A NEW MODEL OF PARALLEL PARTICLE SWARM OPTIMIZATION ALGORITHM FOR SOLVING NUMERICAL PROBLEMS

Poria Pirozmand[1], Hamidreza Alrezaamiri[2], Ali Ebrahimnejad[3*], Homayun Motameni[4]

[1]School of Computer and Software, Dalian Neusoft University of Information, Dalian, China

[2]Department of Computer Engineering, Babol Branch, Islamic Azad University, Babol, Iran

[3]Department of Mathematics**,** Qaemshahr Branch, Islamic Azad University**,** Qaemshahr, Iran

[4]Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran

E-mail: poria@neusoft.edu.cn[1], hamidreza.alreza@baboliau.ac.ir[2], a.ebrahimnejad@qaemiau.ac.ir[3*] (corresponding author), motameni@iausari.ac.ir[4]

DOI: https://doi.org/10.22452/mjcs.vol34no4.5

ABSTRACT

Evolutionary algorithms are suitable methods for solving complex problems. Many changes have thus been made on their original structures in order to obtain more desirable solutions. Parallelization is a suitable technique to decrease the runtime of the algorithm, and therefore, to obtain solutions with higher quality. In this paper, a new algorithm is proposed with two approaches, which is based on a parallelization technique with shared memory architecture. In the proposed algorithm, the search space is firstly decomposed into multiple equal and independent subspaces. Then, a subtask is performed on each subspace simultaneously in a parallel manner which leads to providing more qualified solutions. Splitting the search space into smaller subspaces causes the algorithm to find optimal solutions in each region in an easier way. The algorithm RAPSO is improved with applying a new acceleration coefficient which has been named IRAPSO. In the proposed algorithm, the IRAPSO is used as the subtask. For the sake of testing the proposed algorithm, fourteen well-known benchmarks of numerical optimizing problems are inspected. Then, the proposed algorithm is compared with algorithms PPBO and PSOPSO that were both based on the island model. The results of the proposed algorithm are much better than those of the other two algorithms.

Keywords: ***Island Model, Particle Swarm Optimization, RAPSO, Numerical Problems***

1.0 INTRODUCTION

Optimization approaches are widely used in different sciences and industries. The most famous optimization methods are linear, non-linear, and metaheuristic. Among these methods, metaheuristic algorithms have received considerable attention due to their simplicity and great ability in solving NP-hard problems. The most famous metaheuristic algorithms inspired by nature are the genetic algorithm [1], ant colony algorithm [2], bee colony algorithm [3] and particles swarm optimization algorithm [4]. Due to the fact that in many problems, increase of optimization, even in little amounts, leads to much saving and profit, there have always been attempts to improve optimization methods, as a result of which their higher rates of efficiency improves performance. In line with this attempt, in many researches, there have been attempts to obtain more desirable results from these algorithms by imposing some changes on the standard structure of the metaheuristic algorithms [5, 6]. For example, in the PSO algorithm, a part of the researches conducted were on the changes in variables such as velocity coefficient, number of particles, inertia amount and particles topology [7], while another part of the researches were on change in the structure of the PSO algorithm in order to solve discrete problems [8]. The main aim of the current study is to present a novel optimization algorithm based on parallelism that has acceptable results in terms of criteria, quality of solutions and run-time. In the proposed algorithm, one of the best versions of the PSO algorithm is improved by applying the acceleration coefficient. Then the problem is solved by dividing the search space and using parallelization techniques.

One of the methods for decreasing the run-time of algorithms is the parallelization technique. The parallelization technique can be investigated from different points of view. From an architectural perspective, the machines are divided into two groups, namely shared memory machines and distributed memory ones. In shared memory machines, all processors have access to shared memory, and the communications between them is done by reading and writing on the shared memory. In distributed memory machines, each processor has its own local memory, and communication between processors is done by the interchange of messages through networks [9]. There are two general types of parallelization techniques, namely data parallelization and task parallelization. In data parallelization, a special algorithm is imposed in different sets of data. This type of parallelization can be imposed easily on machines of shared memory. In task parallelization, the program is divided into multiple separated tasks. Each of these tasks is performed by its separate algorithm. Pipeline is a clear example of task parallelization [10].

The existing models for parallelization of the PSO algorithm can be categorized into three models [11]:

i) The Farming or Master-Slave Model: In this model, a processor named 'master' has the total procedure control of the program, and for decreasing run-time, it distributes the computations among slave processors and then archives the obtained results. This model is useful when the number of processors is small, or the slaves' computation performance is lengthy. If not so, the overload of communications between processors can impose expenses which are more than the advantages of parallelization.

ii) The Island Model: In this model, the swarm of particles is divided into several sub-swarms, and each processor runs the PSO algorithm independently on its own sub-swarm. After a given number of iterations, the migration action is performed in such a way that the processors send their best individual's position to their neighboring islands. Through this, other islands will be able to make use of these best positions in their own islands. In this model, the policy of migration is very important, and if an island is trapped in a local optimum, it may escape from that local optimum by achieving best individual position around its neighbors.

iii) The Diffusion Model: This model has been taken from the Island model with this difference that the swarm of each island has only one particle. This means that the number of total particles equals the number of islands. In this model, positions' updating action of particles is done by special policies.

Parallelization of each of these models by architecture of distributed memory, forces the computations to face two unwanted problems:

a) Heterogeneity of processors: Processors with lower processing speed always keep other processors waiting for them and act as a bottle neck of speed.

b) Delay of network: Because communication between processors is done through networks, the possibility of unpredicted delays is high.

In what follows, research studies conducted similarly will be reviewed. The first case of the parallelization particle swarm optimization algorithm was done in 2004 [12]. Parallelization of this algorithm is still an interesting and exciting topic, and is used for solving complex problems [13]. Gelado et al. [14] considered the architecture of distributed memory with heterogeneous processors. They investigated the Island model with different migration policies and different topologies. Zhao et al. [15] used the parallel algorithm of PSO in the architecture of distributed memory for solving complex problems. Li et al. [16] implemented the parallel algorithm of PSO in the Island model, and for improving the quality of solutions, they prevented the immature convergence. In order to improve the Island model, Chang et al. [17], introduced three strategies of migration to decrease the overload of communication between processors to make a balance between migration of particles and overload of processor communications. Moraes et al. [18] introduced the paralleled and asynchrony mode of this algorithm for improving the ability of PSO algorithms in solving complex problems. This algorithm, named AIU-PPSO, was implemented based on distributed structure and the Master-slave model. Setting the communication in this algorithm has been based on the MPI method. Tian and Lai [19] used one of the improved versions of the particles' swarm optimization algorithm named QPSO based on the distributed memory architecture, both in the Island model and the Master-slave model in two modes of synchrony and asynchrony for solving the inverse shape identification problem. In that study, parallel synchrony QPSO has achieved more qualified solutions, while parallel asynchrony QPSO performed faster in terms of operating and run-time speed. In paper [20],

authors proposed a variation of the algorithm called parallel swarms oriented particle swarm optimization (PSOPSO) which consists of a multi-stage and a single stage of evolution. In the multi-stage of evolution, individual sub swarms evolve independently in parallel, while in the single stage of evolution, the sub swarms exchange information and use a new velocity equation. In order to increase the rate of convergence and diversity of particles in the swarm via techniques, Schutte et al. [21] applied improved acceleration coefficients and divided search space into blocks based on the island model. Lai and Zhou [22] presented a multiple sub swarm parallel version of PSO based on the phenomenon of osmosis. This algorithm can adaptively decide when, how many, and from which sub swarm to which sub swarm particles will migrate. In other words, the three parameters, namely migration interval, migration rate, and migration direction need not be manually set in advance. Moreover, the PPBO can adaptively migrate a reasonable number of particles in the right direction. Campos et al. [23] presented two parallel PSO strategies based on multiple swarms to solve many objective optimization problems. The first strategy is based on Pareto dominance while the second is based on decomposition. Swarms execute on independent processors and communicate using a connected network [23]. Alrezaamiri et al. introduced the parallel multi-objective artificial bee colony (PMOABC) algorithm for solving the next release problem. PMOABC is an algorithm based on the master-slave model in the shared memory architecture. The authors could reach better solutions with this algorithm in comparison to the ABC algorithm [24]. Grisales-Noreña et al. introduced a parallel particle swarm optimization algorithm based on the master-slave model for an energy management system. The authors reduced run-time by allocating an evaluation of the objective function of particles to slaves. The algorithm could solve this giant problem with the acceptable solution in reasonable run-time [25]. The main contributions of this research are summarized as follows:

- A new technique for parallelization of metaheuristic algorithms is proposed.
- We improved one of the best versions of the PSO algorithm by applying the acceleration coefficient.
- The proposed algorithm makes the best use of the computer hardware capacity.
- It provides proper results for complex and large-scale problems.

The reminder of this paper is organized as follows: In Section 2, the structure of the PSO algorithm and the changes that have been made in different versions of it has been presented. In Section 3, the proposed algorithm is described in the architecture of shared memory. In Section 4, fourteen benchmarks used for testing the algorithm are introduced and the obtained results are analyzed. In Section 5, conclusions and suggestions for further studies are provided.

2.0 THE STRUCTURE OF ALGORITHM PSO

For the first time, Eberhart and Kennedy [4] introduced the PSO algorithm as a new optimization method in 1995. In this approach, the computations are derived as a result of observing the swarm behavior of some groups of animals, especially birds where each individual in society forms its next movement by considering shared information among members as well as its personal information. The PSO algorithm involves an initial swarm of particles that are dynamically moving in a search space with adjusted velocities. The velocity of particles is used for gaining an update on positions of particles. Each particle has a limited memory that saves the best position of its flying experience.
One of the key and attractive aspects of the PSO algorithm is its simplicity, where it involves only two equations of velocity and position. The position of each particle is indicative of a solution to the problem. The vectors of position x_i and velocity v_i are two vectors related to each particle *i* in the search space with N dimensions that are stated in the forms of $x_i = [x_{i1}, x_{i2}, ..., x_{in}]$ and $v_i = [v_{i1}, v_{i2}, ..., v_{in}]$. The velocity vector of each particle, which is based on the best search of the particle itself, and also on the best general experience of particles and the velocity vector of the particle, is updated according to the following equations.

$$v_i^{k+1} = wv_i^k + c_1 r_1 \left(pbest_i^k - x_i^k \right) + c_2 r_2 (gbest^k - x_i^k) \quad (1)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (2)$$

In which c_1 and c_2 are two positive constants, and r_1 and r_2 are two random numbers with uniform distribution in the range [0, 1]. Also, *w* is the inertia weight that can be a constant number and indicates the amount of effectiveness of the previous speed of the particle, $pbest_i^k$ is the best position of particle *i* that is obtained based on the experience of this

particle, $gbest^k$ is the best position of the particle based on general group experience. Moreover, *k* is an iteration index. The topology of the network is indicative of the manner of neighborhood among particles. Liu et al. introduced some topologies of the PSO algorithm [26].

2.1 Synchronism Particles Swarm Optimization Algorithm

Particles in the original particles' swarm optimization algorithm have complete information about their neighbors. It means that all particles know the position of the best particle in the search space, and by considering this knowledge, they move toward the best found position. This algorithm, which is the original algorithm of particle swarm optimization (SPSO), is also known as the Synchronous PSO (see Algorithm 1).

```
While not stopping condition do
    Foreach particle p in swarm do
        p. update ();
    end
    Foreach particle p in swarm do
        p. evaluate ();
        p. communicate ();
    end
end
```

Algorithm 1: Pseudocode of synchronous particle swarm optimization algorithm

In the first loop, each particle updates itself based on equations (1) and (2). In the second loop, each particle measures its position fitness in the evaluation function, and then, in the communication function, if its current position is better than its *pbest* position or *gbest* position, then its amount will be replaced.

2.2 Asynchronous Particle Swarm Optimization Algorithm

This algorithm is named the Asynchronous PSO (APSO). The main difference with the synchronous algorithm is in the available information of neighbor particles. In asynchronous updating, the particles have deficient information from the position of *gbest*. This has been explained in Algorithm 2.

```
While not stopping condition do
    Foreach particle p in swarm do
        p. update ();
        p. evaluate ();
        p. communicate ();
    end
end
```

Algorithm 2: Pseudocode of asynchronous particle swarm optimization algorithm

In this algorithm, when each particle is selected, it firstly updates its velocity and position vectors by the obtained information on that time from its neighboring particles. It then measures the fitness of its position. If the current position is better than that of its own *pbest* or neighbor's *gbest*, it changes their amounts. This algorithm has lower running time in comparison with SPSO in cases that the condition for ending the algorithm is the existence of equal numbers of iterations. Thus, APSO can be a good alternative for SPSO in problems where running time is very important.

2.3 Random Asynchronous Particle Swarm Optimization Algorithm

In this algorithm, which is also known as Random Asynchronous PSO (RAPSO), in each iteration, the particles are selected randomly for processing [27]. Random selection of particles is done by uniform distribution and is independent from pervious selection. It means that in each iteration, based on the swarm particle number, a particle is randomly selected to be updated, evaluated, and communicated. Consequently, in each iteration, it is possible for some particles to be selected more than once, while some others may not be selected at all. The pseudocode of this algorithm is given in Algorithm 3.

```
While not stopping condition do
    For i=1 : size (swarm) do
        particle p = randomParticle( swarm );
        p. update ();
        p. evaluate ();
        p. communicate ();
    end
end
```

Algorithm 3: Pseudocode of random asynchronous particle swarm optimization algorithm

3.0 PROPOSED ALGORITHM

We recall that the distributed architecture has two problems, namely unwanted delays of network, and heterogeneity of processors. Therefore, we hereby introduce our proposed algorithm based on the architecture of shared memory with the aim of using the optimum ability of multi-core processors. From the experimental and practical perspective, in the PSO algorithm, it is recognizable that for finding the optimum solution, the number of iterations and also the number of particles of the algorithm should be proportional to the size of the search space. That is, for finding the optimum solution, the more complicated and the bigger the search space, the higher number of particles needed, and the algorithm requires more iterations. This fact also causes an increase in run-time. Thus, in the proposed algorithm, we want to divide the search space to some equal separated subspaces and simultaneously, in a parallel mode, on each one of these divisions, perform a separated subtask. Finally, after all the performed parallel subtasks, we compare the *gbests* obtained from all subtasks with each other and consider the best *gbest* obtained from them as the global solution to the problem. Because the optimum point of the problem belongs to at least one of these subspaces, the subtask performed in this subspace can find the optimum solution more easily than the performance of the serial mode, because the searching space has been shortened much more. Because the search space for each algorithm has been much smaller, and the chance of finding the optimum solution has been increased, we can also decrease the number of iterations or particles of algorithms in this mode to cause a decrease in run-time. This technique causes improvements in the quality of solutions and also decreases run-time. We can also allocate the execution of each one of these subtasks to a core or a processor. With this technique, we can use the full capacity of the CPU to reduce run-time while not allowing any kernels to remain idle. Fig. 1 shows an example of dividing a search space and allocating the subtask to each of these subspaces. In Fig. 1.A, search space is depicted. It is assumed that the problem has 6 dimensions. Dimensions have been drawn with a green line. It is also assumed that the problem boundary is [0, 400]. In serial execution, an algorithm is executed on the total of this space for finding an optimum solution. In Fig. 1.A, it is assumed that the RAPSO algorithm is performed. In Fig. 1.B, the search space is divided into two equally separated subspaces in each of which a subtask is performed. At the end of their execution and after comparison of *gbests* obtained from them, the final solution of the problem is achieved. In Fig. 1.C, the space of the problem is divided into four, and in Fig. 1.D, into eight subspaces.

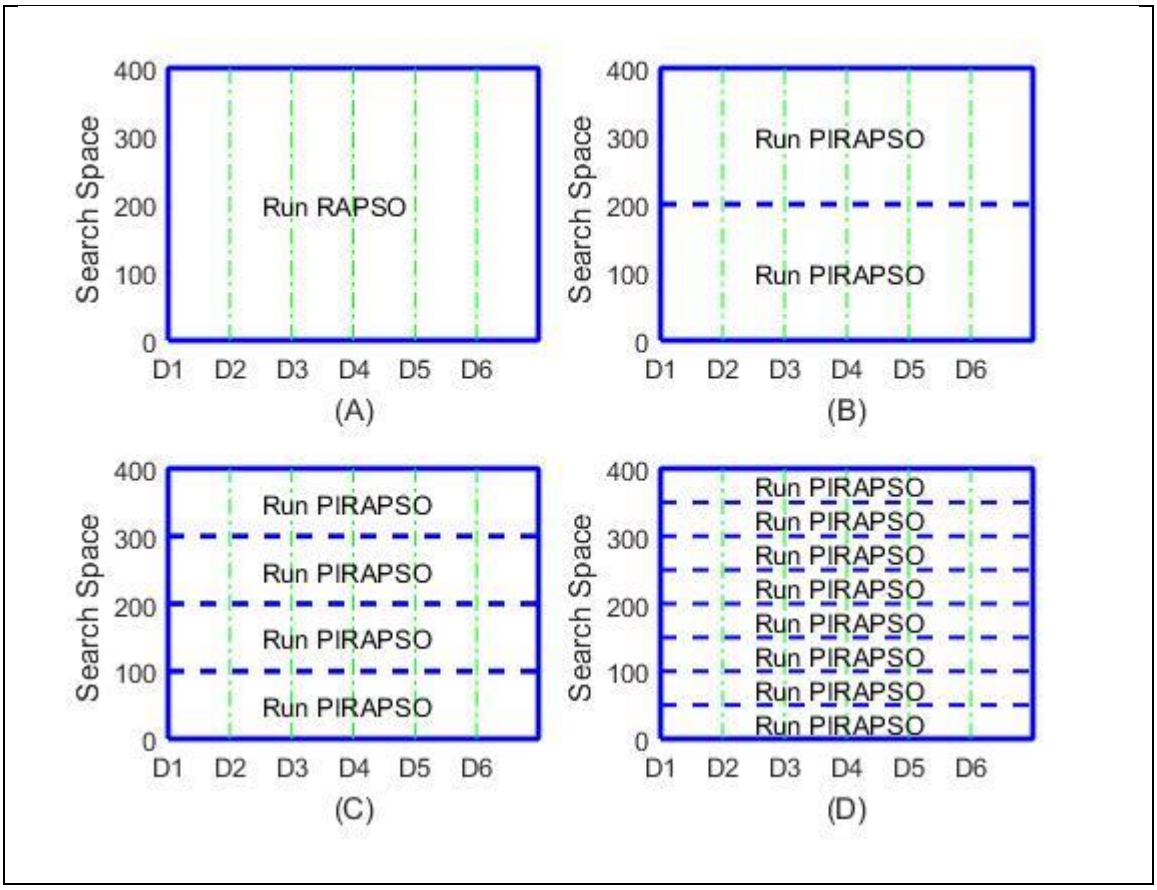

Fig. 1: Example of dividing a search space and allocating the subtask to each one of subspaces

Equations (3)-(5) are used to divide the search space:

$$LowRange^{1} = LowBoundary \tag{3}$$

$$LowRange^{n} = HighRange^{n-1} \quad , n = 2, ..., N \tag{4}$$

$$HighRange^{n} = LowRange^{n} + \frac{HighBoundary - LowBoundary}{N} \quad , n = 1, 2, ..., N \tag{5}$$

Here, N is defined as the number of the subspaces. If the problem space is defined as $[LowBoundary, HighBoundary]$, the search range of the *nth* subspace is defined by $[LowRange^{n}, HighRange^{n}]$. The user can select the appropriate number of the subtasks according to the hardware capacity and complexity of the problem. Today, most home computers have at least 8 cores and user can set algorithm to a number between 2 to 8. For example, if a user selects 3, algorithms, runs with 3 equal subspaces and 3 independent subtasks. If computers have 16, 32 or more cores, the algorithm can tune with these numbers. In the implementations we use three modes based on the number 2, 4, and 8. In serial execution, if the algorithm is for example operated in a total space with 500 iterations and 120 particles, for situations like Fig. 1.C, the proposed algorithm can be implemented in two ways. In the first way, the proposed algorithm can run four independent subtasks in each subspace. Each subtasks can have 125 iterations and 120 particles. In the second way, the proposed algorithm can run four independent subtasks in each subspace. Each subtasks can have 500 iterations and 30 particles. The proposed algorithm's run-times in both two approaches, are approximately equal to the serial algorithm runtime. But because of shrinking the search space, the best solution quality in both ways of the proposed algorithm will be much better than in serial execution. We aim to improve the RAPSO algorithm by applying

the acceleration coefficient presented in paper [27]. We thus use the name IRAPSO for this algorithm. Authors in [28] argued that a fixed value for the inertia weight w or a linear decrease in its value would fail to achieve the best solution because the search process of PSO is complex and non-linear. With this assumption, the function $\varphi(t) = \left| (gbest - x_j(t-1)) / (pbest_j - x_j(t-1)) \right|$ was proposed where $\left|(gbest - x_j(t-1))\right|$ is the Euclidean distance between the best global position at iteration *t-1* and the position x_j of particle *j*. It is possible to use the function $\varphi(t)$ to calculate the amount of w, c_{1j} and c_{2j} for each particle *j*, as opposed to the global w, c_1 and c_2 weights of the original PSO, using equations (6)-(8).

$$w(t) = \frac{w_{initial} - w_{final}}{1 + e^{\varphi(t)(t-((1+\ln(\varphi(t)))k_{\max})/\mu)}} + w_{final} \tag{6}$$

$$c_{1j}(t) = c_{1j}(t-1)\varphi(t)^{-1} \tag{7}$$

$$c_{2j}(t) = c_{2j}(t-1)\varphi(t) \tag{8}$$

where $k_{\max}$ is the maximum number of iterations, $w_{initial}$ and w_{final} are the initial and final values allowed for the inertia value, and μ is an adjustment factor to ensure that w keeps the reverse S change. The updating of weight w for the particle *j* is based on a variable sigmoid function. That means that the algorithm should search in a larger step size at the early iteration stage and should ensure the ability of global search. While the iteration goes on, the algorithm should search with a smaller step size at the later iteration stage and ensure the ability of local search. This is shown in Fig. 2. Equation (9) shows the new velocity equation of particle *j* at iteration *t* using adaptive weights [29].

$$v_j(t+1) = w(t)v_j(t) + (c_{1j}(t)r_1) * (pbest_j(t) - x_j(t)) + (c_{2j}(t)r_2) * (gbest - x_j(t)) \tag{9}$$

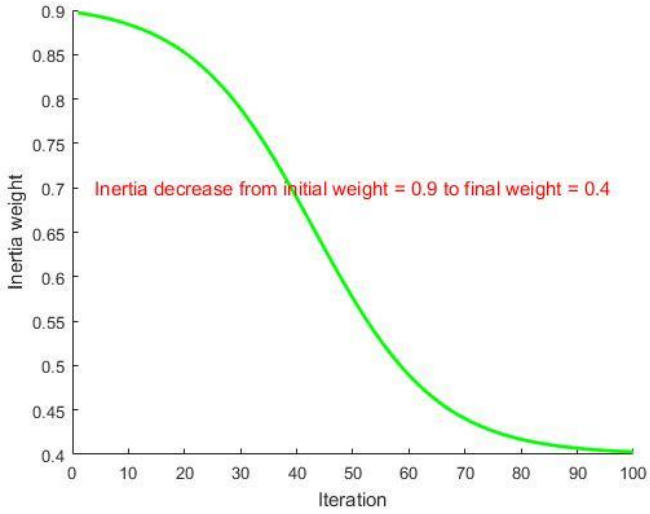

Fig. 2: Example of inertia decrease

In the simulation of the proposed algorithm, we use these inertia weights to update the vector velocity particles. We name the first and the second approaches by the *Parallel Improved RAPSO* (PIRAPSO1) and the *PIRAPSO2*, respectively. The difference between the proposed algorithm and serial performing is that in the serial model, the abilities of hardware are not used in parallel operating, and also, one algorithm is forced to search the total search space, while in the proposed algorithm, with the synchronized execution of separated subtasks in small separated subspaces, the chance of finding the optimum solution is increased, and there is no need for numerous iterations.

The difference of the proposed algorithm with the Island model is that in the Island model, the search space and numbers of iterations (like serial mode) are the same for all algorithms, and only the swarm is divided among algorithms. While in the first approach of the proposed algorithm, the search space and numbers of iterations are divided among algorithms. Also, each subtask runs with the complete particles swarm. In the second approach of the proposed algorithm, the search space and swarm are divided between subtasks, but the number of iterations is not divided. Each subtask runs with complete iterations. In both approaches, the subtask that is responsible for performing in the subspace where the optimum solution is located, can find the solution more easily.

The island model uses the migration policy, which in finding the optimum solution, helps the model. However, the much frequent interchange of messages between islands leads to a delay and increase of run-time. In this model, the initial swarm is divided among all islands, the increase of the number of islands leads to islands with low population. Having a small number of particles in an algorithm causes weakness in the search. In both approaches of the proposed algorithm, because the subtasks are performed independently from each other, we do not need the migration policy, and only at the end of the executions is it that subtasks send their *gbest* for determining the general solution of the problem to each other. The proposed algorithm is typically a parallelization on data and space. The proposed algorithm maintains the simple structure as well as the implemental and computational advantages of the original RAPSO. Advantages of the proposed algorithm compared to other versions of the PSO algorithm are the high-quality solutions and low run-time. Another advantage of the proposed algorithm is the use of hardware capabilities, such as the use of all processor cores that serial algorithms do not use. The applications of the proposed algorithm are approximately the same as the PSO algorithm and can thus be used in many optimization scopes.

4.0 EXPERIMENTS AND RESULTS

In this section, fourteen benchmarks are used to test the proposed algorithm and analyze the obtained results. The fourteen well-known benchmarks of optimization problems have been listed in Table 1 [30]. The optimal value of all minimization objective functions given in Table 1 is zero, i.e., $f(x^*) = 0$ where x^* is the optimal solution.

The proposed algorithm has made use of a computer with the following hardware: CPU Intel Core i7, 2.8 GHz, RAM 16GB, an operating system of Windows 10, and Matlab R2016a.

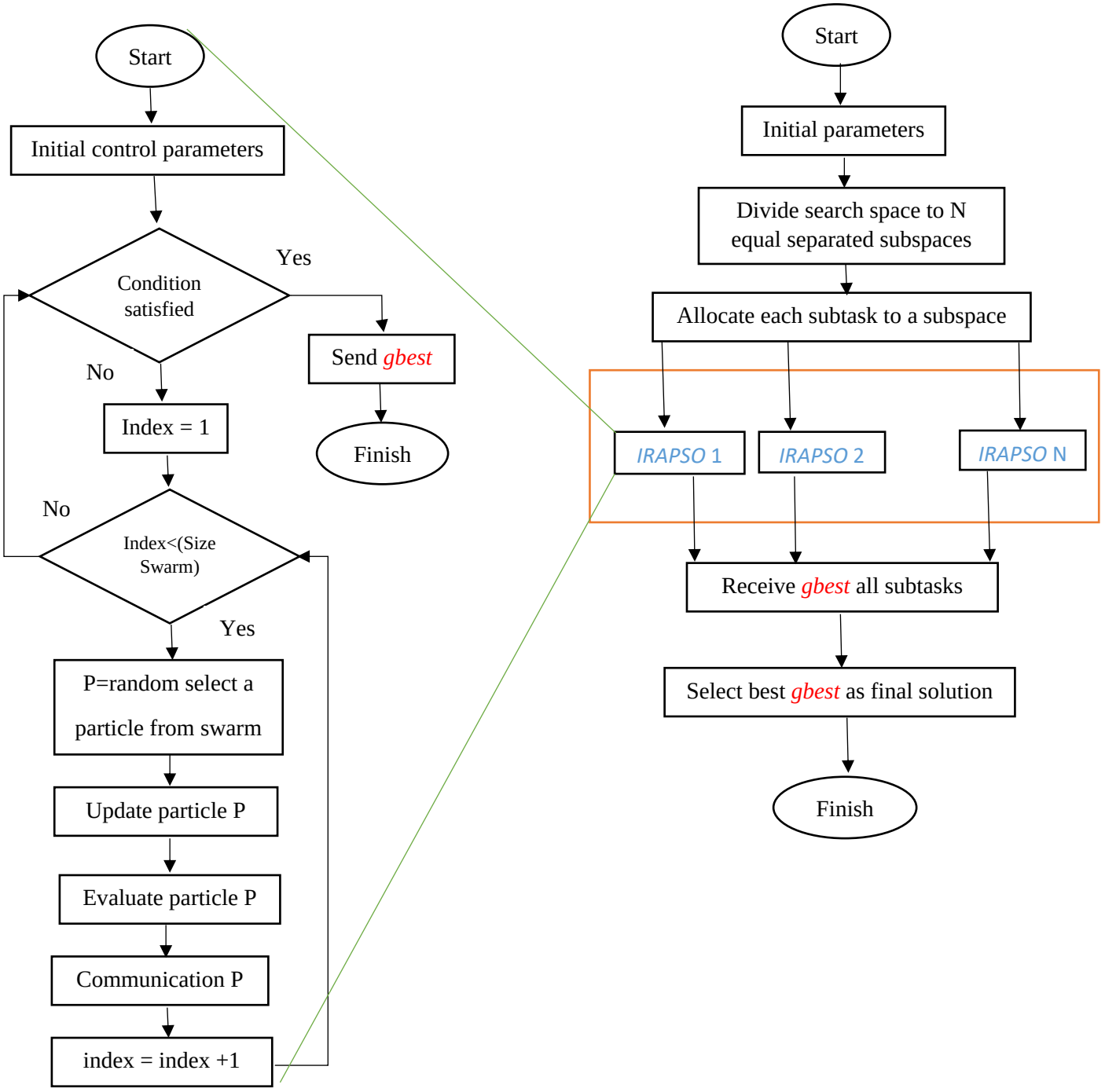

Fig. 3: Flowchart of the proposed algorithm

Table 1: List of benchmarks

NO	Function Name	Formula	Range	$f(x^*)$
1	Spherical	$\sum_{i=1}^{n} x_i^2$	[-5.12 , 5.12]	0
2	Powell Sum	$\sum_{i=1}^{n} \lvert x_i \rvert^{i+1}$	[-1 , 1]	0
3	Schwefel 2.20	$\sum_{i=1}^{n} \lvert x_i \rvert$	[-100 , 100]	0
4	Hyper Ellipsoid	$\sum_{i=1}^{n} i x_i^2$	[-5.12 , 5.12]	0
5	Chung Reynolds	$(\sum_{i=1}^{n} x_i^2)^2$	[-100 , 100]	0
6	Dixon Price	$(x_1-1)^2 + \sum_{i=2}^{n} i(2x_i^2 - x_{i-1})^2$	[-10 , 10]	0
7	Rosenbrock	$\sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	[-30 , 30]	0
8	Griewank	$\sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod \cos(\frac{x_i}{\sqrt{i}}) + 1$	[-100 , 100]	0
9	Rastrigin	$10n + \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i)$	[-600 , 600]	0
10	Ackley	$20 + e - 20\exp[-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}] - \exp[\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)]$	[-32.77,32.77]	0
11	Quartic	$\sum_{i=1}^{n} i x_i^4$	[-1.28 , 1.28]	0
12	Qing	$\sum_{i=1}^{n} (x_i^2 - i)^2$	[-500 , 500]	0
13	Csendes	$\sum_{i=1}^{n} x_i^6 (2 + \sin\frac{1}{x_i})$	[-1 , 1]	0
14	Quintic	$\Sigma_{i=1}^{n} (\Sigma_{j=1}^{i} x_j)^2$	[-10 , 10]	0

Table 2: Parameters of algorithm

Parameters	Value
Benchmark dimensions	50
Acceleration	$C_1=C_2=2$
Topology	Star
Maximum velocity	$0.2 * \lvert x_{max} - x_{min} \rvert$
Number of particles	120
Number of Iterations	500

In the first examination, quality of the best solution (*gbest*) and run-time of three versions of the introduced serial of the PSO algorithm are investigated. Table 2 shows the number of parameters used in the simulation. To evaluate the serial version, each algorithm was performed 20 times. The average value of the best solution is shown in Table 3. The standard deviation and average run-time for each benchmark are given in Table 3. In this table, the best solutions in each benchmark have been shown in red font faces. The difference of best solution quality in three versions of the algorithm was not very considerable, but the best solution was in the algorithm RAPSO, and then APSO, and finally SPSO or the standard version of the algorithm. In terms of the run-time criterion, there was no considerable difference among three algorithms.

Table 3: Performance of versions of serial of particles swarm optimization algorithm

NO	Function Name	SPSO			APSO			RAPSO		
		AVG	STD	Time (s)	AVG	STD	Time (s)	AVG	STD	Time (s)
F1	Spherical	1.7e+1	1.6e+1	2.76	7.80	1.2e+1	2.73	1.9e+2	2.0e+1	3.17
F2	Powell Sum	1.2e-6	7.9e+6	2.81	1.6e-10	4.7e-10	3.03	4.3e-11	8.3e-11	3.46
F3	Schwefel	3.38e+2	1.8e+2	2.83	3.33e+2	1.1e+2	2.78	2.7e+2	8.2e+1	3.36
F4	Hyper Ellipsoid	8.7	1.6e+1	2.84	2.09e+1	2.4e+1	2.79	3.1e+1	2.4e+1	3.22
F5	Chung Reynolds	1.2e+8	2.1e+8	2.74	6.0e+7	5.1e+7	2.74	1.0e+8	1.1e+1	3.16
F6	Dixon Price	1.0e+5	1.1e+5	2.63	4.7e+4	6.6e+4	2.60	1.3e+5	1.6e+5	3.07
F7	Rosenbrock	1.0e+5	1.1e+5	2.67	2.8e+4	4.2e+4	2.59	1.9e+4	3.7e+4	3.11
F8	Griewank	1.7	1.4	3.47	2.18	2.43	3.2	1.49	2.02	3.65
F9	Rastrigin	3.1e+2	4.7e+1	2.73	2.78e+2	6.0e+1	2.74	3.13e+2	4.4e+1	3.11
F10	Ackley	5.1	5.0	2.76	3.43	5.65	2.69	6.37	6.29	3.20
F11	Quartic	1.5e+1	2.0e+1	3.03	1.66e+1	1.3e+1	2.97	1.15e+1	9.55	3.50
F12	Qing	3.8e+3	3.8e+3	2.53	5.5e+3	8.5e+3	2.60	4.64e+2	5.7e+2	2.97
F13	Csendes	3.4e-7	2.6e-7	3.20	1.05e-6	1.4e-7	3.10	1.1e-7	1.0e-7	3.57
F14	Quintic	1.1e+1	5.7e+1	4.20	4.6	7.2	4.0	4.5e-1	6.1e-1	4.48

During test of different versions of the serial, the RAPSO algorithm had the best quality of the solution. We improved the algorithm RAPSO with applying a new acceleration coefficient and named it IRAPSO. In the proposed algorithm, IRAPSO is used as the subtask. For experiments, we ran the proposed algorithm and compared it with the algorithms SPSO, PSOPSO, and PPBO. Results obtained from these comparisons are provided in Table 4.

In Table 4.a, we compared the first approach of the proposed algorithm with the SPSO (a serial algorithm) and PSOPSO and PPBO (the two samples of the island model algorithms). In this test, the number of swarm members in all algorithms is equal to 120 particles. The SPSO algorithm is simulated with 500 iterations. The PSOPSO and PPBO algorithms have 4 islands. In these two algorithms, particles are split equally between the islands. That means each island has 30 particles. The PSOPSO algorithm aims to improve the quality of the solutions using two-step searches and the PPBO algorithm according to specific migration policies. The PIRAPSO1 algorithm is implemented in this test with 4 (IRAPSO) independent subtasks. In other words, the algorithm divides the search space into four equal regions, and a subtask runs independently in each region. The number of particles in each subtask is 120, and the number of their iterations is (500/4) =125. Also, the number of problem dimensions in this test was equal to 50.

The purpose of this test is to evaluate the quality of the solutions of the PIRAPSO1 algorithm in different benchmarks compared to other algorithms. In this table, the mean, standard deviation, and runtime of each algorithm are shown. These numbers are an average of 20 independent runs of each algorithm. The solutions obtained by the most successful algorithm in each benchmark are shown in red. The PIRAPSO1 algorithm obtained the best solution in most benchmarks. For example, the PIRAPSO1 algorithm was the only algorithm that could obtain the absolute solution for the F1, F4, F5, and F11 benchmarks. In benchmarks where the solutions obtained by other algorithms are better than PIRAPSO 1, the difference between the solutions were very slight. After the PIRAPSO1 algorithm, the PPBO algorithm had the best quality of solutions. The lowest-quality solutions belong to the PSOPSO algorithm. This test shows that dividing the search space and parallel execution of subtasks with high particles per subspace is much better than dividing the swarm between subtasks whereby each subtask searches the whole space. In this test, the parallel algorithms PPBO,

PSOPSO, and PIRAPSO1 were able to reduce the run-time significantly compared to the serial run-time. The lowest run-time was related to the PPBO algorithm because this algorithm avoids unnecessary migration using a clever policy and minimizes the delay imposed by migrations. After the PPBO algorithm, the PIRAPSO1 and the PSOPSO algorithm had shorter run-time with a negligible time difference, respectively. To test the quality of the PIRAPSO1 algorithm in more complex spaces, we performed the new test with the same previous test details in a 100-dimensional space. In Table 4.b, the mean value of solutions, the standard deviation, and the execution time of the algorithms are given.

Table 4.a: First approach of the proposed algorithm compared with other algorithms in 50 dimensions

NO	SPSO (Serial)			PSOPSO (4 Islands)			PPBO (4 Islands)			PIRAPSO 1 (4 Subtasks)		
	AVG	STD	Time	AVG	STD	Time	AVG	STD	Time	AVG	STD	Time
F1	1.74e+1	1.6e+1	2.76	1.05e+2	2.3e+1	1.03	2.61	8.28	0.90	0	0	0.90
F2	1.2e-6	7.9e+6	2.81	1.9e-2	3.5e-1	1.04	6.24e-8	7.6e-6	0.84	4.4e-11	1e-10	1.04
F3	3.3e+2	1.8e+2	2.83	1.1e+3	3.2e+1	0.83	6.7e+1	2.0e+1	0.78	3.71e+2	7.1e+1	0.92
F4	8.7e0	1.6e+1	2.84	1.0e+2	1.1e+1	1.00	1.7e0	0.7e+1	0.79	0	0	0.85
F5	1.2e+8	2.1e+8	2.74	1.8e+8	3.8e+7	0.85	6.6e+6	2.6e+7	0.75	0	0	0.90
F6	1.0e+5	1.1e+5	2.63	7.1e+5	2.1e+5	0.86	4.92e+4	1.0e+5	0.74	3.0e0	2.1e+1	0.89
F7	1.3e+4	3.1e+4	2.67	4.3e+7	1.1e+7	0.85	3.2e+4	4.3e+4	0.76	4.9e+1	1.1e+1	0.85
F8	1.7e0	1.4e0	3.47	1.1e+1	1.1e0	1.01	3.4e-1	9.8e-1	0.95	2.9e-1	1.6e-1	1.11
F9	3.1e+2	4.7e+1	2.73	2.2e+1	9.3e-1	0.93	3.0e+2	4.9e+1	0.75	8.3e+1	1.6+1	1.07
F10	5.1e0	5.0e0	2.76	1.8e+1	4.4e-1	0.92	2.3e0	8.2e-1	0.80	2.9e0	3.8e0	1.00
F11	1.5e+1	2e+1	3.03	5.1e+1	1.6e+1	0.98	8.0e-1	1.8e-1	0.85	0	0	1.02
F12	3.8e+3	3.8e+3	2.53	5.1e+10	1.1e+10	0.81	4.4e+5	5.0e+5	0.67	3.1e+4	3.5e+3	0.94
F13	3.4e-7	2.6e-7	3.20	3.9e-1	9.3e-2	1.01	8.7e-6	8.7e-6	0.91	5.7e-12	6e-12	1.10
F14	1.1e+1	5.7e+1	4.20	4.28e+4	8.59e+3	1.37	3.5e+1	1.6+1	1.16	5.5e+1	1.1e+1	1.26

Table 4.b: Second approach of the proposed algorithm compared with other algorithms in 100 dimensions

NO	SPSO (Serial)			PSOPSO (4 Islands)			PPBO (4 Islands)			PIRAPSO 1 (4 Subtasks)		
	AVG	STD	T	AVG	STD	T	AVG	STD	T	AVG	STD	T
F1	1.0e+2	5.7e+1	3.13	2.64e+2	1.44e+1	0.89	1.87e+1	1.84e+1	0.92	7.20	7.21	0.91
F2	1.8e-1	2.3e-1	3.49	2.88e-2	1.9e-2	1.19	1.8e-1	1.6e-1	1.04	2.89e-10	5.82e-10	1.13
F3	1.2e+3	2.8e+2	3.08	2.5e+3	1.6e+2	1.01	5.84e+2	2.93e+2	0.83	8.5e+2	1.33e+3	0.96
F4	1.1e+2	3.0e+1	2.94	2.6e+2	1.4e+1	1.10	1.87e+1	1.84e+1	0.87	6.21	7.58	0.97
F5	1.8e+9	2.0e+9	3.06	1.02e+10	1.11e+9	0.96	9.46e+6	1.47e+8	0.87	1.43e+8	6.44e+9	0.90
F6	3.2e+6	1.1e+6	2.78	4.41e+6	2.92e+5	0.92	1.42e+6	7.64e+5	0.80	7.21e+2	2.17e+3	0.90
F7	3.9e+7	5.1e+7	2.89	3.57e+8	2.99e+7	0.96	7.68e+6	2.69e+7	0.89	2.63e+5	2.29e+8	0.91
F8	1.4e+1	4.6e0	3.81	2.62e+1	1.38	1.16	2.79	1.76	1.15	1.36	1.02	1.09
F9	8.3e+2	5.4e+1	2.87	1.17e+3	3.8e+1	1.02	8.33e+2	6.19e+1	0.93	1.77e+2	1.76e+1	1.05
F10	1.8e+1	1.5e0	2.92	1.92e+1	1.47e-1	1.50	2.13e+1	3.10e+1	0.97	6.61	7.22	1.01
F11	1.7e+2	9.5e+1	3.67	2.9e+2	3.5e+1	1.26	5.0e+1	2.9e+1	1.04	0	0	0.98
F12	7.1e+10	6.1e+10	2.65	1.53e+11	1.7e+10	0.94	4.13e+9	2.1e+10	0.83	2.74e+5	1.07e+4	0.99
F13	2.5e-1	4.7e-1	3.73	1.32	1.78e-1	1.25	2.09e-1	4.40e-1	1.16	4.41e-5	1.40e-5	1.29
F14	4.6e+4	6.7e+4	5.54	1.39e+5	2.18e+4	1.81	5.84e+3	4.36e+3	1.60	2.10e+2	1.84e+1	1.55

In this table, the algorithm that obtained the best solution in each benchmark is also shown in red. It is evident that as the search space becomes more complex, the PIRAPSO1 algorithm proves to be far superior. Across the 12 benchmarks, the PIRAPSO1 algorithm obtained the best solution average. It could even find the absolute solution to the problem in benchmark F11. After the PIRAPSO1 algorithm, the best solutions belong to the PPBO algorithm. The quality of the solutions of the PSOPSO algorithm has fallen sharply as the search space has become more complex so that it will be even more inefficient than the SPSO serial algorithm in some benchmarks. In terms of run-time, as in the previous test, the best run-time belongs to the PPBO and the PIRAPSO1 algorithms, respectively. Although the PSOPSO algorithm has not been successful in improving the quality of the solutions, it has been able to significantly reduce run-time compared to the serial algorithm due to the use of parallel modelling in the island model style. This test proved that in

complex and complicated problems, serial algorithms cannot be useful. To determine whether the proposed algorithm is more successful in the first approach or second, and also to determine which approach of the proposed algorithm has the best performance and with how many subtasks in each run, we performed the next test. The previous test proved that in complex and complicated problems, serial algorithms cannot be useful. In this test, only parallel algorithms are evaluated. The results of the evaluation are shown in Table 5. The search space is 100-dimensional. For each algorithm, 120 particles and 500 iterations are considered. The algorithms are evaluated in three modes: with 2 subtasks (2nd), 4 subtasks (4th), and 8 subtasks (8th). The users can set the number of subtasks to a preferred number. But in the implementations, we use three modes based on numbers 2, 4, and 8. In the first case, the PPBO and PSOPSO algorithms have two islands where each of them has 60 particles and 500 iterations. The first proposed algorithm (PIRAPSO1) divides the search space into two distinct regions and runs a subtask in each region with 120 particles and 250 iterations. The second proposed algorithm (PIRAPSO2) splits the search space into two distinct regions and runs a subtask in each region with 60 particles and 500 iterations. Each algorithm is executed 20 times independently. Table 5 shows the solutions' average, standard deviation, and run-time of each algorithm. In the 2nd, 4th, and 8th modes, any algorithm that could obtain the best solution in each benchmark is highlighted with red in Table 5. Also, any algorithm that obtained the best solution in all three modes is marked with yellow.

Table 5.a: Compare the two approaches of the proposed algorithm with different subtasks in 100 dimensions

			F1	F2	F3	F4	F5	F6	F7
2 subtasks	PIRAPSO1 (2 Subtask) *Iter=250, Particles=2*120*	AVG	2.35e+1	0	2.11e+3	2.39e+1	1.43e+9	2.74e+4	7.43e+6
		STD	2.22e+1	0	2.60e+2	2.29e+1	1.77e+8	9.23e+4	1.46e+6
		Time	1.53	1.98	1.53	1.66	1.59	1.59	1.57
	PIRAPSO2 (2 Subtask) *Iter=500, Particles=2*60*	AVG	6.29e+1	0	2.01e+3	6.29e+1	8.20e+9	4.1e+4	1.61e+6
		STD	4.97e+1	0	2.60e+2	4.97e+1	6.30e+8	7.15e+3	3.48e+6
		Time	1.54	1.87	1.53	1.53	1.53	1.61	1.55
	PSOPSO (2 Island) *Iter=500, Particles=2*60*	AVG	2.57e+2	1.82e-2	2.41e+3	2.57e+2	9.71e+9	4.12e+6	3.27e+8
		STD	2.13e+1	1.36e-2	1.62e+2	1.90e+1	1.61e+9	5.00e+5	6.28e+7
		Time	1.60	2.01	1.51	1.58	1.57	1.48	1.53
	PPBO (2 Island) *Iter=500, Particles=2*60*	AVG	8.59e+1	9.62e-2	9.67e+2	8.59e+1	1.44e+9	1.91e+6	1.63e+8
		STD	5.30e+1	1.12e-1	1.63e+2	5.30e+1	1.85e+9	9.70e+5	9.28e+7
		Time	1.30	1.74	1.37	1.39	1.35	1.32	1.31
4 subtasks	PIRAPSO1 (4 Subtask) *Iter=125, Particles=4*120*	AVG	7.20	2.89e-10	8.50e+2	6.21	1.43e+8	7.12e+2	2.63e+5
		STD	7.21	5.82e-10	1.33e+3	7.58	6.44e+9	2.17e+3	2.29e+8
		Time	0.91	1.13	0.86	0.97	0.90	0.90	0.91
	PIRAPSO2 (4 Subtask) *Iter=500, Particles=4*30*	AVG	1.57e+1	2.37e-6	1.25e+3	1.57	4.87e+9	7.60e+2	3.65e+5
		STD	0.98e+1	4.98e-6	1.84e+2	9.86	4.49e+7	2.37e+3	5.95e+5
		Time	0.87	1.02	0.82	0.95	0.90	0.82	0.89
	PSOPSO (4 Island) *Iter=500, Particles=4*30*	AVG	2.64e+2	2.88e-2	2.52e+3	2.64e+2	1.02e+10	4.41e+6	3.57e+8
		STD	1.44e+1	1.9e-2	1.64e+2	1.44e+1	1.11e+9	2.92e+5	2.99e+7
		Time	0.89	1.19	1.01	1.10	0.96	0.92	0.96
	PPBO (4 Island) *Iter=500, Particles=4*30*	AVG	1.87e+1	1.8e-1	5.84e+2	1.87e+1	9.46e+6	1.42e+6	7.68e+6
		STD	1.84e+1	1.6e-1	2.93e+2	1.84e+1	1.47e+8	7.64e+5	2.69e+7
		Time	0.87	1.04	0.83	0.87	0.87	0.80	0.89
8 subtasks	PIRAPSO1 (8 Subtask) *Iter=63, Particles=8*120*	AVG	1.30	3.38e-11	4.45e+2	2.45	1.28e+6	9.05e+1	1.85e+4
		STD	1.03	6.92e-12	5.74e+1	1.76	1.32e+6	8.9e+1	2.91e+4
		Time	0.80	0.85	0.79	0.77	0.75	0.78	0.76
	PIRAPSO2 (8 Subtask) *Iter=500, Particles=8*15*	AVG	4.91e+1	8.19e-7	6.55e+2	4.91	5.39e+6	1.16e+2	3.07e+4
		STD	1.89e+1	1.64e-6	1.20e+3	1.89	5.44e+6	2.75e+1	6.02e+4
		Time	0.69	0.85	0.70	0.73	0.70	0.75	0.74
	PSOPSO (8 Island) *Iter=500, Particles=8*15*	AVG	2.85e+2	4.3e-2	2.61e+3	2.80e+2	1.19e+9	4.98e+6	3.91e+8
		STD	2.60e+1	3.07e-2	1.29e+2	2.23e+1	2.31e+9	8.97e+5	3.07e+7
		Time	0.70	0.85	0.72	0.81	0.71	0.73	0.72
	PPBO (8 Island) *Iter=500, Particles=8*15*	AVG	1.16e+1	3.8e-2	3.19e+2	1.16e+1	2.08e+7	7.96e+5	5.8e+7
		STD	3.67e0	2.7e-9	1.41e+2	3.67	9.94e+6	4.02e+5	2.46e+7
		Time	0.61	0.72	0.62	0.64	0.65	0.63	0.65

The similarity of the two approaches of the proposed algorithm is that both of them divide the search space. The difference between the two approaches is that the first approach divides the total number of iterations equally between subtasks with space division, while the second approach divides the whole swarm among subtasks with space division. In most benchmarks, the solution to the first approach of the proposed algorithm is better than that of the second. This suggests that a greater number of particles is more useful than a greater number of iterations. Obviously, an increase in the number of particles and iterations leads to an increase in the likelihood of the favorability of the solutions, but the run-time increases accordingly. For this reason, an algorithm in this scope is considered more successful if it provided higher quality solutions in shorter time. One of our goals in presenting these two approaches is that the volume of computations and run-time will not increase. The run-times of the two approaches of the proposed algorithm are not very different. In fact, the computational load of both approaches is completely equal. For example, in the 4th mode, the first approach has 4 subtasks, each of which has 125 iterations and 120 particles. In other words, the computational volume is equal to 4*125*120=60000. However, in the second approach, each subtask has 500 iterations and 30 particles. In other words, the computational volume is 4*500*30=60000.

Table 5.b: Compare the two approaches of the proposed algorithm with different subtasks in 100 dimensions.

			F8	F9	F10	F11	F12	F13	F14
2 Subtasks	PIRAPSO1 (2 Subtask) *Iter=250, Particles=2*120*	AVG	2.36	6.09e+2	1.70e+1	0	2.88e+5	7.70e-6	2.75e+2
		STD	2.30	4.57e+1	5.99	0	5.92e+3	3.26e-6	2.26e+1
		Time	1.93	1.77	1.81	1.67	1.63	2.27	2.48
	PIRAPSO2 (2 Subtask) *Iter=500, Particles=2*60*	AVG	4.37	6.72e+2	1.85e+1	2.68	2.90e+5	3.94e-7	2.88
		STD	4.28	8.85e+1	1.71	6.19	2.13e+4	1.79e-7	2.06
		Time	1.82	1.80	1.74	1.62	1.56	2.31	2.47
	PSOPSO (2 Island) *Iter=500, Particles=2*60*	AVG	2.55	1.16e+3	1.91	1.92e+1	1.42e+11	1.20	1.25e+5
		STD	2.03	3.90e+1	2.28e-1	2.05e-1	1.65e+110	1.77e-1	1.39e+4
		Time	1.99	1.59	1.73	1.60	1.50	2.13	3.04
	PPBO (2 Island) *Iter=500, Particles=2*60*	AVG	9.19	8.24e+2	8.24e+2	1.45e+2	2.6e+10	2.80e-1	3.89e+4
		STD	5.06	8.09e+1	8.09+1	8.47e+1	3.21e+10	4.81e-1	5.22e+4
		Time	1.82	1.36	1.41	1.78	1.24	1.80	2.82
4 Subtasks	PIRAPSO1 (4 Subtask) *Iter=125, Particles=4*120*	AVG	1.36	1.77e+2	6.61	0	2.74e+5	4.41e-5	2.10e+2
		STD	1.02	1.76e+1	7.22	0	1.07e+4	1.40e-5	1.84e+1
		Time	1.09	1.05	1.01	0.98	0.92	1.29	1.55
	PIRAPSO2 (4 Subtask) *Iter=500, Particles=4*30*	AVG	2.06	2.01e+2	8.46	1.34e-1	2.94e+5	1.48e-6	2.06e+2
		STD	1.20	4.35e+1	6.32	2.06e-1	2.29e+4	1.15e-6	2.55e+1
		Time	1.05	1.01	0.99	0.92	0.91	1.15	1.39
	PSOPSO (4 Island) *Iter=500, Particles=4*30*	AVG	2.62e+1	1.17e+3	1.92e+1	2.91e+2	1.53e+11	1.32	1.39e+5
		STD	1.38	3.8e+1	1.47e-1	3.5e+1	1.7e+110	1.78e-1	2.18e+4
		Time	1.16	1.02	1.5	1.26	0.94	1.25	1.81
	PPBO (4 Island) *Iter=500, Particles=4*30*	AVG	2.79	8.33e+2	2.13e+1	5.00e+1	4.13e+9	2.09e-1	5.84e+3
		STD	1.76	6.19e+1	3.10e+1	2.92e+1	2.10e+10	4.40e-1	4.36e+3
		Time	1.15	0.93	0.97	1.04	0.83	1.16	1.60
8 Subtasks	PIRAPSO1 (8 Subtask) *Iter=63, Particles=8*120*	AVG	4.81e-1	1.47e+2	7.65	3.14e-3	2.51e+4	1.30e-5	3.31e+2
		STD	3.41e-1	3.95e+1	1.82	9.94e-3	1.59e+4	2.21e-6	3.14e+1
		Time	0.92	0.84	0.81	0.81	0.79	0.97	1.22
	PIRAPSO2 (8 Subtask) *Iter=500, Particles=8*15*	AVG	1.14	2.21e+2	1.06e+1	6.29e-3	3.63e+10	8.69e-2	6.49e+1
		STD	3.88e-2	4.41e+1	3.45e-2	1.98e-2	7.72e+9	1.50e-2	1.23e+1
		Time	0.83	0.79	0.80	0.76	0.69	0.88	1.18
	PSOPSO (8 Island) *Iter=500, Particles=8*15*	AVG	2.8e+1	1.20e+3	1.93e+1	3.30e+2	1.68e+11	1.66	1.54e+5
		STD	1.82	3.55e+1	2.08e-1	3.53e+1	2.17e+10	3.45e-1	3.57e+4
		Time	0.90	0.78	0.80	0.88	0.70	0.91	1.29
	PPBO (8 Island) *Iter=500, Particles=8*15*	AVG	2.10	8.18e+2	8.21e+2	4.45e+1	1.24e+10	2.72e-5	1.43e+4
		STD	3.50e-1	4.56e+1	7.14e+1	2.45e+1	5.15e+9	2.05e-5	1.15e+4
		Time	0.86	0.68	0.63	0.77	0.62	0.80	1.10

In the case with more particles, the likelihood of falling into the trap of the local optimum decreases, while a higher number of iterations will not help in the existence of the local optimum. For this reason, the first approach of the proposed algorithm is more successful than the second. The results provided in Table 5 indicate this point. We now examine the number of subtasks that the proposed algorithm is more efficient. As we see in Table 5, the best solutions of the proposed algorithm are obtained in 7 benchmarks in the 8th mode, 3 benchmarks in the 4th mode, and 3 benchmarks in 2nd mode. The proposed algorithm in the F9 and F12 benchmarks, which have the largest and the most complex range of searches, could find the best solution in the 8th mode. However, in the F2 and F11 benchmarks, which have a small search range, the 2nd mode could find the most favorable solution. The proposed algorithm has the worst solution in benchmarks with a small range in both the 4th and 8th modes compared to the 2nd mode. The reason is that after updating the equations of velocity and position, the particles usually go out of the boundary of the space and are replaced with a random position. The excessive shrinking of the regions causes the particles to leave the region during updating, and the process of particles moving towards the optimal solution will run into difficulties, while in benchmarks with large regions, the division of the search space into 4 or 8 regions leads to an improved quality of the solutions. It can be concluded from this point that the proposed algorithm has the best quality when its number of subtasks is commensurate with the range of the search space. In other words, more subtasks are needed for problems with large regions, and fewer subtasks are needed for problems with smaller regions. In Table 5, across 13 benchmarks, the best solutions of the two approaches in the proposed algorithm are better than the best solutions of the two PPBO and PSOPSO algorithms. As the number of islands is increased, the quality of solutions obtained by the PSOPSO algorithm decreases. In this algorithm and the PPBO algorithm, an increase in the number of islands means a reduction of the number of particles per island. The scarcity of particles in the islands makes particles unable to search the search space well and will not lead to a good solution. Of course, the PPBO algorithm has partly overcome this problem using a clever migration policy, but an excessive increase in the number of islands will slow down this algorithm as well. Fig. 4 gives the best solution value of simulated algorithms in F1, F8, F9, and F11 benchmarks in the 2nd, 4th, 6th, and 8th modes.

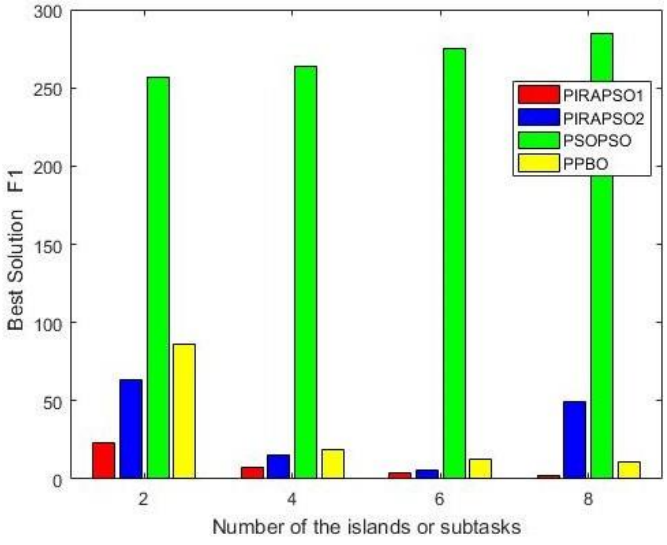

Fig. 4.a: Results of the benchmark F1.

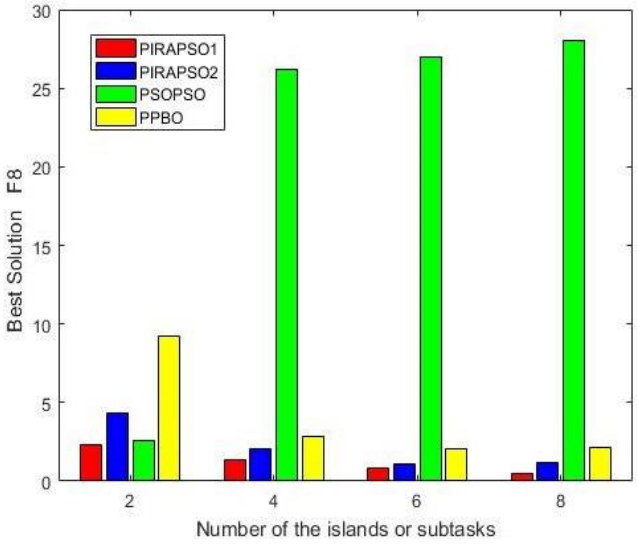

Fig. 4.b: Results of the benchmark F8.

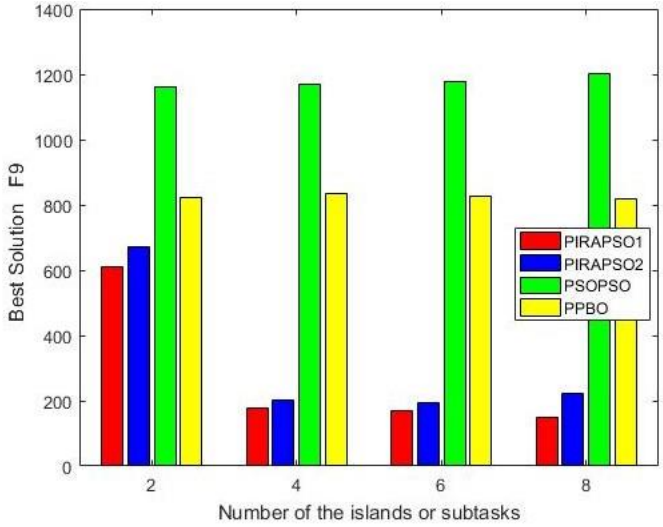

Fig. 4.c: Results of the benchmark F9.

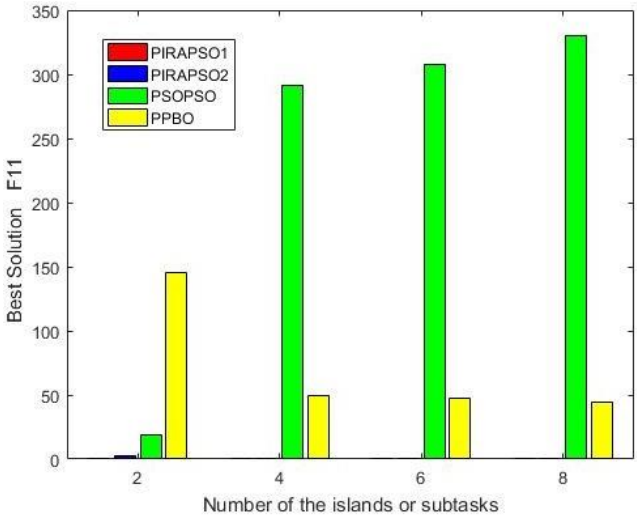

Fig. 4.d: Results of the benchmark F11.

In Fig. 4.a, the best results are given by the PIRAPSO1 algorithm, where with the increase of the number of subtasks, the quality of the solutions is improved; however, the worst results are given by the PSOPSO algorithm where with the increase of the number of the islands, quality of the solutions is decreased. In Fig. 4.b, algorithm PSOPSO with 2 islands gives good results, but with the increase in the number of the islands, the quality of the results fell sharply. The solutions of the PPBO algorithm are improved with an increased number of islands. The result of the PPBO algorithm is acceptable for 6 and 8 islands. In Fig. 4.c, with the increase in the number of the subtasks, the solution difference between the two approaches of the proposed algorithm with the two other algorithms increased. In Fig 4.d, the PIRAPSO 1 algorithm with 2, 4, 6 subtasks finds the optimum solution. The PIRAPSO 2 algorithm finds solutions which are very close to the optimum point. The difference between the results of the PIRAPSO 2 with the two others algorithm is very high, thus one cannot see the solutions' bars of this algorithm in the plot.

5.0 CONCLUSION

In this paper, a new parallelization-based algorithm has been introduced for solving numerical problems. At the first stage of the proposed algorithm, the algorithm RAPSO was improved by applying a new acceleration coefficient namely IRAPSO. Subsequently, the search space was split into several independent subspaces, after which the separated subtasks were performed in parallel in each subspace. In the proposed algorithm, two approaches were presented. In the first approach, the search space and number of iterations was divided between subtasks, while the swarms were not divided. Then, each subtask was run with a complete particle swarm. In the second approach, the search space and swarm were divided between subtasks, but the number of iterations was not divided. Each subtask here runs with complete iterations. After the implementation of all subtasks, the solutions were derived. By comparison, the best solution of the problem was selected. The operation of each one of the subtasks can be given to a core or a processor. This algorithm can use the full capacity of the hardware. For analyzing performance, the two approaches of the proposed algorithm were operated on fourteen famous benchmarks of numerical problems and compared with the two algorithms PPBO and PSOPSO based on the Island model, which is one of the most common parallelization models. It was observed that the proposed algorithm reached better solutions. Dividing the search space to the subspaces and allocating a subtask to each of them causes the action of searching to be done better, and thus better solutions will be obtained from the problem. The comparison of the solutions in the proposed algorithm with the two algorithms based on the island model confirms this claim. Moreover, from the theoretical view, it is obvious that a search in a space with a specific number of particles and iterations can yield better solutions than the search in multiple spaces with the same number of the particles and iterations. It should be noted that combination of subspaces will not provide better solution by interaction between the solutions of subspaces. Because when the best point there exists in a subspace, only the

corresponding subtask can find it. Solutions of the other subtasks (and the other subspaces) cannot help the algorithm. If after executing algorithm, all subtasks run again in the best subspace independently, we can be hopeful to reach the best solution. But this approach increase run time seriously. However, for the future works, if only quality of the solution be criteria, this idea may be useful.

The proposed algorithm can be extended and used in other scopes of the engineering as well. Future researches in this scope can be done in the following paths:

- Investigating a two-step search, meaning that after executing the proposed algorithm, all subtasks are independently run once again in the best subspace.
- Determining the best number of subtasks with attention to the space of the problems.
- Investigating the proposed algorithm with more complicated benchmarks.

REFERENCES

[1] D. Arindam, R. Pradhan, A. Pal and T. Pal, "A genetic algorithm for solving fuzzy shortest path problems with interval type-2 fuzzy arc lengths", *Malaysian Journal of Computer Science,* Vol.31, No.4, 2018, pp. 255-270.

[2] M. Poongothai, A. Rajeswari and A. Jabar Ali. "A MULTIOBJECTIVE APPROACH FOR REAL TIME TASK ASSIGNMENT PROBLEM IN HETEROGENEOUS MULTIPROCESSORS", *Malaysian Journal of Computer Science,* Vol.32, No.2, 2019, pp. 112-132.

[3] A. Ebrahimnejad, M. Tavana and H.R. Alrezaamiri, "A novel artificial bee colony algorithm for shortest path problems with fuzzy arc weights", *Measurement,* Vol.93 ,2016,pp. 48-56

[4] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory”. In Micro Machine and Human Science, 1995. MHS'95, Proceedings of the Sixth International Symposium on (pp. 39-43). IEEE.

[5] H.R. Alrezaamiri, A. Ebrahimnejad, H. Motameni, “Software requirement optimization using a fuzzy artificial chemical reaction optimization algorithm”. *Soft Computing,* 2018, pp.1-16.

[6] D, Emmanuel Gbenga, and E.I. Ramlan, "PDPSO: THE FUSION OF PRIMAL-DUAL INTERIOR POINT METHOD AND PARTICLE SWARM OPTIMIZATION ALGORITHM", *Malaysian Journal of Computer Science,* Vol 31, No. 1, 2018, pp-17-34.

[7] W. Sun, A. Lin, H. Yu, Q. Liang and G. Wu, “All-dimension neighborhood based particle swarm optimization with randomly selected neighbors”, *Information Sciences,* Vol 405, 2017, pp.141-156.

[8] A. Ebrahimnejad, Z. Karimnejad, H.R. Alrezaamiri, “Particle swarm optimisation algorithm for solving shortest path problems with mixed fuzzy arc weights”, *International Journal of Applied Decision Sciences,* Vol.8, No.2, 2015, pp.203-222.

[9] S. Santander-Jiménez, M.A. Vega-Rodríguez, “On the design of shared memory approaches to parallelize a multiobjective bee-inspired proposal for phylogenetic reconstruction”, *Information Sciences,* Vol.324, 2015, pp.163-185.

[10] R.C. Green, V. Agrawal, “A case study in multi-core parallelism for the reliability evaluation of composite power systems”, *The Journal of Supercomputing,* Vol.73, No.12, 2017, pp.1-25.

[11] G.A. Laguna-Sánchez, M. Olguín-Carbajal, N, Cruz-Cortés, R. Barrón-Fernández and J.A. Álvarez-Cedill, “Comparative study of parallel variants for a particle swarm optimization algorithm implemented on a multithreading GPU”,*Journal of Applied Research and Technology,* Vol.7, No.3, 2009, pp.292-307.

[12] J.F. Schutte, J.A. Reinbolt, B.J. Fregly, R.T. Haftka and A.D. George, "Parallel global optimization with the particle swarm algorithm", *International Journal for Numerical Methods in Engineering*, Vol.61, No.13, 2004, pp.2296-2315.

[13] L. Mussi, Y.S. Nashed and S. Cagnoni, "GPU-based asynchronous particle swarm optimization", *In Proceedings of the 13th annual conference on Genetic and evolutionary computation*. July 2011, pp.1555-1562.

[14] I. Gelado, J.E. Stone, J. Cabezas, S. Patel, N. Navarro and W.M.W. Hwu, "An asymmetric distributed shared memory model for heterogeneous parallel systems", *In ACM SIGARCH Computer Architecture News*, Vol.38, No.1, 2010, pp. 347-358.

[15] Bingyu, Z.Y.Y.J. Li, Z. Chuansheng, "A parallel particle swarm optimization algorithm based on multigroup for solving complex functions optimization", *Journal of Computer Engineering and Applications*, Vol.41, No.16, 2005, pp.58-60.

[16] J.Z. Li, W.N. Chen, J. Zhang and Z.H. Zhan, "A Parallel Implementation of Multiobjective Particle Swarm Optimization Algorithm Based on Decomposition", In *2015 IEEE Symposium Series on Computational Intelligence*, 2015, pp.1310-1317. IEEE.

[17] J.F. Chang, J.F. Roddick, J.S. Pan and S.C. Chu, "A parallel particle swarm optimization algorithm with communication strategies", *Journal of Information Science and Engineering*, 2005, pp.809- 818.

[18] A.D.O.S. Moraes, P.L.D.C. Lage and A.R. Secchi, "ASYNCHRONOUS PARALLELIZATION OF THE PARTICLE SWARM OPTIMIZATION ALGORITHM AND ITS APPLICATION TO PARAMETER ESTIMATION IN A LARGE DIMENSIONAL SPACE", 22nd International Congress of Mechanical Engineering,2013, pp.3490- 3501.

[19] N. Tian, C.H. Lai, "Parallel quantum-behaved particle swarm optimization", *International Journal of Machine Learning and Cybernetics*, Vol.5, No.2, 2014, pp.309-318.

[20] T. Gonsalves and A. Egashira, "Parallel swarms oriented particle swarm optimization", *Applied Computational Intelligence and Soft Computing*, Vol.2013, No.14, 2013, pp.1-8.

[21] M. Bisheban, M.J. Mahmoodabadi and A. Bagheri, "Partitioned Particle Swarm Optimization", *J Appl Computat Math*, Vol.2, No.3, 2013, pp.1-10.

[22] X. Lai and Y. Zhou "An adaptive parallel particle swarm optimization for numerical optimization problems", *Neural Computing and Applications*, 2018, pp.1-19.

[23] De Campos Jr, Arion, Aurora TR Pozo, and Elias P. Duarte Jr. "Parallel multi-swarm PSO strategies for solving many objective optimization problems." *Journal of Parallel and Distributed Computing* 126 (2019): 13-33.

[24] Alrezaamiri, Hamidreza, Ali Ebrahimnejad, and Homayun Motameni. "Parallel multi-objective artificial bee colony algorithm for software requirement optimization." *Requirements Engineering* (2020): 1-18.

[25] Grisales-Noreña, Luis F., Oscar Danilo Montoya, and Carlos Andrés Ramos-Paja. "An energy management system for optimal operation of BSS in DC distributed generation environments based on a parallel PSO algorithm." *Journal of Energy Storage* 29 (2020): 101488.

[26] Q. Liu, W. Wei, H. Yuan, Z.H. Zhan and Y. Li, "Topology selection for particle swarm optimization", *Information Sciences*, Vol.363, 2016, pp.154-173.

[27] J. Rada-Vilela, M. Zhang and W. Seah, "Random asynchronous PSO", *In Automation, Robotics and Applications (ICARA), 2011 5th International Conference on* (pp. 220-225). IEEE.

[28] Y. Zhang, X. Xiong and Q. Zhang, "An improved self-adaptive PSO algorithm with detection function for multimodal function optimization problems", *Mathematical Problems in Engineering*, Vol.2013, 2013, pp.1-8.

[29] Silva Filho, M. Telmo, B.A. Pimentel, R.M. Souza and A.L. Oliveira, "Hybrid methods for fuzzy clustering based on fuzzy c-means and improved particle swarm optimization", *Expert Systems with Applications*, Vol.42, No.17, 2015, pp.6315-6328.

[30] M. Jamil, X.S. Yang, "A literature survey of benchmark functions for global optimisation problems", *International Journal of Mathematical Modelling and Numerical Optimisation*, Vol.4, No.2, 2013, pp.150-194.