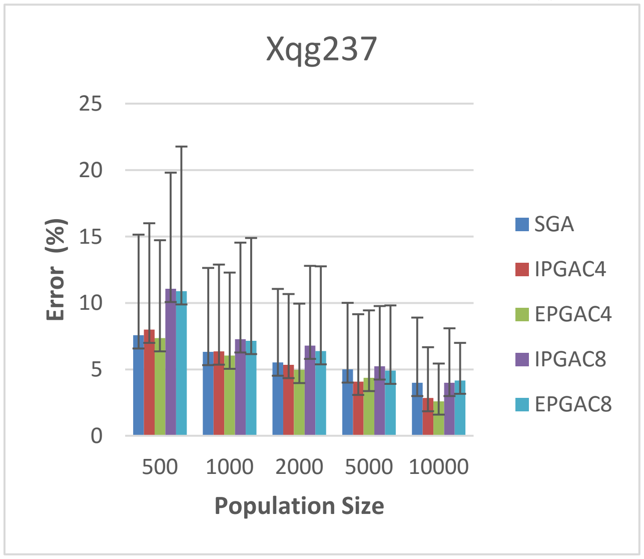
Fig. 6 reports the percentage of average solution accuracy achieved by SGA and PGAs on 20 runs on each algorithm and TSP dataset. We can observe from the figure that, the EPGA on 4-node cluster outperforms the SGA and IPGA for all considered datasets. Even if PGA obtain 'similar' solution accuracy as the SGA, the PGA outperforms the SGA in term of required time to get the same solution with different population size. We can observe also, as the population size increases, the PGAs performance improves rapidly in against to SGA. In Fig. 6(a), the EPGA in C4 cluster obtains better results and outperforms SGA and IPGA when the population size increases to 1k and above. The SGA obtains better results with small population sizes (i.e., 500 and 1000). This can be explained by the fact that the number of individuals on each island will be less than the original population (in case of 500, each island will have 125 and 63 individuals in C4 and C8 respectively), therefore the population diversity is also less than the original GA. This degrades search performance. In (b), the IPGA and EPGA in C4 cluster, outperform the SGA in all population sizes. The EPGA in C8 cluster outperforms the SGA after the population size increased to 2000 individuals, while the IPGA in C8 cluster remains the worst.



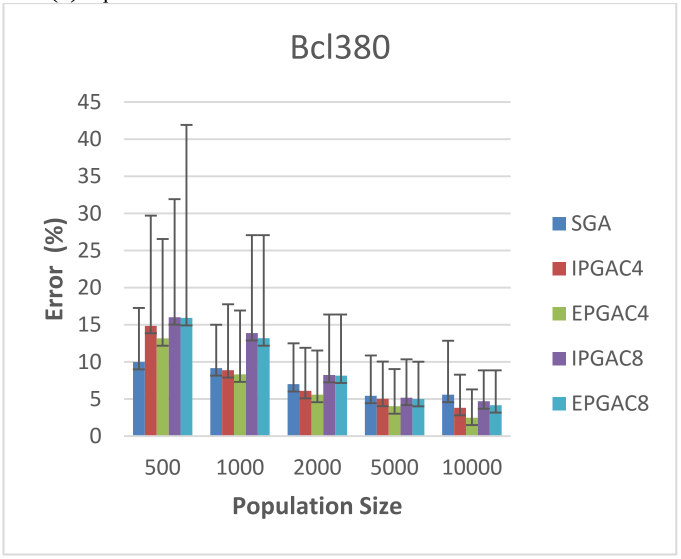
(a) ft70



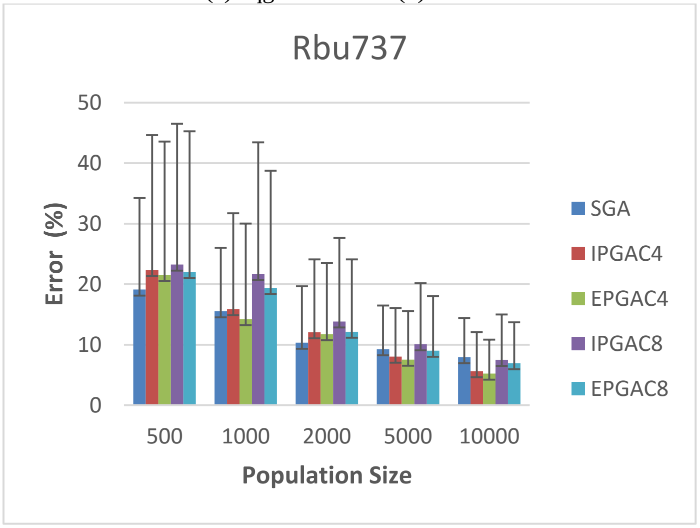
(b) xqf131



(c) xqg237



(d) bcl380



(e) rbu737

Fig. 6. The Percentage of the Average Solution Quality Achieved by SGA and PGAs on the Five TSBLIB Instances.

In the xqg237 dataset, also, the EPGA obtains better results than all other models in all population sizes. In (d) and (e), we can observe that the SGA outperforms the PGA models in term of solution quality in the population sizes between 500 and 2000, but at the same time it takes very long execution time in against to PGAs. The PGAs obtained 'similar' and 'better' results faster than SGA but in larger population size. We can say that scaling up PGAs with large population tend to find better performance over the SGA within a reasonable time.

D. Overhead

The overhead time is the additional time other than the computation, due to communication and Hadoop platform tasks. The overhead is the reason that prevents the PGAs to have a speedup near the ideal on Hadoop platform. To measure the overhead for each PGA generation, we assign to each MapReduce job an initialization, computation, and finalization times for both Map and Reduce phases [16]. Fig. 7 shows the time measurement method for a MapReduce job.

- Map Initialization time is the time required to let the first Mapper start its computation.
- Map Finalization and Reducer Initialization time which is the time of the last mapper and the first reducer. Both of them are measured in the same way.
- Reducer Finalization time is the time of the last ending reducer.

In general, the overhead time in Hadoop corresponds to the sum of the overhead times of multiple jobs. Fig. 8 shows the mean computation and overhead times for each PGA on two datasets xqg237 and rbu737 in population size 2k. In Hadoop MapReduce, the overhead can be calculated using the sum of the overhead times of multiple jobs [16]. As we can see from the figure that, the overhead time for the IPGA and EPGA is the same and light in term of overhead time. That because we reduced the number of launched jobs during the PGAs execution time in order to control the overhead of HDFS accesses, which is limited to the migration phase only. From Fig. 8, we can observe that the overhead time is almost constant over the different jobs, and the map initialization phase takes longer time than the reducer initialization, that because in reducer phase nodes are already prepared to start reducer task when the Mappers are finishing. We also observed the overhead time independent of the dataset size.

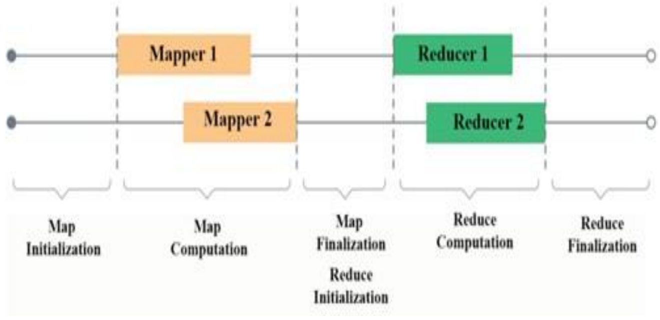
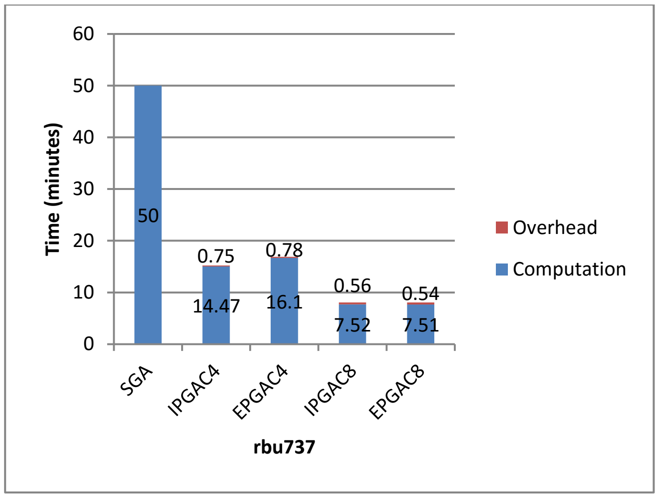
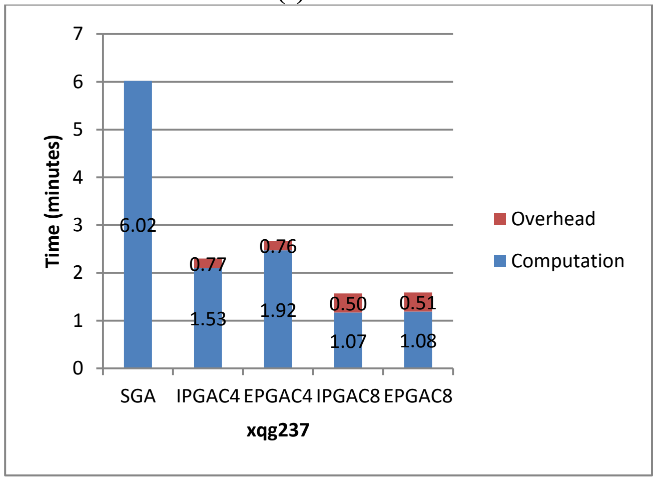


Fig. 7. The Time Measurement Method for Multiple Nodes.



(a) rbu737.



(b) xqg237.

Fig. 8. The Computation and Overhead Times for Each PGAs Model.

VI. Conclusion

In this paper, we designed a Hadoop MapReduce platform which is a general framework for implementing parallel genetic algorithms based on two models; the island model and the elite model. The traveling salesman problem is used as a benchmark in the experimental evaluation of those two Parallel Genetic Algorithms (PGAs).

We empirically assessed the effectiveness of those two PGA models in terms of execution time, speed up, overhead and solution quality by using five datasets form TSPLIB [24]. The datasets were chosen considering their different sizes in order to vary the execution times of the GAs. Additionally, varying population size and the cluster size were configured based on 4 and 8 parallel nodes.

We found that the PGAs find better solutions faster than Sequential Genetic Algorithm (SGA) when the problem size increases as well as when the population size increases. The EPGA outperforms the IPGA in term of the solution quality in a similar time for all the considered datasets and clusters.

We observed the effect of large population size, large populations (5k and 10k individuals) tend to find better solutions and need fewer generation periods to obtain good results which reduce the overall Hadoop overhead.

We also found that increasing the number of nodes in a cluster reduces the execution time. The use of the PGA models enabled to speed up the average execution time overall datasets with respect to SGA and tend to the ideal speedup value.

The overhead of HDFS access and communication is reduced in the PGAs since the number of operations performed on the datastore is limited to the migration phase only. However, Hadoop overhead may impair PGA solutions when executed on small problem instances.

We aim as a future work plan at comparing the performance of our model with an iterative MapReduce framework such as Haloop and Spark. Also, we aim to evaluate both parallel models by applying them to a challenging software engineering problem.

References

[1] Y. J. Gong et al., “Distributed evolutionary algorithms and their models: A survey of the state-of-the-art,” Appl. Soft Comput. J., vol. 34, no. 2013, pp. 286–300, 2015.

[2] H. Ra and N. Erdoğan, “Parallel Genetic Algorithm to Solve Traveling Salesman Problem on MapReduce Framework using Hadoop Cluster,” Int. J. Soft Comput. Softw. Eng. [JSCSE], vol. 3, no. 3, pp. 380–386, 2013.

[3] N. E. A. Khalid, A. F. A. Fadzil, and M. Manaf, “Adapting MapReduce framework for genetic algorithm with large population,” in Proceedings - 2013 IEEE Conference on Systems, Process and Control, ICSPC 2013, 2013, no. December, pp. 36–41.

[4] J. Dean and S. Ghemawat, “MapReduce,” Commun. ACM, vol. 51, no. 1, p. 107, Jan. 2008.

[5] P. Sachar and V. Khullar, “Genetic Algorithm Using MapReduce-A Critical Review,” i-manager’s J. Cloud Comput., vol. 2, no. 4, pp. 35–42, 2015.

[6] E. Apostol, I. Băluţă, A. Gorgoi, and V. Cristea, “A Parallel Genetic Algorithm Framework for Cloud Computing Applications,” Pop F., Potop-Butucaru M. (eds). ARMS-CC 2014. Lect. Notes Comput. Sci. vol 8907. Springer, Cham, vol. 8907, pp. 113–127, 2014.

[7] W. Tom, Hadoop: The Definitive Guide. FOURTH EDITION The Definitive Guide STORAGE AND ANALYSIS AT INTERNET SCALE, 4th ed. O’Reilly Media, 2015.

[8] R. Li, H. Hu, H. Li, Y. Wu, and J. Yang, “MapReduce Parallel Programming Model: A State-of-the-Art Survey,” Springer, Int. J. Parallel Program., vol. 44, no. 4, pp. 832–866, 2016.

[9] F. Imeson and S. L. Smith, “A language for robot path planning in discrete environments: The TSP with Boolean satisfiability constraints,” in Proceedings - IEEE International Conference on Robotics and Automation, 2014, pp. 5772–5777.

[10] D. W. Huang and J. Lin, “Scaling populations of a genetic algorithm for job shop scheduling problems using mapreduce,” in Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010, 2010, pp. 780–785.

[11] K. Miclaus, R. Pratt, and M. Galati, “The Traveling Salesman Traverses the Genome: Using SAS® Optimization in JMP® Genomics to build Genetic Maps,” 2012.

[12] J. Singh and A. Solanki, “An Improved Genetic Algorithm on MapReduce Framework Using Hadoop Cluster for DNA Sequencing,” Int. J. Adv. Res. Comput. Sci. Softw. Eng., vol. 5, no. 6, pp. 1238–1244, 2015.

[13] N. Bansal, A. Blum, S. Chawla, and A. Meyerson, “Approximation algorithms for deadline-TSP and vehicle routing with time-windows,” in Proceedings of the thirtysixth annual ACM symposium on Theory of computing, 2004, pp. 166–174.

[14] H. Bennaceur and E. Alanzi, “Genetic Algorithm For The Travelling Salesman Problem using Enhanced Sequential Constructive Crossover Operator,” Int. J. Comput. Sci. Secur., vol. 11, no. 3, p. 42, 2017.

[15] Z. H. Ahmed, “Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator,” Int. J. Biometrics Bioinforma., vol. 3, no. 6, pp. 96–105, 2010.

[16] F. Ferrucci, P. Salza, and F. Sarro, “Using Hadoop MapReduce for Parallel Genetic Algorithms: A Comparison of the Global, Grid and Island Models,” no. x, pp. 1–33, 2017.

[17] C. Jin, C. Vecchiola, and R. Buyya, “MRPGA: an extension of MapReduce for parallelizing Genetic Algorithms,” in Proceedings - 4th IEEE International Conference on eScience, eScience 2008, 2008, pp. 214–221.

[18] A. Verma, X. Llorà, D. E. Goldberg, and R. H. Campbell, “Scaling Genetic Algorithms using MapReduce,” in In International Conference on Intelligent Systems Design and Applications (ISDA), 2009, pp. 13–18.

[19] A. Subasi and D. Keco, “Parallelization of genetic algorithms using Hadoop Map / Reduce,” SouthEast Eur. J. Soft Comput., no. June, pp. 127–134, 2012.

[20] T. Enomoto and M. Kimura, “Improving Population Diversity in Parallelization of a Real-Coded Genetic Algorithm Using MapReduce,” in Scientific Cooperations International Workshops on Electrical and Computer Engineering Subfields, 2014, no. August, pp. 234–239.

[21] R. Kondekar, A. Gupta, G. Saluja, R. Maru, A. Rokde, and P. Deshpande, “A MapReduce based hybrid genetic algorithm using island approach for solving time dependent vehicle routing problem,” in 2012 International Conference on Computer and Information Science (ICCIS), 2012, vol. 1, no. 2003, pp. 263–269.

[22] A. Rao, K. Hegde, K. Rao, IAnitha and Hegde, A. Rao, and S. K. Hegde, “Literature Survey On Travelling Salesman Problem Using Genetic Algorithms,” Int. J. Adv. Res. Eduation Technol., vol. 2, no. 1, p. 4, 2015.

[23] L. N. G. Sanchez, J. J. T. Armenta, and V. H. D. Ramırez, “Parallel Genetic Algorithms on a GPU to Solve the Travelling Salesman,” Difu100ci@, vol. 8, no. 2, pp. 79–85, 2014.

[24] A. Rohe, “VLSI Data Sets.” [Online]. Available: http://www.math.uwaterloo.ca/tsp/vlsi/index.html. [Accessed: 17-Nov-2018].

[25] G. Reinelt, “TSPLIB.” [Online]. Available: https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. [Accessed: 17-Nov-2018].