Scalable Distributed Genetic Algorithm using Apache Spark (S-GA)

Fahad Maqbool[1] Saad Razzaq[1] Jens Lehmann[2] and Hajira Jabeen[2]

[1] University of Sargodha, Pakistan fahad.maqbool@uos.edu.pk, saad.razzaq@uos.edu.pk
[2] Bonn University, Germany, jabeen@cs.uni-bonn.de, lehmann@cs.uni-bonn.de

Abstract. In the era of big data with real-time data acquisition tools, the solutions to large-scale optimization problems are strongly desired. Genetic Algorithms are an efficient optimization algorithms that have been successfully applied to solve a multitude of optimization problems. The growing need for large-scale optimization and inherent parallel evolutionary nature of the algorithm, calls for exploring them for parallel processing using existing parallel, in-memory, computing frameworks like Apache Spark. In this paper, we present a framework for Scalable Genetic Algorithms on Apache Spark (S-GA). The S-GA makes liberal use of Sparks RDDs for parallel, distributed processing. We have tested S-GA on several benchmark functions for large-scale continuous optimization ranging up to 2000 dimensions, 10,000 population, and 40 million generations. We have tested and compared our results with the Sequential Genetic Algorithm (SeqGA) and the results of our proposed parallel model have been found better, in addition to scaling to large-scale optimization problems.

Keywords: Apache Spark · Parallel Genetic Algorithms · Function Optimization · Hadoop Map Reduce.

1 INTRODUCTION

Owing to the inherently decentralized nature of Genetic Algorithms (GA), a multitude of variants of parallel GA (PGA) have been introduced to date [1, 2]. However, their application has been limited to moderately sized optimization problems and focus mostly remained on speeding up the performance of otherwise time-consuming and inherently iterative algorithms. To deal with large-scale optimization problems, a number of architectures ranging from; multi-core systems to standalone clusters, making use of distributed storage file system or distributed processing frameworks like Apache Hadoop, have been proposed to achieve scalability in PGA [3–6].Hadoop Map Reduce [7] is a reliable, scalable and fault tolerant framework. However, Hadoop requires writing data to HDFS after each iteration to achieve this fault tolerance. In case of CPU bound iterative processing, e.g. Genetic Algorithms, this I/O overhead is undesirable and substantially dominates the processing time. PGA has been explored for numerous interesting applications like software fault prediction [8], test suite generation [9], sensor placement [10] assignment scheduling [11–13], dynamic optimization [23], adapting offspring population size and number of islands [24]. Most of the above-mentioned works have used distributed frameworks and they evaluate the effectiveness of PGAs in term of execution time, computation effort, solution quality, and compare the results with SeqGA. However, they have experimented with simpler problems which can be solved using limited population size and less number of generations overlooking the scalability that can be achieved by these frameworks to solve large-scale optimization problems. Apache Spark is an open source distributed cluster computing framework that has gained fame in recent years. It performs well for large-scale distributed operations due to its faster in-memory operations. It performs well for iterative operations besides its high memory requirement [14].Contrary to Hadoop, Spark keeps data in memory and uses lineage graphs to achieve resilience and fault tolerance. This makes the computing faster and eliminates the I/O overhead incurred in case of map-reduce. Spark provides API for SQL like processing, stream processing using concepts of mini-batches, iterative machine learning algorithms and Graphs processing library [15].Sparks efficient data processing has proven to 100 times faster for in-memory operations and 10 times faster for disk operations compared to Hadoop Map Reduce. Moreover, it is designed to have a high frequency of in-memory operations as compared to disk operations. A comparison of Hadoop Map Reduce based parallel implementations of the three main parallel models of GA i.e. global single-population master-slave GAs (global model), single-population fine-grained (grid model), multiple-population coarse-grained (island model) is discussed by F. Ferrucci et.al [8]. Their study reveals that overheads of Hadoop distributed file system make Global and Grid models less attractive as compared to Island model for parallelizing GA in term of HDFS access, communication and execution time, since Island model performs less HDFS operations, resulting in optimized resource utilization and efficient execution time. However, they reported experimental results of Global, Grid, and Island models on 200 population size, with a run of maximum 300 generations on smaller problems with a limited number of dimensions (up to 18).They concluded that distributed frameworks provide efficient support for data distribution, parallel processing, and memory management. But they incur the overhead of communication delays. In this paper, we propose a scalable GA, S-GA. The proposed algorithm is evaluated for scalability with the SeqGA and found to be time efficient. S-GA tries to reduce the communication between master and worker nodes of Apache Spark for efficient resource utilization. In the traditional Island model, the communication among different islands is directly proportional to the population and solution size. This directly translates to communication overhead in large-scale optimization problems making this model less suitable for scalability. In S-GA, the communication is independent of the population size and is limited by the migration rate and problem size, hence, reducing a significant amount of data transfer between parallel computations making it a suitable choice for scalable problems. S-GA is available as open source, freely available software. The paper is structured as follows: In section 2, related work is discussed. Background knowledge section is explored in section 3. The proposed S-GA is detailed in section 4. Experiments and evaluation are shown in section 5. Conclusion and future work are presented in section 6.

2 RELATED WORK

The island model was initially proposed by D. Whitley et.al [16]. It was expected that Island model would outperform SeqGA, because of the diversity of chromosomes and migration of individuals among several islands. However, results revealed that Island model may perform better only if increasing the population size does not help in solving the problem and migration transfer information among sub-populations is handled carefully. Island model uses a large population divided among different subpopulations, preserving genetic diversity among these subpopulations, while SeqGA uses single large population pool. Map Reduce framework is used to make GA scalable [17]. Experiments performed on OneMax problem addressed the scalability and convergence in terms of decreased time per iteration. This was achieved by increasing the number of resources while keeping problem size fixed. Convergence and scalability were discussed up to 105 and 108 variable problems for counting and maximizing 1s in a bit string. D. Keo and A. Subasi [18] discussed PGA using Hadoop map-reduce framework. Their focus was to improve final solution quality and cloud resource utilization. They obtained improved performance in term of convergence but couldnt improved in term of solution quality. Edgar et.al [19] proposed the speed up to multi-objective optimization using EA. Experiments were performed on various objective functions including Simple Evolutionary Multi-objective Optimizer (SEMO), global SEMO for OneMinMax, and Lots bi-objective function. The speedup was gained by optimizing in fewer generations compared to other classical parent selection methods with diversity based parent selection. The technique focused on individuals having high diversity but located in poorly explored areas. Wanru et.al [20] contributed theoretical aspect of maximizing the diversity of a population in EA that contains several high-quality solutions. They worked on OneMax and Leading One's problems. Results revealed that algorithm efficiently maximizes the diversity of a population. Technique lacks in the experimental framework and didnt address how diversity in solutions helps in large-scale optimization problems. GA and PGA are widely used as an application area. Amaro B. et.al [21] applied parallel biased random-key GA with multiple populations on irregular strip packing problem. In this problem set, items of variable length and fixed width should be placed in a container. Their focus is to reduce the area required to fulfill the given demand. For an efficient layout scheme, they have used the collision-free region as a partition method along with a meta-heuristic and a placement algorithm. Computations performed on datasets show competitive results with other relevant techniques. F. Gronwald et.al [22] presented determining a source of air dispersion. A concentration profile was compiled by considering the readings from different points in an area. They used Backward

Parallel Genetic Algorithm (BPGA) that utilizes multiple guesses in a generation, and the best one is determined by a fitness function. This best guess is used in the reproduction of next generation. After several generations, BPGA finds the location and amount of pollutant source in air. Experimental results were not compared with any other model. PGA using Spark framework was proposed for the pairwise test suite generation. Parallelization was achieved in term of fitness evaluation and genetic operations. Results were compared with SeqGA on synthetic and real-world datasets. PGA performs better than SeqGA in term of test suite size generation [9]. In this paper, we proposed scalable PGAs for large-scale optimization problems using Apache Spark S-GA. Inbuilt features of Apache Spark and independence of S-GA from migration overhead with an increase in population size, makes S-GA scalable. We have compared S-GA results with SeqGA for several different parameters. Details have been discussed in experiment section.

3 BACKGROUND

3.1 Apache Spark

A group at the University of California, Berkeley started Apache Spark Project in 2009 for distributed data processing. Apache Spark provides data sharing abstraction using Resilient Distributed Datasets (RDD). While running on a cluster, the master node is responsible for the creation of RDD while each worker node can process a portion of the distributed RDD. Each RDD is divided into logical partitions and each worker node can process one or more partitions. RDD supports two types of operations: i.) Transformations, ii.) Actions. Transformations are lazy operations that create a new dataset from existing data in RDD. By lazy evaluation we means that transformations are not applied until an action is encountered. The actions are aggregate operations that transfer data from worker node to master node.

RDD is an immutable data structure, once created cant be modified. The only way to update RDD data is to create a new RDD. Creation is always done at the master node and then it is distributed among the worker nodes in the form of logical units called **Partitions** (Pti). Each worker node can contain one or more partitions. Reducing the number of RDD creations massively reduces the communication overhead as each RDD creation results in data communication between master and worker nodes. Creation of a new RDD results in network communication between master and worker nodes. Hence avoiding aforementioned transformation results in significant reduction of communication overhead

3.2 Sequential Genetic Algorithm (SeqGA)

SeqGA also known as Canonical GA is a stochastic search method that is used to find the optimal solution for optimization problem. It consists of a single pool of population (panmixia) and applies stochastic operators (i.e. Selection, Crossover, Mutation, and Survival Selection) to create a new evolved population. For large scale problems, SeqGA may require more computational efforts including more memory and long execution time. Table 1. explains the evolution process of SeqGA.

Table 1. Sequential Genetic Algorithm.

```
$P <-$Generate Initial Population
while ( Stopping Criteria Not Met )
$P <-$ Select Parents ($P$)
$P<-$ Crossover ($P$)
$P <-$ Mutate ($P$)
$P<-$ Survival-Selection ($P\ U\ P$)
end while
```

Parent Selection specifies how individuals will be selected for reproduction. Crossover helps to explore the search space by generating new solutions while mutation exploits the solutions for improvement. Finally, Survival-Selection scheme decides the number of individuals to be selected from parents and offspring for the next generation. For experiments, we have used roulette wheel, uniform, interchange, and weak parent as selection, crossover, mutation, and survival techniques respectively.

3.3 Parallel Genetic Algorithm (PGA)

Generally, there are three main models to parallelize GA i.e. global single-population master-slave GAs (global model), single-population fine-grained (grid model), multiple-population coarse-grained (island model) [1]. The **Global Model** works like SeqGA with one population. The master is responsible for handling the population by applying GA operators while slave manage the fitness evaluation of individuals. In **Grid Model**, GA operators are applied within each subpopulation and each individual is assigned to only one sub-population. This helps in improving the diversity but it has a problem of getting stuck in local optima. This model has high communication overhead due to frequent communication between subpopulations. In **Island, Model**, the population is subdivided into islands/groups.GA operates on these islands independently with the ability to exchange/migrate some of the individuals later. This helps in increasing the diversity of chromosome and avoiding to get stuck in local optima.Mostly, PGA divides a population into multiple sub-populations. Each population independently searches for an optimal solution using stochastic search operators like crossover and mutation. Although this parallelization is intrinsic in nature to GA, this parallelization comes at the cost of significant communication overhead. This overhead is a major hurdle to the ideal speed-up that we may achieve using parallel/distributed techniques. A parallel approach to evolutionary algorithm should make optimal resource utilization and reduce communication overhead to gain speedup. Previously proposed PGA implementations majorly differ how they structure their population. This structure majorly affects PGA execution time. The topology of PGA determines how subpopulations can share their solutions, i.e. a sub-population can send the solution to which other sub-population(s) or a sub-population can receive solution from which other sub-population(s). PGA, when executed using distributed frameworks like Apache Spark is not affected by the topology being used. This is due to the reason that migrant solutions are broadcasted to all the partitions independent of the topology. Based on the topology, each partition/structure will select the solutions to replace its weakest solutions.

4 SCALABLE DISTRIBUTED GENETIC ALGORITHM USING APACHE SPARK (S-GA)

S-GA creates an initial random population of solutions and distributes them on different partitions using RDD. Each partition performs GA operations and fitness evaluation independent of other partitions. For experiments, we have used roulette wheel, uniform, interchange, and weak parent as selection criteria, crossover, mutation, for generating new solutions, and survival selection for deciding offsprings selection for next generation. Based on the selection scheme, individuals from each partition are selected for crossover and mutation. The crossover and mutation rate decides number of individuals to be selected for evolution. In S-GA each partition replaces its weakest solution by the fittest solution broadcasted by each partition. **Migration Rate** (Mr) specifies the number of solutions to be broadcasted in each migration. This rate is inversely proportional to the execution time of PGA. S-GA significantly reduces the communication overhead by creating a single RDD, throughout the execution.

Migration Interval/Gap *(Mi)* defines the number of generations after which S-GA broadcasts fittest individual(s) of each partition to other partitions. Afterwards, S-GA replaces its weakest solutions at each partition by broadcasting the best solutions. S-GA evolves the individual(s)/solution(s) at each partition for Mi number of generations. S-GA avoids transformation of existing Population RDD to a new evolved population RDD after migration interval. Afterwards, S-GA migrates best solution of each partition to all the partitions. Fig. 2 explains the idea with an example. Lets assume value of $Mi = 2$ and fitness function as a sphere (i.e. $f(x_i) = \sum_{i=1}^{n} x_i^2$. Initial at first generation random population is generated. New solutions are explored using crossover and mutation operators.After every 2 generations, best solutions from each partition are migrated to the very next generation. As the solution migrates, the next generation at each partition picks all the migrated solutions and replace them with weakest initial randomly assigned subpopulation. The pseudo code of S-GA

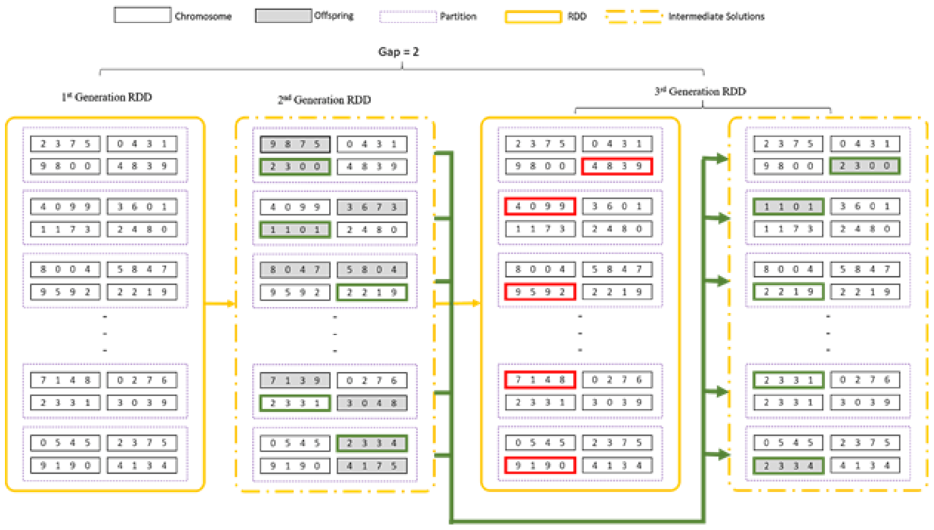

Fig. 1. Figure 2: Evolution process of S-GA

Table 2. Pseudo code of S-GA.

```
N : Population Size
P : Population
Pj: Sub-Population at partition
D: Dimensions
G: Generations
Pti: Partition i, where i =1,, m
Mi:Migration Interval / Gap
f : Fitness Function
Mr: Migration Rate
```

```
1  $P$ Randomly initialize population
2 Distribute $P$ among m partitions
3 $G = 1$
4 while ( stopping$_c riteria_n ot_m et$)
5 at$_e ach_p artition_P ti$
6 $Pi$Pick —$Pi$—-($Mr$*$m$) best solutions from Pi  ($Mr$*$m$) Migrated solutions
7 for k: $1$ to $Mi$
8 $Pi$ Select Parents ($Pi$)
9 $Pi$ Crossover ($Pi$)
10 $Pi$ Mutate ($Pi$)
11  $Pi$ Survival-Selection ($Pi$  $Pi$)
12 end for
13 Migrate S individuals from Pti
14 end at$_e ach_p artition_P ti$
15  $G$= G + Mi
16 end while
```

has been described in Fig. 3. In start variable are defined. Random population is initialized at line (1) and distributed among partitions (m) at line (2). Generation is initialized at line (3). While stopping criteria not reached from line (4-11) best solutions are selected from each partition (Pi) and migrate them to next generation according to migration rate (Mr). New solutions are explored in line (7-9) and best are selected for next generation using survival-selection scheme.

Table 3. Experimental configurations for SeqGA and S-GA.

S-GA Parameters	SeqGA Parameters
Population Size N: 5000, 10000	Population Size N: 5000, 10000
Crossover scheme: Uniform	Crossover scheme: Uniform
Mutation: Interchange	Mutation: Interchange
Survival Selection: Weak Parent	Survival Selection: Weak Parent
Selection Scheme: Roulette Wheel	Selection Scheme: Roulette Wheel
Crossover Probability: 0.3	Crossover Probability: 0.3
Mutation Probability: 0.05	Mutation Probability: 0.05
No of Partitions pti: 12, 24, 36	
Migration Rate Mr: 1, 5, 10	
Migration Interval Mi: 5000, 10000	

5 EXPERIMENTS

5.1 Experimental Setup

Experiments are performed on three nodes cluster: DELL PowerEdge R815, 2x AMD Opteron 6376 (64 Cores), 256 GB .RAM, 3 TB SATA RAID-5 with spark-2.1.0‘ and ‘Scala 2.11.8. The configuration parameters of S-GA and GA are detailed in in Table 1.

5.2 Evaluation Metrics

Execution Time The execution time of SeqGA and S-GA was measured using system clock time. This time was recorded for a maximum of 40 million generations. Table 2. Shows average execution time over 5 runs for each configuration of SeqGA and S-GA. We can observe that execution time of SeqGA has almost doubled when we increase the population size from 5000 to 10,000. However, this is not the case in most of the cases for S-GA, where there is a slight increase in execution time with an increase in population while keeping rest of the configuration constant. This difference in time reduces with an increase in the number of partitions. Migration size defines the total number of migrated individual(s) by all partitions after Mi (Migration Size = Mr * m). Increase in migration size results in increased network overhead and hence execution time. But on the other hand this also helps S-GA to converge in a lesser number of generations. Table 4. lists the execution time of Sphere, Ackley, Rastrigin and Griewank functions for optimization using 2000 dimensions. Results reveal that S-GA outperforms SeqGA while being scalable to larger sized optimization parameters e.g. dimensions, partitions, migration size and migration interval.

Table 4. Experimental Results of S-GA and SeqGA. Dimensions: 2000, Value To Reach (VTR):0.0005, crossover scheme: Uniform, Mutation: Interchange, Replacement Scheme: Weak parent, Selection Scheme: Roulette Wheel, Crossover Probability: 0.3, Mutation Probability: 0.05, Population: 10000, Partitions: 36, Migration Rate: 10, Migration Gap: 10000 tables.

f	S-GA			SeqGA			Speed Up
-	Gen	Time	Error	Gen	Time	Error	-
SPHERE	1.8E+07	5150	0	2089	6451	0	1.25
ACKLEY	1.5E+07	4306	4.44	1695	6716	0	1.56
RASTRIGIN	1.3E+07	3723	0	1503	5775	0	1.55
GRIEWANK	4E+06	1306	0	821	3483	0	3.07

Speed Up Speed up is the ratio of sequential execution time to the parallel execution time. It reflects how much parallel algorithm is faster than a sequential algorithm. Table 4 and 5 reflects speed up for all the cases where SeqGA and S-GA converge to VTR (Value To Reach). VTR defines the threshold for convergence. We have used $\frac{1}{Number of Dimensions}$ as VTR in experimentation. Speed up values in Table 4 and 5 shows that that speedup of S-GA is more if we increase population and partitions size.

6 CONCLUSION

In this paper, we have proposed initial results for Scalable Parallel GA (S-GA) using Apache Spark for large-scale optimization problems. We have compared our results with SeqGA technique. We have tested our technique for Sphere, Ackley, Rastrigin, and Griewank functions that are typical benchmarks for continuous optimization problems. We have used Population up to10000, Dimensions up to2000, Partition Size up to 36, migration rate up to 10, and migration interval gap to 10,000. S-GA has outperformed SeqGA for higher population, partitions, migration rate, and migration interval in term of execution time. In future, we plan to extend S-GA and evaluate different migration and distribution strategies for larger scale and more complex optimization problems.

References

1. G. Luque. , E. Alba.: Parallel genetic algorithms: Theory and real-world applications (2011)

2. D. S. Knysh, V. M. Kureichik.: Parallel genetic algorithms: A survey and problem state:Journal of Computer and Systems Sciences International, vol. 49, no. 4, pp. 579-589 (2010)
3. F. Chvez, F. Fernndez, C. Benavides, D. Lanza, J. Villegas, L. Trujillo, G. Olague and G. Romn.: ECJ+HADOOP An Easy Way to Deploy Massive Runs of Evolutionary Algorithms: European Conference on the Applications of Evolutionary Computation, Porto (2016)
4. L. Di Geronimo, F. Ferrucci, A. Murolo and F. Sarro.: A parallel genetic algorithm based on Hadoop MapReduce for the automatic generation of JUnit test suites: In IEEE International Conference on Software Testing, Verification and Validation (2012)
5. P. Salza, F. Ferrucci, and F. Sarro.: Develop, deploy and execute parallel genetic algorithms in the cloud: Genetic and Evolutionary Computation Conference (GECCO) (2016)
6. L. Zheng, Y. Lu, M. Ding, Y. Shen and M. Guoz, : Architecture-based performance evaluation of genetic algorithms on multi/many-core systems,: IEEE International Conference on Computational Science and Engineering (2011)
7. I. T. Hashem, N. B. Anuar, A. Y. Gani, F. Xia and S. U. Khan, S. U, : MapReduce Review and open challenges, : Scientometrics (2016)
8. F. Ferrucci, S. Pasquale, and S. Federica,: "Using Hadoop MapReduce for Parallel Genetic Algorithm : A Comparison of the Global, Grid and Island Models. :" Evolutionary computation Early Access, pp. 1-33,(2017)
9. R.-Z. Qi, Wang and S.-Y. Li, :A Parallel Genetic Algorithm Based on Spark for Pairwise Test Suite, : Journal of Computer Science and Technology, vol. 31, no. 2, p. 417427, (2016).
10. C. Hu, G. Ren, C. Liu, M. Li and W. Jie, : A Spark-based genetic algorithm for sensor placement in large-scale drinking water distribution systems, : Cluster Computing, The Journal of Networks, Software Tools, and Applications, vol. 20, no. 2, pp. 1089-1099, (2017).
11. D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee, : Efficient hierarchical parallel genetic algorithm using grid computing, : Future Generation Computer Systems, pp. 658-670, (2007).
12. Y. Y. Liu and S. Wang, :A scalable parallel genetic algorithm for the generalized assignment problem, : Parallel computing, pp. 98-119, (2015).
13. A. e. a. Trivedi, :Hybridizing genetic algorithm with differential evolution for solving the unit commitment scheduling problem., : Swarm and Evolutionary Computation, pp. 50-64, (2015).
14. L. Gu and H. Li, : Memory or time Performance evaluation for iterative operation on Hadoop and spark, : High-Performance Computing and Communications: IEEE International Conference on Embedded and Ubiquitous Computing (HPCC EUC), (2013).
15. W. M. Ahmad and S. Jabin, : Big Data: Issues, Challenges, and Techniques in Business Intelligence, : Proceedings of CSI-50th Golden Jubilee Annual Convention, Springer (AISC Series), (2015).
16. D. Whitley, S. Rana, and R. B. Heckendorn, :The island model genetic algorithm: On separability, population size, and convergence, : CIT. Journal of computing and information technology, vol. 7, no. 1, pp. 33-47, (1999)
17. A. Verma, X. L. or'a, D. E. G. erg and R. H. C. ell, : Scaling simple, compact and extended compact genetic algorithms using MapReduce, : Illinois Genetic Algorithms Laboratory (Illinois) report no. 2009001, illegal, Uni of Illinois, Urbana-Champaign, (2009).

18. D. Keo and A. Subasi, : Parallelization of genetic algorithms using Hadoop Map/Reduce, : SouthEast Europe Journal of Soft Computing, vol. 1, no. 2, (2002)
19. E. C. Osuna, W. Gao, F. Neumann and D. Sudholt, : Speeding up evolutionary multi-objective optimization through diversity-based parent selection, : Genetic and Evolutionary Computation Conference, Berlin, Germany, (2017).
20. W. Gao and F. Neumann, : Runtime Analysis of Maximizing Population Diversity in Single-Objective Optimization,: Genetic and Evolutionary Computation Conference, Vancouver, Canada, (2014)
21. B. A. Junior, P. R. Pinheiro, and P. V. Coelho, : A Parallel Bi-ased Random-Key Genetic Algorithm with Multiple Populations Applied to Irregular Strip Packing Problems, : Mathematical Problems in Engineering, (2017).
22. F. Gronwald, S. Chang and A. Jin, : Determining a Source in Air Dispersion with a Parallel Genetic Algorithm, : International Journal of Emerging Technology and Advanced Engineering, vol. 7, no. 8, (2017).
23. A. Lissoni and C. Witt, : A Runtime Analysis of Parallel Evolutionary Algorithms in Dynamic Optimization, : Algorithmica, vol. 78, no. 2, pp. 641-659, (2017).
24. J. Lssig and D. Sudholt, : Adaptive population models for offspring populations and parallel evolutionary algorithms, Schwarzenberg, Austria: 11th Workshop Proceedings on Foundations of Genetic Algorithms, (2011).