Journal of Fundamental and Applied Sciences

ISSN 1112-9867

Available online at http://www.jfas.info

Research Article

Special Issue

SCRIPTING LANGUAGE DESIGN AND THE IMPLEMENTATION TEST FOR PSO-GA HYBRIDIZATIONS

S. Masrom[1], S. Z. Z. Abidin[2,*], N. Omar[2], Z. I. Rizman[3] and A. S. A. Rahman[4]

[1]Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Tapah, Perak, Malaysia

[2]Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Shah Alam, Selangor, Malaysia

[3]Faculty of Electrical Engineering, Universiti Teknologi MARA, 23000 Dungun, Terengganu, Malaysia

[4]Faculty of Science and Information Technology, Universiti Teknologi PETRONAS, Seri Iskandar, Perak, Malaysia

Published online: 15 January 2018

ABSTRACT

This paper introduces a new design of a set of scripting language constructs and the implementation test for the algorithms development. The design of the scripting language is presented in form of syntactic specification and Deterministic Finite Automaton (DFA). Based on the several algorithms of PSO-GA hybrids that have been developed with the scripting language constructs, the Characters of Code (COC) are measured in order to test the easiness of the programming language. The results show that across all algorithms, the scripting language is anticipated to enable easy programming which has been presented by the very less number of COC compared to the JAVA programming language. Furthermore,

Author Correspondence, e-mail: zaleha@tmsk.uitm.edu.my

doi: http://dx.doi.org/10.4314/jfas.v10i1s.65

based on observation from repeated experiments, the optimization results of all the algorithms developed with the scripting language are shown to be very accurate within the scale of results generated by JAVA codes.

Keywords: rapid algorithm; development; hybridization dynamic; parameterizations; scripting language.

1. INTRODUCTION

Improving Particle Swarm Optimization (PSO) with hybridization techniques has gained wide attention from many researchers [6, 10, 14-21]. However, implementing the hybrid algorithms involves some difficulties with a repetitive algorithm design and development. In most cases, to complete the tasks is time consuming. In order to accomplish the tasks easier and faster, the researchers should be provided with a software tool that provides ready-to-use design and implementation [3, 12].

To date, there exist software tools for different meta-heuristics techniques [2, 4-5, 7, 11] but very limited tools that enable PSO-GA hybridizations. While the software tools allow users to combine different kinds of meta-heuristics, deep and wider knowledge in programming and meta-heuristics algorithms is required as the software tools not specifically designed for PSO and GA algorithms.

Responding to the limitation of existing software tools, the aim of this paper is to introduce a set of scripting language constructs for the PSO-GA hybrids. The scripting language constructs were developed based on the proposed implementation frameworks that have been described in the previous paper [7].

The remaining content of this paper is organized as follows. In section 2, a comparison study among the existing software tools for meta-heuristics is provided. Then, the designs of the scripting language constructs are presented in section 3, followed with the evaluations in section 4 before concluding remarks in section 5.

2. SOFTWARE TOOLS FOR META-HEURISTICS

As mentioned in the previous section, software tools help users to easily design, develop and testing meta-heuristics. Table 1 lists some of the software tools.

Table 1. Comparison of software tools for meta-heuristics

Software Tools	Single	Hybrid
iOpt	√	X
Hotframe	√	√
Mallba	√	√
JEO	√	X
HeuristicLab	√	√
JSwarm	√	X
SwarmOps	√	X
MDF	√	√
ParadisEO	√	√
EASEA	√	X
EAML	√	X
PPCEA	√	X
ESDL	√	X

As single paradigm is simpler than meta-heuristics hybrids, the majority of the existing software tools were designed to be more applicable for the easy implementation of single meta-heuristics. Some of the software are JEO [2], EAML [11], iOpt [12], EASEA [3], JSwarm, SwarmOps and ESDL [9]. Some of these software supports more than one meta-heuristics, but each of them is independently executed without any sense of internal interactions that can enable low-level meta-heuristics hybridization. For example, iOpt support different meta-heuristics like Genetic Algorithm, Local Search and Simulated Annealing but the hybridization of these different algorithms involves extensive programming modifications. The software tools that support hybridization are Hotframe [5], Mallba [1], HeuristicLab [13], MDF [2] and ParadisEO [22]. While the existing software are flexible for the development of many kinds of meta-heuristics, they are not designed to be specific for PSO-GA hybrid, which led to major programming modifications.

3. THE SCRIPTING LANGUAGE DESIGN

In this part, the syntactic specifications of the scripting language constructs are described. As a scripting language, the arguments are simple facilitate fast scripting for commonly-used components, which are optional to any chosen parameters.

3.1. Syntactic Specification

In this part, the syntactic specifications of the scripting language constructs are described. As a scripting language, the arguments are simple facilitate fast scripting for commonly-used components, which are optional to any chosen parameters.

3.1.1. General Components of PSO-GA Hybridization

The scripting language specifications for the general algorithm components are defined as following Fig. 1.

```
SGMutation |SGCrossover | SGCrossMutation | SinglePSO
[Name ^AlgoName] [ENum ^ExperimentNum]
[Name ^AlgoName] [ENum ^ExperimentNum]
```

```
SEARCHSPACE [particle ^particle] [Dim ^particle dimension]
```

```
PROBLEM [^ProbName | userdefined] [min   ^Min]
```

Fig.1. General components specifications

The general components are comprised of the predefined framework [7] that has been used as a keyword for identifying the type of hybridizations. It can be either SGMutation, SGCrossover or SGCrossMutation to implement a single PSO algorithm, the keyword used to represent the algorithm is single PSO. The relevant parameters for the general specifications are the algorithm name that uses the keyword Name, experiment number with keyword ENum, iteration number as ITER and population size as PSize. These parameters are governed by string variable ^AlgoName, integer variable ^ExperimentNum, integer variable ^IterationNum and integer variable ^PopulationSize respectively.

Other related components are the search space and problem. The search space definition uses SEARCHSPACE keyword that consists of solutions representation governed by a boolean variable ^particle (default particle=true). Each particle has the dimension size Dim governed by integer variable ^particledimension. The Problem specifications begin with PROBLEM keyword with the parameters problem name governed by ^ProbName and problem objective by a boolean variable ^Min (default min=true). The variable ^ProbName calls the related predefined problem formulation available at the back-end software library. A more flexible option for problem definition is userdefined that permits user to insert new user-defined codes segment, providing programmers with the capability to incorporate new formulation of optimization problem.

3.1.2. PSO Update

The scripting language specifications for the general algorithm components are defined as following Fig. 2.

```
UPDATE [basic | constriction | inertia]
[c1 ^c1value] [c2 ^c2value]
[MaxP ^maxp] [MinP ^minp] [MaxV ^maxv][MinV ^minv]
```

```
constriction [^constrictionvalue] | [random #random] | [time-vary #time_vary] | [adaptive #adaptive]
```

```
Inertia [const    ^value] |    [random #random]    | [adaptive #adaptive]
```

Fig.2. PSO update specifications

The scripting language is designed for three common types of PSO update namely basic, constriction and inertia. The personal and global learning rates are governed by c1value and c2value respectively. The maximum and minimum position (MaxP and MinP) specify the bounded areas each particle can be positioned, while maximum and minimum velocity (MaxV and MinV) determines the limitation change for one particle can accelerate for each iteration. All these parameter values are governed by its specific real variable (^maxp, ^minp, ^maxv, ^minv).

Besides predefine value for constriction and inertia rates, formulation of dynamic parameterizations can also be used. As described in [7], the dynamic parameterizations can be calculated through #random, #time-vary and #adaptive functions.

3.1.3. GA Crossover

As given in Fig. 3, the crossover component has three related configurations which are crossover rate defined in Crate specification, crossover operation C_operation manipulated from #c_operation function and selection operation S_operation from #selection function.

```
Crossover [Crate ^value] [C_operation #c_operation] [S_operation #selection]
```

```
Crate[const ^value] | [random #random] [time-vary #time_vary] | [adaptive #adaptive]
```

Fig.3. Crossover specifications

3.1.4. GA Mutation

Two important specifications in mutation are mutation rate Mrate and mutation operation M_operation where value for mutation rate is optional to different constant and dynamic parameterizations. Formulation of mutation operation is governed by #m_operation function. The scripting language specifications for the mutation components are defined as following Fig. 4.

```
Mutation [Mrate ^value] [M_operation
```

```
Mrate[const ^value] | [random #random] [time-vary #time_vary] | [adaptive #adaptive]
```

Fig.4. Crossover specifications

3.1.5. Dynamic Parameterizations

There are variation formulations have been used for the dynamic parameterizations that uses time-vary and adaptive approaches. The formulations of each approaches are controlled by input parameters. The language specifications for the time-vary and adaptive parameterizations are written as the following Fig. 5.

```
time-vary [LD] | [NLD] | [LI] | [NLI]
adaptive [ISA] | [Speed] | [Ratio] | [Rank]
```

```
LD [^min    ^max] | NLD[^min   ^max   ^ n-value] |
LI [^min    ^max] | NLI[^min   ^max   ^ n-value]
```

```
ISA [^e_value    ^a_value] | Speed[^a_value   ^b_value] |
Ratio [] | Rank [^min ^max]
```

Fig.5. Dynamic parameters specifications

The time-vary formulations are linear increasing (LI), linear decreasing (LD), nonlinear increasing (NLI) and nonlinear decreasing (NLD). All time-vary parameterizations requires minimum and maximum value in relation to specific parameter. The parameter n is used in variation of non-linear time-vary (NLD, NLI) where n is a nonlinear modulation index in the range of between [min: max]. The adaptive parameterizations are developed based on finesses performance, which is optional to Individual Search Ability (ISA), Speed, Ratio or Rank formulation. ISA involves e and a parameters, Speed requires a and b parameters, while Rank formulation employs minimum and maximum similar to time-vary schemes. The e, a and b are a constant value close to zero.

3.2. Deterministic Finite Automaton (DFA)

The following Fig. 6 is the Deterministic Finite Automaton (DFA) of the proposed scripting language. Generally, DFA consists of five tuples (Q,Σ,δ,I,f), where each element in Q indicates one state, including the initial state I and the final state f. Besides, there exists different types of states that are classified according to $s,m,c \in Q$ where s1..s15 are representing the states of the single PSO components, c1..c5 are representing crossover states and m1..m4 for the mutation states. Σ denotes the finite set of input symbols in the language and δ is the transaction function $\delta : Q\Sigma \rightarrow Q$ from one state to another, which is represented in an arrow.

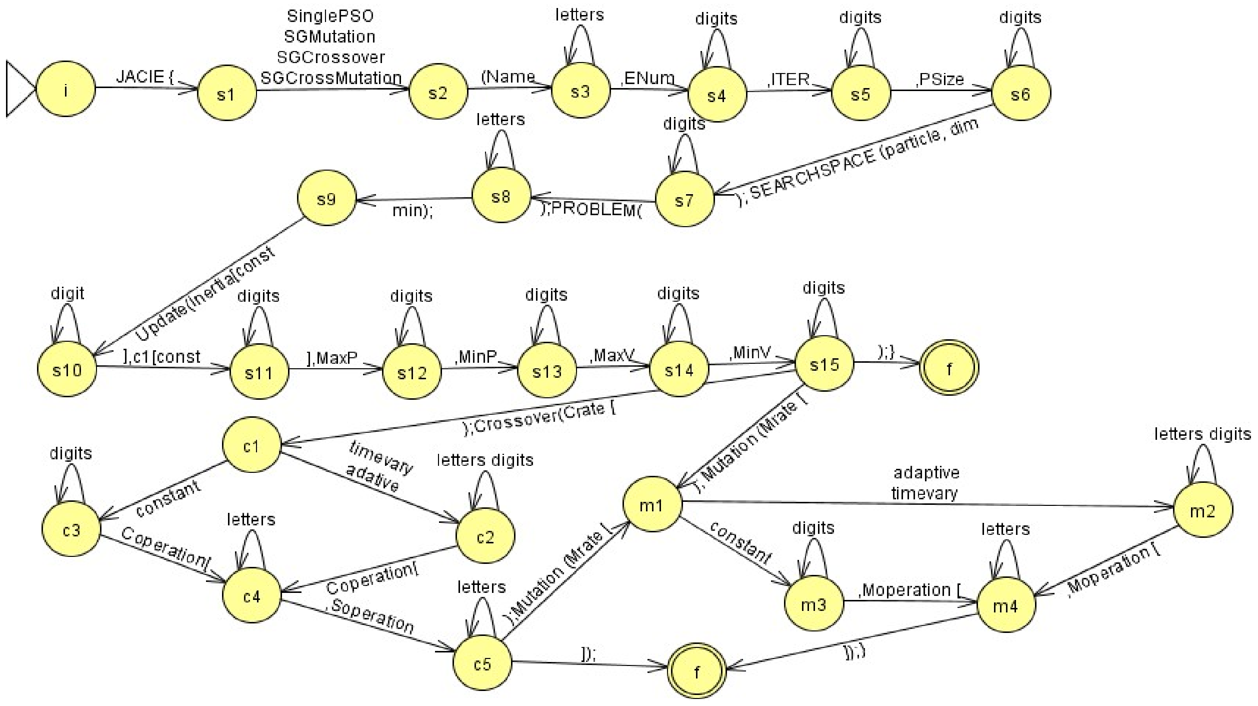

Fig.6. DFA of the scripting language

4. THE IMPLEMENTATION

An intermediate compiler is required to translate the scripting language into JAVA codes for the execution. The compiler software for the scripting language is an enhancement from the existing scripting language named JACIE (Java-based Authoring language for Collaborative Interactive Environments) version II. JACIE provides scripting language that is used to support rapid programming environment for distributive and collaborative applications. Since JACIE is a language based platform, the inclusions of new components as well as a new algorithm are always permitted. Hence, the compiler can be extended to translate new scripting language constructs for different domain applications.

Different algorithms have been developed with the scripting language, which were presented in [8]. As an example, the following Fig. 7 shows the codes for PSO-GA hybrids with both mutation and crossover.

The program begins with JACIE keyword to indicate the compiler is under JACIE architecture. To denote that the algorithm used both crossover and mutation, the SGCrossMutation keyword is used. At line 6, the crossover rate is determined through a linear decreasing parameterization.

```
JACIE{
SGCrossMutation(Name TCMR, ENum 40,ITER 3000, PSize 40);
SEARCHSPACE(particle, dim 30);
PROBEM(Sphere, min);
UPDATE (inertia[const 0.4],c1[const 2],c2[const2],MaxP 10.0, MinP -5, MaxV 10, MinV
  5);
Crossover (Crate[time-vary LD 0.6 1.0], C_ooperation[pbest], S_operation[roulettwheel]);
Mutation(Mrate const 0.1, M_operation[Gaussian]);
}
```

Fig.7. An example of the scripting language codes

5. THE EVALUATION

Several codes of the algorithms have been successfully executed with the scripting language constructs. The optimization results are observed as to test the validity of the algorithms developed with the scripting language. Firstly, the results of all different PSO-GA hybrids named Time-vary Crossover Rate (TCR), Time-vary Mutation Rate (TMR), Time-vary Crossover Rate (TMR) with different time-vary parameterizations (linear increasing (LI), linear decreasing (LD), non-linear increasing (NLI), non-linear decreasing (NLD)) that are developed with the scripting language constructs and JAVA can be compared from the Fig. 8.

The results show that all the LLH of PSO-GA with time-vary parameterization developed with the scripting language construct are able to produce very low mean best fitness within the scale of results generated by the JAVA codes. A very slightly different can be seen from TCR for f1(Sphere) and f3(Rastrigin), from TMR for f1(Sphere), f2(Rosenbrock) and f5(Griewank) as well as from TCMR for f4(Levy) and f7(Alpine). Furthermore, the comparison of mean best fitness for all the PSO-GA hybrids with Adaptive Crossover Rate (ACR), Adaptive Mutation Rate (AMR) and Adaptive Crossover and Mutation Rate (ACMR) with four adaptive approaches (SPEED, RATIO, RANK, ISA) is presented in the following Fig. 9.

Similarly, the results generated by all the PSO-GA hybrids developed with the two development approaches do not show extreme distinctions in adaptive parameterizations. Most of them are able to generate results within the range of results produced by the JAVA codes. From ACR, very slight difference can be seen on f2(Rosenbrock), f4(Levy) and f7(Alpine) while from TMR, no difference output has been observed. From ACMR, f1(Sphere) faces a small different of results between the two approaches.

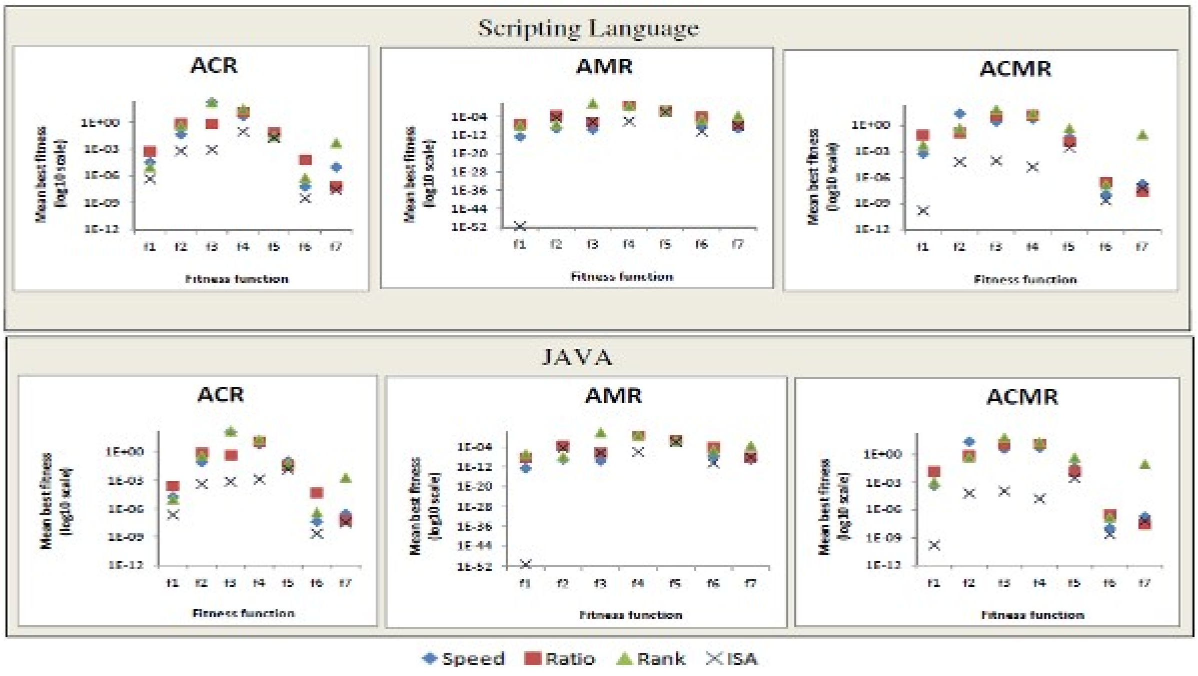

Fig.8. Comparison of mean best fitness between the scripting language and JAVA language development approaches for the LLH of PSO-GA with time-vary parameterization

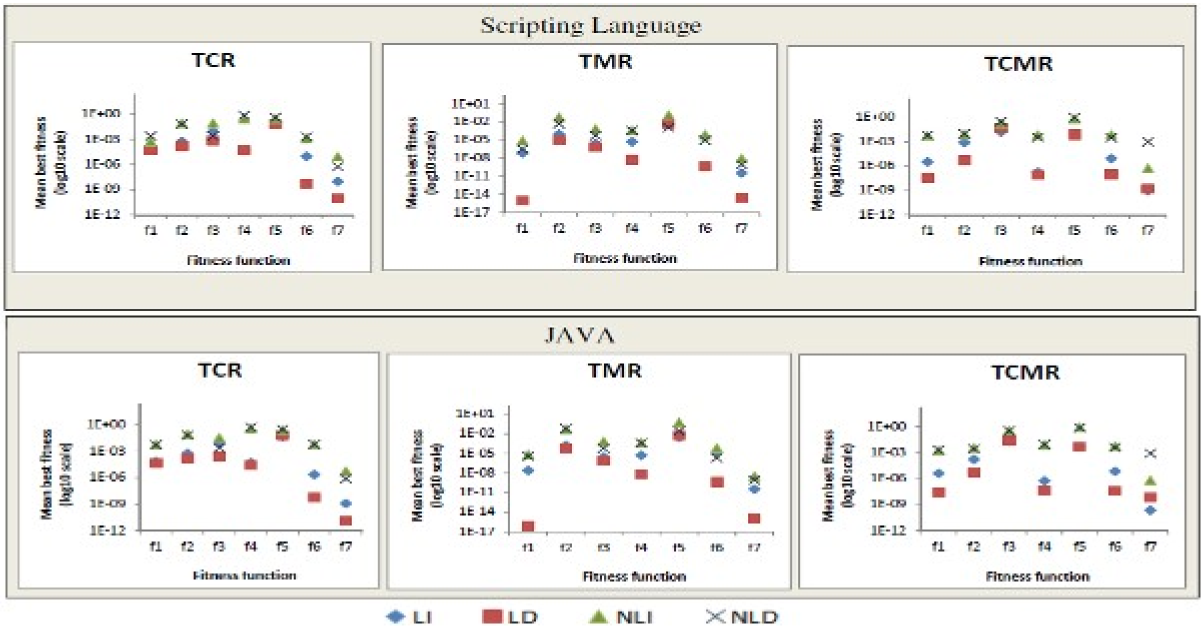

Fig.9. Comparison of mean best fitness between the scripting language and JAVA language development approaches for the LLH of PSO-GA with adaptive parameterization

The easiness test is valuable to indicate the effort in creating and reading the programs or algorithm descriptions. It uses volumetric measures which identify the lines of code characters of code (COC) in the programs. The COC of scripting language codes is compared with the relevant JAVA source codes that were manually developed in the extended JSwarm software framework. Only codes that an end user would need to write is included while compilers and predefined operators, comments as well as begin and end symbols () are not assessed. Besides, other codes used in the previous version of JACIE codes, which are not related to this work is also eliminated from this assessment. Table 2 shows the COC in each description.

Table 2. COC of different PSO-GA hybrids developed with the scripting language and JAVA

PSO-GA Hybrids	Scripting Language	Main JAVA
TMR	244	3163
TCR	269	2995
TCMR	323	3362
TIW	198	2662
AMR	245	3239
ACR	270	3410
ACMR	324	3575
AIW	199	2936

The algorithms are named as TMR to represent time-vary mutation rate, TCR for time-vary crossover rate, TCMR for time-vary crossover mutation rate, TIW for time-vary inertia weight, AMR for adaptive mutation rate, ACR for adaptive crossover rate, ACMR for adaptive crossover mutation rate and AIW for adaptive inertia weight. The TIW and AIW are the single PSO algorithm with time-vary and adaptive inertia weight respectively.

The results in Table 2 show that the scripting language programs consistently have shown less code than the main JAVA codes. This implies that to use the scripting language requires less effort to create and read the programs.

6. CONCLUSION

For all algorithms of PSO-GA hybrids, the proposed scripting language constructs have consistently shown very less code than the main JAVA codes. This implies that to use the scripting language constructs requires less effort. The scripting language constructs participated in enabling easy programming elements as the following:

It is wordless with simple and straightforward statements to be used and comprehended by different kind of users, for examples researcher and student. The statements and keywords are thoroughly designed to be closely alike to the relevant components of PSO and GA for examples Crossover, Mutation, problem and so on.

Additionally, all the scripting codes for all developed algorithms have a very less number of lines, words and characters than the main JAVA codes written in the software framework. It also has very small number of common symbols namely comma, semicolon, parenthesis and bracket. More importantly, the symbols and keywords are distinguishable each other in relation to different purposes. All the statements have consistent structure in such that it begins with the keyword, followed by an open parenthesis, parameter value, close parenthesis and ended with a semicolon.

7. ACKNOWLEDGEMENTS

The authors would like to thank Ministry of Education Malaysia and Universiti Teknologi MARA for their financial support to this project under FRGS Grant No. FRGS/1/2015/ICT01/UITM/02/1.

8. REFERENCES

[1] Alba E, Almeida F, Blesa M, Cabeza J, Cotta C, Díaz M, Dorta I, Gabarró J, León C, Luna J, Moreno L. MALLBA: A library of skeletons for combinatorial optimisation. In 8th European Parallel Processing Conference, 2002, pp. 63-73

[2] Arenas M G, Dolin B, Merelo J J, Castillo P A, Viana I, Schoenauer M. JEO: Java evolving objects. In 4th Annual Conference on Genetic and Evolutionary Computation, 2002, pp. 991-991

[3] Collet P, Lutton E, Schoenauer M, Louchet J. Take it EASEA. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo & H. P. Schwefel (Eds.), Parallel problem solving

from nature. Berlin: Springer-Verlag, 2000, pp. 891-901

[4] Dower S, Woodward C J. ESDL: A simple description language for population-based evolutionary computation. In 13th Annual Conference on Genetic and Evolutionary Computation, 2011, pp. 1045-1052

[5] Fink A, Voß S. Hotframe: A heuristic optimization framework. In S. Voß, & D. L. Woodruff (Eds.), Optimization software class libraries. New York: Springer, 2002, pp. 81-154

[6] Yassin I M, Zabidi A, Ali A M, Syahirul M, Md Tahir N, Zainol Abidin H, Rizman Z I. Binary particle swarm optimization structure selection of nonlinear autoregressive moving average with exogenous inputs (NARMAX) model of a flexible robot arm. International Journal on Advanced Science, Engineering and Information Technology, 2016, 6(5):630-637

[7] Masrom S, Abidin S Z Z, Omar N, Nasir K. Rapid prototyping for low-level hybridization of PSO-GA. In H. Fujita, A. Selamat, & H. Haron (Eds.), Frontiers in artificial intelligence and applications. Amsterdam: IOS Press, 2014, pp. 495-512

[8] Masrom S, Abidin S Z Z, Omar N. Easy and concise programming for low-level hybridization of PSO-GA. In International Conference on Intelligent Software Methodologies, Tools, and Techniques, 2014, pp. 1-14

[9] Talbi E. G. Metaheuristics: From design to implementation. New Jersey: John Wiley and Sons, 2009

[10] Thangaraj R, Pant M, Abraham A, Bouvry P. Particle swarm optimization: Hybridization perspectives and experimental illustrations. Applied Mathematics and Computation, 2011, 217(12):5208-5226

[11] Veenhuis C, Köppen M. XML based modelling of soft computing methods. In J. Benötez, O. Cordón, F. Hoffmann, & R. Roy, (Eds.), Advances in soft computing. London: Springer, 2003, pp. 149-158

[12] Voudouris C, Dorne R, Lesaint D, Liret A. iOpt: A software toolkit for heuristic search methods. In T. Walsh (Ed.), Principles and practice of constraint programming. Berlin: Springer-Verlag, 2001, pp. 716-729

[13] Wagner S, Affenzeller M. HeuristicLab: A generic and extensible optimization environment. In B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, & N. C. Steele, (Eds.),

Adaptive and natural computing algorithms. Vienna: Springer, 2005, pp. 538-541

[14] Masrom S, Rahman A S, Abidin S Z, Omar N, Rizman Z I. The implementation frameworks of meta-heuristics hybridization with dynamic parameterization. Journal of Fundamental and Applied Sciences, 2017, 9(6S):558-576

[15] Mahtar S N, Masrom S, Omar N, Khairudin N, Rahim S K, Rizman Z I. Trust aware recommender system with distrust in different views of trusted users. Journal of Fundamental and Applied Sciences, 2017, 9(5S):168-182

[16] Masrom S, Abidin S Z, Omar N, Rizman Z I. Software framework for optimization problems and meta-heuristics based on scripting language. Journal of Fundamental and Applied Sciences, 2017, 9(5S):33-48

[17] Ibrahim R, Leng N S, Yusoff R C, Samy G N, Masrom S, Rizman Z I. E-learning acceptance based on technology acceptance model (TAM). Journal of Fundamental and Applied Sciences, 2017, 9(4S):871-889

[18] Masrom S, Abidin S Z, Omar N, Rahman A S, Rizman Z I. Dynamic parameterizations of particle swarm optimization and genetic algorithm for facility layout problem. ARPN Journal of Engineering and Applied Sciences, 2017, 12(10):3195-3201

[19] Ibrahim R, Masrom S, Yusoff R C, Zainuddin N M, Rizman Z I. Student acceptance of educational games in higher education. Journal of Fundamental and Applied Sciences, 2017, 9(3S):809-829

[20] Indera N I, Yassin I M, Zabidi A, Rizman Z I. Non-linear autoregressive with exogeneous input (NARX) Bitcoin price prediction model using PSO-optimized parameters and moving average technical indicators. Journal of Fundamental and Applied Sciences. 2017, 9(3S):791-808

[21] Zabidi A, Yassin I M, Tahir N M, Rizman Z I, Karbasi M. Comparison between binary particles swarm optimization (BPSO) and binary artificial bee colony (BABC) for nonlinear autoregressive model structure selection of chaotic data. Journal of Fundamental and Applied Sciences, 2017, 9(3S):730-754

[22] Cahon S. ParadisEO: A platform for the design and deployment of parallel hybrid metaheuristic on clusters and grids. PhD thesis, France: University of Science and Technology

of Lille, 2005

How to cite this article:
Masrom S, Abidin SZZ, Omar N, Rizman ZI, Rahman ASA. Scripting language design and the implementation test for pso-ga hybridizations. J. Fundam. Appl. Sci., 2018, *10(1S), 869-883*.