Applying Swarm Optimization Techniques to Calculate Execution Time for Software Modules

Nagy Ramadan Darwish
Department of Computer and Information Sciences, Institute of Statistical Studies and Research, Cairo University, Cairo, Egypt

Ahmed A. Mohamed
Department of Information System, Higher Technological Institute, 10th of Ramadan City, Egypt

Bassem S. M. Zohdy
Department of Business Technology, Canadian International College, Cairo, Egypt

***Abstract*—This research aims to calculate the execution time for software modules, using Particle Swarm Optimization (PSO) and Parallel Particle Swarm Optimization (PPSO), in order to calculate the proper time. A comparison is made between MATLAB Code without Algorithm (MCWA), PSO and PPSO to figure out the time produced when executing any software module. The proposed algorithms which include the PPSO increase the speed of executing the algorithm itself, in order to achieve quick results. This research introduces the proposed architecture to calculate execution time and uses MATLAB to implement MCWA, PSO and PPSO. The results show that PPSO algorithm is more efficient in speed and time compared to MCWA and PSO algorithm for calculating the execution time.**

Keywords*—Particle Swarm Optimization; Parallel Particle Swarm Optimization; MATLAB Code without Algorithm

I. INTRODUCTION

Testing is a crucial phase that is performed during software development. It is a primary technique which is used to gain consumer confidence in the software. It is conducted by executing the program developed with test inputs and comparing the observed output with the expected one [1, 2]. Testing is the writing and applying all software tests to ensure the confidence in the operation of the program. Testing is the phase of development that is carried out after the main coding efforts [3].

Execution time is the time during which software is running. Calculating execution time is very important in many fields, such as; medical system, army system, and airlines system … etc., Where any time delay in these software misfortunes may occur.

This research selects primary studies that published between 2005 and 2015, while searching many electronic databases in order to determine if similar work had already been performed, and locates potentially relevant studies.

C. Mao [4] proposed a search-based test data generation solution for software structural testing using particle swarm optimization (PSO) technique. A. Windisch, S. Wappler and j. Wegener [5] applied a particle swarm algorithm for evolutionary structural testing. R. Ding and H. Dong [6] proposed a hybrid particle swarm genetic algorithm to apply in software testing using case automate generations. A. S. Andreou, K. A. Economides and A. A. Sofokleous [7] proposed an enhanced testing framework that combines data flow graphs with genetic algorithms (GA) to generate optimum test cases. P. Palangpour, G.K. Venayagamoorthy, and S.C. Smith [8] presented a pipelined architecture for hardware particle swarm optimization (PSO) implementation to achieve much faster execution times than possible in software. A. Mansoor [9] developed AI technique based on genetic algorithm for the optimization of software test data. M. Syafrullah and N. Salim [10] proposed a new approach based on particle swarm optimization techniques in order to improve the accuracy of term extraction results. J. H. Andrews, T. Menzies, and F.Li [11] described a system that is based on a genetic algorithm (GA) to find parameters for randomized unit testing that optimize test coverage. J. CHANG and J. Pan [12] presented a Parallel Particle Swarm Optimization (PPSO) algorithm and a three communication strategies. D. Arora, A. S. Baghel [1] presented a method that uses genetic algorithm and particle swarm optimization for optimizing software testing by finding the most error prone paths in the program.

The method used here in this research is to calculate execution time for software modules, this could be achieved by using the Particle Swarm Optimization (PSO) techniques, as the each population in each iteration search for best execution time through particles in this population, and finally compare the best solution to produce the best execution time, also the use of parallel particle swarm optimization helps to run the populations and particles in distributed processing systems to help find the best solution in parallel, then selecting the best execution time. It is very crucial phase for any software to determine its quality and ability to meet requirements, which could be achieved through test this software, testing as a phase of software engineering process, literally takes about 40~50% of the development efforts in software houses [13].

It is noteworthy that life critical software could use more efforts and resources, if it is not tested perfectly, the software may cause dangerous consequences as timetable delays, cost overrun. Also software community aims to deliver high quality software to customers, to ensure that the software will run perfect with no delays in execution time, as this is the aim of this research is to calculate the execution time [14, 15, 16]. Also the proposed algorithm in this research is done automatically through a testing tool that produces the results of execution time, also trials have been done and the results of sample code is depicted below in implementation part.

The paper is organized as follows: in part II an introduction and brief description of PSO algorithm, in part III brief description of the two types of PPSO techniques, in part IV description of the proposed PSO and PPSO algorithms, followed by the implementation in part V, and then analysis and results in part VI, the conclusion and future work in part VII.

II. PARTICLE SWARM OPTIMIZATION

Modelling of swarms was initially proposed by Kennedy to simulate the social behaviour of fish and birds, the optimization algorithm was presented as an optimization technique in 1995 by Kennedy and Eberhart, PSO has particles which represent candidate solutions of the problem, each particle searches for optimal solution in the search space, each particle or candidate solution has a position and velocity. A particle updates its velocity and position based on its inertia, own experience and gained knowledge from other particles in the swarm, aiming to find the optimal solution of the problem [17].

The particles update its position and velocity according to the following Equation:

$$v_1^{k+1} = wv_1^k + c_1 rand_2 \times (pbest_i - s_i^k) + (gbest_i - s_i^k) \quad (1)$$

Where:

v_i^{k+1} = Velocity of agent i at iteration k,
w = Weighting function,
cj = Weighting factor,
rand = Random number between 0 and 1,
S_i^k = Current position of agent iteration k,
pbesti = Pbest of agent i,
gbesti = gbest of the group.

The weighting function used in Equation 1:

$$w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} \times iter \quad (2)$$

Where:

Wmax= Initial weight,
Wmin= Final weight,
itermax= Maximum iteration number,
Iter = Current iteration number.

According to more than ninety modifications are applied to original PSO [17, 18].

III. PARALLEL PARTICLE SWARM OPTIMIZATION

PSO is optimized to be implemented on distributed systems, the iterations and particles within each iterations of the PPSO are independent of each other, so results could be parallel analysed. PPSO could be divided into two types [19, 20, 21]:

1) Synchronous Parallel Particle Swarm Optimization

PSO parallel implementation is to simply evaluate the particles (solutions), or in other words the execution time produced within each iteration in parallel, without changing the overall logic of the algorithm itself. In this implementation, all particles within design iteration are sent to the parallel computing environment, and the algorithm waits for all the analyses to complete before moving to the next iteration. This implementation is referred to as a synchronous implementation. This method is used in this research [22, 23].

2) Asynchronous Parallel Particle Swarm Optimization Algorithm

Considering an asynchronous algorithm means that particles (solutions) or as mentioned before execution time produced in the next iteration are analysed before the current design iteration is completed. The goal is to have no idle processors as one move from one iteration to the next. [22]

The key to implementing an asynchronous parallel PSO algorithm is to separate the update actions associated with each point and those associated with the swarm as a whole. These update actions include updating the inertia value and the swarm and point histories. For the synchronous algorithm, all the update actions are performed at the end of each design iteration. For the asynchronous algorithm, researchers want to perform point update actions after each point is analysed and the swarm updates actions at the end of each design iteration. The parts of the algorithm that need to be considered when looking at the update actions are the velocity vector, and the dynamic reduction of the inertia value. [22]

The velocity vector is the centre point of any PSO algorithm. For each design point, the velocity vector is updated using the following dynamic properties for that point: the previous velocity vector; the current position vector; and the best position found so far. In addition, the updated inertia value and the best position for the swarm as a whole are also required. To do the velocity update in an asynchronous fashion, researchers need to update the position vector and the best position found so far for each design point directly after evaluating that point. For the best position in the swarm, researchers have two choices:

Use the best position in the current iteration, or use the best position found so far. To keep the best position for the swarm current when moving to the next design iteration, before the current iteration is completed, it is necessary to use the best position found so far rather than the best position in the current iteration. This setup allows the algorithm to update all required dynamic properties of the velocity vector directly after evaluating each design point, except for the inertia value. The inertia value is the only iteration level update required to compute the velocity vector and is updated at the end of each design iteration. The craziness operator is the only other iteration level update and is also performed at the end of each design iteration. [23]

The asynchronous algorithm is thus very similar to the synchronous algorithm, except that researchers update as much information as possible after each design point is analysed. The inertia is only applied when design iteration is completed. Of course, this could result in some points of the next design iteration being analysed before the inertia operator is applied for that design iteration. However, the influence on the overall performance of the algorithm seems to be negligible [24].

IV. PROPOSED PSO & PPSO ALGORITHMS

The proposed architecture is based on PSO and PPSO algorithms to calculate execution time depicted below in figure 1. Additionally, in order to evaluate the execution time for software module, a proposed PPSO (Parallel PSO) algorithm to calculate execution time also introduced and the recommended execution time strategy is determined for implementing this PPSO algorithm.

- The PSO Algorithm to Calculate Execution Time can be listed in following steps:

A. Initialize the population with N Particles. And Set iterations counter I = 0.

B. Apply Fitness function: Calculating the fitness value by calculating the percentage of this particle will share in minimizing the total processing time to find the optimal solution.

C. Compare the calculated fitness value of each particle with its (lbest). If current value is better than (gbest), then reset (gbest) to the current index in particle array. Select the best particle as (gbest).

D. Calculated fitness value among the neighboured particles in the network achieved so far in the iteration.

E. Update each Particle Velocity and position according to Eq. (1).

$$v_1^{k+1} = wv_1^k + c_1 rand_2 \times (pbest_i - s_i^k) + (gbest_i - s_i^k) \quad (1)$$

Where I= 0, 1, 2… M-1

To prevent the velocity from becoming too large, researchers set a maximum value to limit the range velocity as

$$-VMAX \leq V \leq VMAX$$

F. Cost function assigns the highest fitness value in the iteration and which has a current position (xi).

t = t +1.

G. End While.

H. Stop.

So these recursive steps continue until reaching the termination condition, and the termination condition achieved when the cost function finishes the execution, and finds the optimal time and solution.

In PPSO, computation time of PSO can be reduced with the parallel structure. Parallel processing aims at producing the same results achievable using multiple processors with the goal of reducing the run time. The same steps described in PSO will be applied, but in step (a) PPSO will define how many group of processors needed for the cost function to be executed, because it can be designed to be 2n sets.

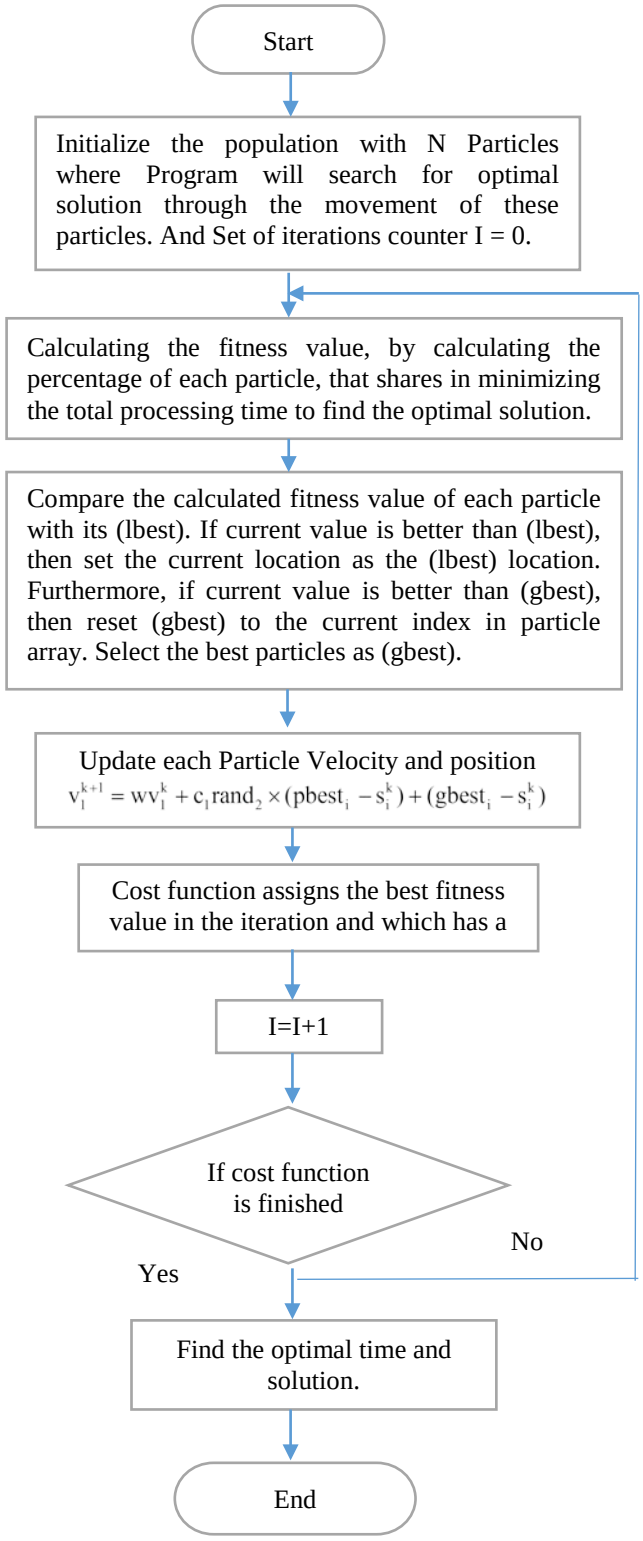

Fig. 1. Proposed PSO Based Algorithm to Calculate Execution Time

The performance of the Parallel PSO can be evaluated using Amdahl's Law Eq. [25].

Speedup (Sp) = $1/f_s + f_p/p$

Where:

fs= serial fraction of code

fp= parallel fraction of code

P= number of processors

Suppose serial fraction of code (0.5), parallel fraction of code (0.5) and number of processors (2, 4, 8, 16, 32, 64, 128, 256, 512, and 1024).

If P=2: Sp= 1 / (0.5+0.25) = 1.33

TABLE I. PPSO ALGORITHM

P	SP	Elapsed Time
2	1.33	0.02
4	1.60	0.017
8	1.79	0.012
16	1.89	0.008
32	1.96	0.005
64	2.00	0.003
128	2.00	0.002
256	2.00	0.001
512	2.00	0.001
1024	2.00	0.0003

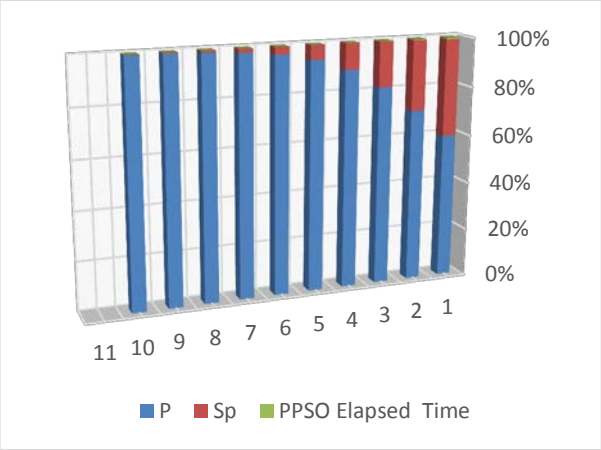

Fig. 2. PPSO Algorithm

This figure shows that the increase of number of processors, increase in speed results and decrease in elapsed time.

V. IMPLEMENTATION OF THE PROPOSED ALGORITHM

This section introduces implementation of MCWA, PSO and PPSO where cost function (CF) is:

```
CF: function z=Sphere(x)
    z=sum (x. ^2);
   end
```

This paper implementation proposed cost function uses MCWA, PSO and PPSO. The first implementation uses MCWA where results are shown below in table 2:

TABLE II. RESULTS OF MCWA

Iteration	CF	Elapsed Time(ET)
1	1	16.16
2	4	97.15
3	9	109.26
4	16	118.05
5	25	126.38
6	36	133.59
7	49	140.11
8	64	147.25
9	81	157.59
10	100	169.34

In table 2 each test case (iteration) to optimize the cost function but elapsed time increased and shown below in figure3:

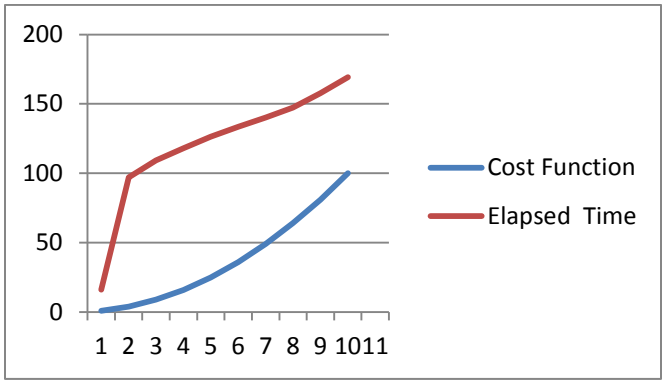

Fig. 3. Releationship between CF and ET in MCWA

In figure 3 shown relationships between CF and ET where found positive relationship between them.

The second implementation uses PSO where results are shown below in table 3:

TABLE III. RESULTS OF PSO

Iteration	CF	ET
1	58.73	0.04
2	24.48	0.07
3	15.00	0.10
4	5.85	0.13
5	5.85	0.16
6	5.85	0.19
7	5.34	0.21
8	2.73	0.24
9	2.73	0.27
10	2.73	0.30

In table 3 each test case (iteration) to optimize the cost function but elapsed time decreased compared with MCWA and shown in figure 4.

In figure 4 shown relationships between CF and ET where found inverse relationship between them.

The third implementation uses PPSO where results are shown in table 4.

In table 4 each test case (iteration) to optimize the cost function but elapsed time decreased compared with MCWA, PSO and shown in figure 5.

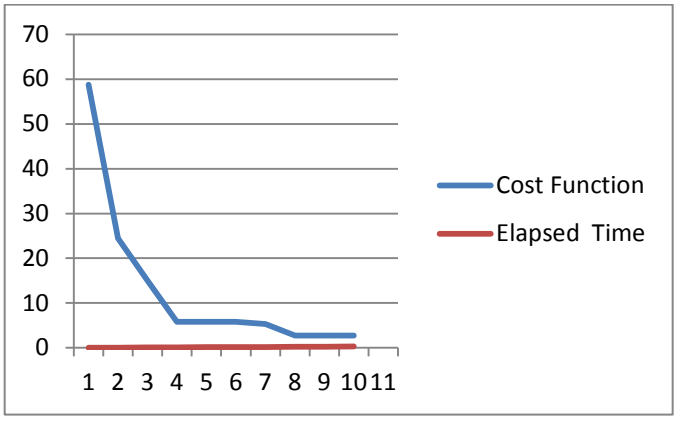

Fig. 4. Releationship between CF and ET in PSO

TABLE IV. RESULTS OF PPSO

x	P	Sp	CF	ET
1	2	1.33	58.73	0.02
2	4	1.60	24.48	0.017
3	8	1.79	15.00	0.012
4	16	1.89	5.85	0.008
5	32	1.96	5.85	0.005
6	64	2.00	5.85	0.003
7	128	2.00	5.34	0.002
8	256	2.00	2.73	0.001
9	512	2.00	2.73	0.001
10	1024	2.00	2.73	0.0003

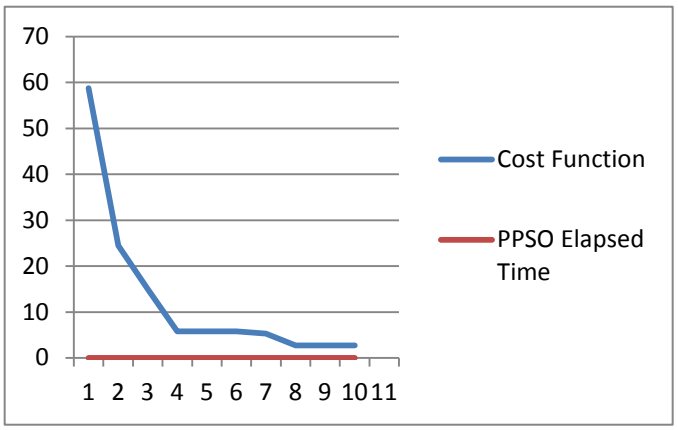

Fig. 5. Releationship between CF and ET in PPSO

In figure 5 shown relationships between CF and ET where found inverse relationship between them.

This paper introduces compared between PSO and PPSO where shown in figure 6.

In figure 7 shown inverse relationships between SP, PSO and PPSO Whenever an increase in speed occur where decreased in PSO and also more decreased in PPSO.

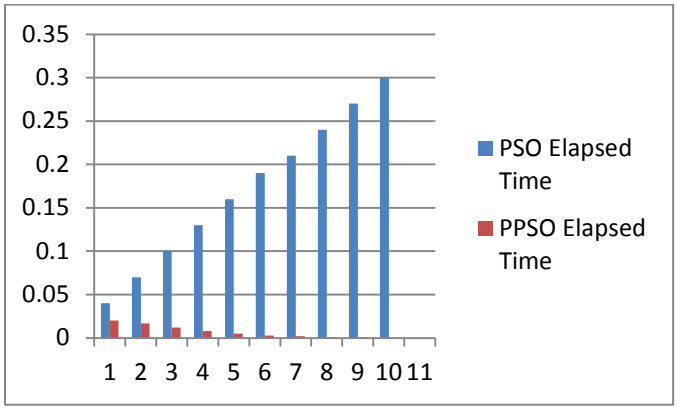

Fig. 6. Releationship between PSO and PPSO

In figure 6 shown relationships between PSO and PPSO where elapsed time in PPSO decreased compared with PSO. In figure 7 shown relationships between SP, PSO and PPSO.

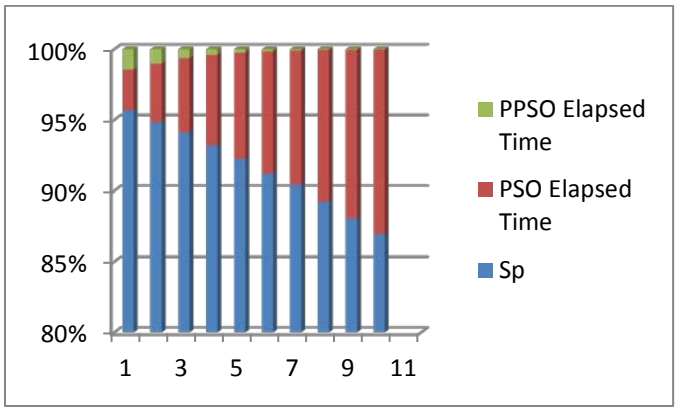

Fig. 7. Releationship between SP, PSO and PPSO

VI. ANALYSIS AND RESULTS

This paper presented the results of PSO and PPSO. In PSO, the results show inverse relationship between CF & ET, although the optimizing, the cost function elapsed time decreased compared with MCWA. In PPSO found that there is an inverse relationship between CF & ET although optimizing the cost function, but elapsed time decreased compared with MCWA and PSO. This paper shows the relationship between PSO and PPSO where elapsed time in PPSO decreased compared with PSO and shows inverse relationship between SP, PSO and PPSO Whenever an increase in speed occur it decreased in PSO and also time decreased more in PPSO.

VII. CONCLUSION AND FUTURE WORK

PSO is relatively recent heuristic approach; it is similar to PPSO in a way that they both are population based evolutionary algorithms.

The research presents the application of PSO and PPSO. The proposed research described the basic concepts of PSO and PPSO, calculating execution time for software modules for using PSO and PPSO and how they are useful in finding the optimal solution to the problem. Comparative study is done between both the algorithm where PPSO can be useful, and showing how PPSO overcome the drawback of PSO. This paper shows that PPSO algorithm is more efficient in speed and time compared with MCWA and PSO algorithm to calculate the execution time.

In future, the researchers aim to apply the proposed approach using ante colony optimization (ACO) and cat optimization (CO) and compare the results with other results produced from other evaluation techniques of swarm optimization algorithms. In addition, the researchers aim to enhance the proposed approach by examining more hybrid techniques to calculate execution time for software modules.

REFERENCES

[1] D.Arora, A.Baghel "Application of Genetic Algorithm and Particle Swarm Optimization in Software Testing "Journal of Computer Engineering , Volume 17, PP 75-78 ,2015.

[2] James H. Andrews, Tim Menzies, Felix C.H. Li, "Genetic Algorithms for Randomized Unit Testing", IEEE Transactions on Software Engineering, vol.37, no. 1, pp. 80-94, January/February 2011.

[3] Nagy Ramadan Darwish "Towards An Approach For Evaluating The Implementation Of Extreme Programming Practices" International Journal of Intelligent Computing and Information Science, Volume 13, PP 55-67,2013.

[4] C. Mao, "Generating Test Data for Software Structural Testing Based on Particle Swarm Optimization," Arabian Journal for Science and Engineering, vol. 39, pp. 4593-4607, 2014.

[5] A. Windisch, S. Wappler and j. Wegener, "Applying Particle Swarm Optimization to Software Testing," in Proc. 9th Conf. Genetic and evolutionary computation, 2007, pp. 1121-1128.

[6] R. Ding and H. Dong "Automatic Generation of Software Test Data Based on Hybrid Particle Swarm Genetic Algorithm," in Electrical & Electronics Engineering (EEESYM), 2012, pp. 670 - 673.

[7] A. S. Andreou, K. A. Economides and A. A. Sofokleous "An Automatic Software Test-Data Generation Scheme Based on Data Flow Criteria and Genetic Algorithms," in 7th Int. Conf. Computer and Information Technology, 2007.

[8] P. Palangpour, G.K. Venayagamoorthy, and S.C. Smith, "Particle Swarm Optimization: A Hardware Implementation," in Proc. Int. Conf. Computer Design, 2009.

[9] A. Mansoor, "Automated Software Test Data Optimization Using Artificial Intelligence," International Journal of Information and Communication Trends, vol. 1, pp. 1-60, 2014.

[10] M. Syafrullah and N. Salim, "Improving Term Extraction Using Particle Swarm Optimization Techniques," Journal of Computing, vol. 2, 2010.

[11] J. H. Andrews, T. Menzies, and F.Li, "Genetic Algorithms for Randomized Unit Testing," IEEE Transactions on Software Engineering, vol. 37, no. 1, 2011.

[12] J. CHANG and j. Pan, "A Parallel Particle Swarm Optimization Algorithm with Communication Strategies," Journal of Information Science and Engineering, vol. 21, pp. 809-818, 2005.

[13] Jovanovic and Irena,"Software TestingMethods and Techniques," May 26, 2008.

[14] Saswat Anand, Edmund Burke, Tsong Y. Chen, et al., "An Orchestrated Survey on Automated Software Test Case Generation" Journal of Systems and, Software, 2013.

[15] "Addressing Software Testing Costs, Complexity and Challenges: Sogeti UK's Annual TestExpo Survey", Survey Report, Sogeti Inc., 2013.

[16] Myers, Glenford J., IBM Systems Research Institute, Lecturer in Computer Science, Polytechnic Institute of New York, "The Art of Software Testing", Copyright 1979. by John Wiley & Sons, Inc.

[17] Aly, Walid; Yousif, Basheer; Zohdy, Bassem "A Deoxyribonucleic Acid Compression Algorithm Using Auto-Regression and Swarm Intelligence". Journal of Computer Science, 690-698, 9(6), 2013.

[18] Syafrullah M., Salim N., 2010, "Improving Term Extraction Using Particle Swarm Optimization Techniques", Journal of Computing, Vol. 2, Issue2, pages 116-120, 2010.

[19] Chang, J.F., Chu, S.C., Roddick, J.F., Pan, J.S., "A parallel particle swarm optimization algorithm with communication strategies", Journal of information science and engineering. 21, 809–818, 2005.

[20] Shu-Chuan, Chu et al. "Parallel Particle Swarm Optimization Algorithms With Adaptive Simulated Annealing", Springer, 31, 2006.

[21] Venter G, Sobieszczanki-Sobieksi J. A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. Journal of Aerospace Computing, Information, and Communication. 6th World Congresses of Structural and Multidisciplinary Optimization, 2005

[22] Şaban Gülcü, and Halife Kodaz, "A novel parallel multi-swarm algorithm based on comprehensive learning particle swarm optimization," Engineering Applications of Artificial Intelligence, vol. 45, pp. 33–45, 2015.

[23] R. V. Kulkarni and G. K. Venayagamoorthy, "Particle swarm optimization in wireless-sensor networks: a brief survey," IEEE Transactions on Systems, Man and Cybernetics Part C, vol. 41, no. 2, pp. 262–267, 2011.

[24] C. A. Voglis , K. E. Parsopoulos , I. E. Lagaris, "Particle swarm optimization with deliberate loss of information", Soft Computing - A Fusion of Foundations, Methodologies and Applications, v.16 n.8, p.1373-1392, August 2012.

[25] Herb Sutter, "The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software", Dr. Dobb's Journal, 30(3), March 2005.