OOPSF: Object Oriented Particle Swarm Framework

ISSN 2281-2911
Vol. 1 N. 3
May 2013

International Journal on Information Technology (IREIT)

Contents:

PWP
Praise Worthy Prize

International Journal on Information Technology (IREIT)

Managing Editor:

Santolo Meo
Department of Electrical Engineering
FEDERICO II University
21 Claudio - I80125 Naples, Italy
santolo@unina.it

The ***International Journal on Information Technology (IREIT)*** is a publication of the **Praise Worthy Prize S.r.l.**.
The Review is published bimonthly, appearing on the last day of January, March, May, July, September, November.

Published and Printed in Italy by **Praise Worthy Prize S.r.l.,** Naples, May 31, 2013.

OOPSF: Object Oriented Particle Swarm Framework

Walid M. Ali

Abstract – *This paper proposes an Object Oriented Particle Swarm Framework (OOPSF) that can be used to apply particle swarm intelligence on different optimization problems. The framework is designed using object oriented paradigm and implemented using Java programming language as an Application Programming Interface (API). Within a single framework, OOPSF attempts to abstract and encapsulate the basic concepts of particle swarm optimization (PSO) along with some modifications that were proposed by different researchers. The proposed API benefits from advantages of object oriented design including abstraction, encapsulation, inheritance and code reuse; finally the paper is concluded with a case study to solve a regression problem using the proposed framework.* ***Copyright © 2013 Praise Worthy Prize S.r.l. - All rights reserved.***

***Keywords**: Particle Swarm, Application Programming Interface, Java, Modeling, Soft Computing, Object Oriented*

Nomenclature

API	Application Programming Interface
OOPSF	Object Oriented Particle Swarm Framework
OOS	Object Oriented System
PSO	Particle Swarm Optimization
H-PSO	Co-operative Particle Swarm Optimization
CPSO	Co-operative Particle Swarm Optimization

I. Introduction

I.1. Soft Computing Algorithms

Soft computing algorithms are typically applied to complex NP-Complete problems where traditional mathematical techniques would fail to reach an optimum solution within reasonable time and using reasonable resources.

A number of soft computing algorithms are based on understanding a kind of nature intelligence and translating it into an applicable algorithm which can be implemented as a computer program, Lotfi Zada coined the term "soft computing" in the year 1994 [1].

Among the soft computing algorithms inspired by nature are: Genetic Algorithms [2], Bat Algorithm [3], Cuckoo Search [4], Ant Colony Optimization and Particle Swarm Optimization (PSO) [5]. PSO is the interest of this research, as the research intention is to model PSO as an object oriented system which consists of a number of objects interacting together.

I.2. Object Oriented System

An object is a general term that refers to a thing or a concept, Object Oriented System (OOS) is composed of software objects; these objects represent entities within a system in consideration, each object has a state and a behavior.

For software objects, the state is defined by the values of a defined set of variables and the behavior is executed by invoking methods within a defined set of methods. Objects are produced from classes where classes act as a template that defines the variables and methods.

An object oriented language is programming languages that have building structures and syntax rules to satisfy the object oriented paradigm. According to TIOBE index [6] which is a measure of popularity of programming language, 60% of popularity of programming languages goes to object-oriented languages rather than functional programming and Procedural programming Languages, with Java –the object oriented programming language- as the most popular.

In this research, Java is chosen as the implementation language of the proposed Object Oriented Particle Swarm Framework (OOPSF), not only for its popularity but for being a robust, platform independent and free programming language.

Object oriented systems share some basic useful concepts [7], these concepts promote code reuse and provides clear structures that are easy to maintain and modify. These concepts mainly include the following:

a. Inheritance: Relationship between classes which enable code reuse, a new class (denoted as subclass) inherits from an existing class (denoted as superclass).
b. Abstraction: Capturing the basic concept of an entity as a class while eliminating unnecessary details
c. Cohesion: Abstraction of only one concept in a class to ensure clarity and integrity.
d. Encapsulation: Language mechanism that offer abstraction and also prevents the misuse of class.

Manuscript received and revised April 2013, accepted May 2013

I.3. Problem Identification

In this research, the object orientation paradigm is utilized to translate the functional specifications of PSO to a conceptual design, the main entities in PSO are encapsulated as classes and the design is implemented using the java programming language as an Application Programming Interface with the name "Object Oriented Particle Swarm Framework" (OOPSF).

OOPSF can be used to tackle different swarm optimization problems. OOPSF abstracts and encapsulated some of the proposed modifications to the standard PSO, this enables it to serve as a test bed for existing PSO modifications, and even new proposed future modification to PSO can be easily implemented and added as amendment to the OOPSF.

Within different soft computing techniques, the proposed API selected PSO as it is more computationally efficient than other population based metheuriustic search algorithms like genetic algorithms (GAs). Rania and Babak [8] compared between GAs and PSO using benchmarks problems, the criteria of comparison was the quality of the proposed solution and the computation effort needed to find that solution, they concluded that PSO can be more computationally efficient and effective than the GAs, especially when attempting to solve unconstrained problems with continuous variables.

OOPSF is intended to be used by two categories of users, the first category include researchers who would like to examine the efficiency of different existing PSO techniques, they will be also capable to extend and modify the existing classes in OOPSF and test new optimization strategies based on PSO.

The second category includes researches who want to solve hard optimization problems using PSO.

The rest of this paper is organized after the current introductory section as follows: section 2 highlights the basic principles of PSO, section 3 provides the basic details of the OOPSF, section 4 presents a case study for a regression problem solved by OOPSF, the final section concludes research and proposes future work

II. Particle Swarm Optimization

II.1. Basics of Standard PSO

Particle swarm optimization is a population based optimization algorithm inspired by the nature [9], it was introduced in 1995 and since then it has acquired the attention of different researches. It is applied to solve problems related to various fields including electrical engineering [10]-[11], mechanical engineering [12], economic and financial problems [13], controller design [14] and scheduling [15].

PSO is inspired by social behavior of bird flocking. Within PSO, solutions are represented by particles, each particle models a bird flying within its flock aiming to find a better position. These particles exist within a generation, during the search, particles are updated and new generations are created until certain criterion is met.

The position of a particle (bird) represents the solution which we are trying to find, during the search, a particle (bird) changes its positions by updating its velocity.

Typically, PSO can be classified into two main categories: continuous PSO and binary PSO, in continuous PSO the particle position is a real value, the decision to how to update the velocity benefits from:-

i. Particle current velocity.
 (representing particle confidence in its inertia)
ii. Best position known by the particle.
(representing the effect of self-knowledge)
iii. Best position known by the whole generation.
 (representing the effect of social knowledge)

The updated position of the j^{th} particle in the i^{th} generation is calculated as follows [16]:

$$vel_j^i = \omega \times vel_j^{i-1} + c_1 \times rnd_1 \times \left(pbest_j^{i-1} - pos_j^{i-1} \right) + \\ + c_2 \times rnd_2 \times \left(gbest - pos_j^{i-1} \right) \tag{1}$$

$$pos_j^i = pos_j^{i-1} + vel_j^i \tag{2}$$

where:

i	Index of generation
j	Index of particle
pos_j^i	Position of j^{th} particle in the i^{th} generation.
ω	Inertia weight
vel_j^i	Velocity of j^{th} particle at the i^{th} generation
c_1, c_2	Balance factors
$pbest_j^i$	Best known j^{th} particle position from start Of search till the i^{th} generation
$gbest$	Best known position for all particles in all previous generations
rnd_1, rnd_2	Random numbers between 0 and 1

Binary PSO [17] is a basic variation of PSO where the positions can only be the binary values 0 or 1, there is no influence of particle memory or social influence, however the particle can search within its space and the particle velocity will be a probability distribution function for the position.

Lately a lot of attention had been directed to binary PSO and different optimization problems were attempted to be solved using it [17].

II.2. Limitations of Standard PSO

The basic PSO drawbacks [18] can be considered as follows:

i. PSO requires an exhaustive tuning for its different configuration parameters, variation in tuning values causes significant variations in performance.
ii. PSO has tendency to reach a semi optimum solution by premature convergence during the hunt for the optimum solution, this is mainly due to the deterioration of performance with the increase of dimensionality of search space.

These drawbacks had initiated a number of researches aiming to modify the standard implementation of PSO into a more efficient and robust search technique, some of the researches were directed towards solving the first drawback by having the tuning parameters as dynamic parameters which can change their values during the search or by tuning these parameters using a metaheuristic approach. The second drawback was tackled by innovative PSO techniques, three of them are describes in the next section.

II.3. Innovative PSO

Some of the new PSO techniques proposed by researchers within the last decade are:

a. *Co-operative PSO (CPSO):* More than one swarm exist, and each swarm is responsible for finding the optimum value of one of the variables of the search space, CPSO uses the particles from these swarms to build a particle that aims to satisfy the optimality of all variables of the search space [19]. Having “n” components to optimize means having “n” swarms each responsible to optimize a single component, the main implication in CPSO is how to calculate the fitness of a particle in each swarm, this is solved by creating a context vector which concatenate the best position found in each swarm into a single vector, this vector is used in swarm *j* by replacing the *j* component of each particle from the *j*th swarm while keeping other components as found in the a context vector and then calculating the fitness.
b. *Hybrid PSO:* A merge of PSO with other search technique, these techniques typically includes evolutionary algorithms, where selection, crossover and mutation operators are applied [20].
c. *Hierarchical PSO (H-PSO):* Particles influence in search is not identical, there is a rank for each particle, a higher rank means existing at an upper level in the hierarchy with a more effect on search, the ranks of particles are updated during the exploration for an optimum solution [21], in a typical implementation of H-PSO, a particle is influenced by the particle directly above it in hierarchy.

III. OOPSF: Object Oriented Particle Swarm Framework

The framework proposed by the author attempts to abstract and encapsulate the basic concepts of the PSO along with some of the basic modifications within a single structure, to accomplish this goal, object oriented design is utilized to abstract the different players' entities in the particle swarm environment into a number of classes.

The design of a class identifies what state can its objects have and what behavior can they do, in the implementation stage, the state is represented as variables and the behavior as methods. All classes are well encapsulated as their variables are declared private and this makes them inaccessible outside the class, accessor and mutator methods are provided as needed.

III.1. OOPSF Structure

For convenience, the related classes are grouped in certain package; OOPSF is designed using three packages as stated in Table I.

TABLE I
PACKAGES IN OOPSF

Package Name	Role in OOPSF
swarm.api	groups the general and abstract classes
swarm.api.continousSwarm	groups the general classes of continuous PSO
swarm.api.binarySwarm	groups the general classes of binary PSO

III.2. Basic Classes and Interfaces in OPSF

The main classes and interfaces in the framework are declared in Table II.

III.3. Role of the Main OOPSF Classes

a- Class SwarmProblem

This abstract class encapsulates any optimization problem that can be solved using PSO, users of OOPSF should create a subclass of this class and add specific variables and methods specific to their problem.

The class declares implementation of Interface FitnessCalculator but leaves the actual implementation to its subclasses.

b- Interface FitnessCalculator

This interface defines only one method "getFitness" that receives a swarm particle and returns its fitness as a double value.

c- Class RegressionModelingProblem

As a subclass of SwarmProblem, this class encapsulates an instance of a regression modeling problem, it implements the method "getFitness" to return a value related to the correct percentage of estimated data.

d- Class SwarmParticle

This class encapsulates a swarm particle which represents a candidate solution, each object of this class has a number, a fitness value and another variable that stores best fitness encountered by particle itself during search, moreover the class declares two static variables to store the best fitness encountered by all particles and the value of the dimension. In OOP, there is only one copy of static variables that is shared between all object, the class stores velocity and position for each particle as objects of relevant classes.

The class is abstract as no objects will be created directly from it, the class defines two important methods updateVelocity () and updatePosition (), these methods are abstract; they are implemented by the subclasses of class SwarmParticle to offer the specific implementation of PSO. Class SwarmParticle has two concrete subclasses: class BinaryParticle which encapsulate a swarm particle in binary PSO and class ContinuesParticle which encapsulates a swarm particle in a continuous PSO.

TABLE II
BASIC CLASSES AND INTERFACES IN OOPSF

Class/Interface identifier	Role in OOPSF
Configuration	An abstract class that encapsulates the settings used during search.
ContinousConfiguration	A subclass of class Configuration, encapsulates configuration of continuous Swarm optimization.
AdaptiveConfiguration	A subclass of class ContinousConfiguration that encapsulates dynamic configurations.
BinaryConfiguration	A subclass of class Configuration that encapsulates configuration of binary Swarm optimization.
IllegalConfigurationException	Encapsulates an exception that is thrown in case the configuration is inconsistent.
SwarmParticle	An abstract class that encapsulates a swarm particle which represents a candidate solution.
Position	An abstract class that encapsulates a position in a certain dimension.
Velocity	Encapsulate the velocity of a particle in a certain dimension
Generation	Encapsulates a group of swarm particles at certain iteration.
SwarmProblem	Encapsulate the optimization problem as a whole.
ContinuesParticle	A subclass of class SwarmParticle encapsulating a particle in continuous swarm optimization.
BinaryParticle	A subclass of class SwarmParticle encapsulating a particle in binary swarm optimization
Result	Encapsulate the result of the search
SwarmException	Encapsulate a runtime error that prevent swarm search from completion
FitnessCalculator	specify an interface for fitness calculation that any swarm problem must implement
BinaryPosition	A subclass of class Position encapsulating a position of a binary particle
ContinousPosition	A subclass of class Position a position of a continuous particle
RegressionModelingProblem	A subclass of class SwarmProblem and encapsulates regression modeling
HierarchyOperations	An interface that defines the operations of H-PSO
HierarchicalParticle	A swarm particle in H-PSO
CoOperativePSO	Encapsulates a group of swarms particles in CPSO
HybirdPSOGeneration	A subclass of Generation and offers the selection, crossover and mutation operators of Genetic algorithms

Class SwarmParticle attempts to abstract some of the basic variations of PSO, one of the problems that this class deals with is the problem of inactive particles[22], these particles stop exploring new solutions and continue to roam within very limited space, this can be noted when their velocity reaches a value near zero and the new positions will not vary a lot from the current position, these particles could be eliminated from the swarm and replaced by new fresh particles, this is accomplished in the class by overloading the method updateVelocity with a version that receives a double value as tolerance, if the new calculated velocity is less than tolerance, the search will remove the particle and insert a new fresh particle instead.

e- Class Position

This class encapsulates the position of a particle, an array of references from this class will be used by a particle swarm instance to store the positions in all dimensions.

f- Class Velocity

This class encapsulates the velocity of a particle; an array of references from this class will be used by a particle swarm instance to store the velocities in all dimensions.

g- Class Generation

This class encapsulate a group of swarm particles at a certain iteration, only one object of class generation is created as this class is designed using the singleton pattern, this object is mutated from one iteration to other, this implementation prevent exhaustive usage of memory.

The class defines a createInitialGeneration() method which will create the initial swarm particles in the initial generations, the class also defines update() method which is responsible of updating particles from one iteration to another and store variables that describes the generation, one of these variables is array of double values which stores the best fitness value in each iteration.

h- Class Configuration

This class encapsulates the configuration used to tune a PSO search; it includes the variables and methods common to both continuous PSO and binary PSO, the variables in this class include:

- Dimension of the problem.
- Number of particles.
- Maximum number of iterations.
- The title of each position.

i- Class ContinousConfiguration

This class encapsulates the configuration used to tune a continuous PSO search; it is a subclass of class Configuration, class ContinousConfiguration defines variables including:

- Inertia weight representing particle confidence in its inertia.
- Balance Factor1 representing particle confidence in its memory.
- Balance Factor2 representing particle confidence in social Knowledge.
- Minimum values for positions in all dimensions.
- Maximum values for positions in all dimensions.

j- Class AdaptiveConfiguration

Class AdaptiveConfiguration extends class ContinousConfiguration and encapsulate a configuration which values can change from iteration to iteration

The main drawback of PSO is the existence of a number of parameters that should be tuned efficiently to ensure the search will result in optimum solution, generally modification of the standard particle swam optimization are directed towards setting these values dynamically during the search rather than having their values as constants.

The idea of adaptive inertia weight was tackled in a number of researches, in [23], Deep had the initial value of w set to 0.9 and linearly decrease to 0.4, the intention of adaptive weight in his research was to prevent the premature convergence,

To implement this proposed modification, class AdaptiveConfiguration overrides the method getW() defined in its super class and returns an inertia weight value which is updated linearly from generation to generation, class AdaptiveConfiguration uses the inertia weight value as the initial value and decrease it linearly till a final value is reached by the end of run.

Not only the inertia weight need be adaptive, The range set for the valid value of a position also affects the quality of the search, one of the modification applied to the standard swarm algorithm is by modifying this range through the run dynamically [24].The idea is captured in the proposed framework by having a method in this class that receives the index of the dimension and the new updated minimum and maximum constraints values.

k- Interface HierarchyOperations

This interface defines the methods that should be implemented by a swarm particles executing a HPSO, the interface defines methods to access and mutate a rank for each particle, and to access the direct upper particle in hierarchy, another version of updateVelocity method is also defined which receive a particle as an argument.

l- Class HierarchyParticle

This class is a subclass of class ContinuesParticle and declares to implement the interface HierarchyOperations; the class also defines a variable rank to store the rank of each particle.

m- Class CoOperativePSO

This class encapsulates CPSO, as in CPSO we have multiple swarms, each swarm is represented as a generation with its own configuration. CPSO is an aggregation of an array of objects from class Generation and an array of objects from class Configuration, the class also declares a swarm particle which will store the context vector.

Fig. 1 and Fig. 2 show UML class diagrams for some of the OOPSF classes.

IV. Sample Case Study

For the purpose of demonstrating and testing our proposed framework, OOPSF is used to solve the problem of parameters estimation of a regression model for the annual employment in the united states from 1890 to 1970 [25], the linear regression model proposed is illustrated in the following equation:

$$\begin{aligned} Y(K) &= a_0 + a_1 \times Y(K-1) + a_2 \times Y(K-2) + \\ &+ a_3 \times Y(K-3) + a_4 \times Y(K-4) \end{aligned} \tag{3}$$

where:
$Y(k)$: value of data series at instance k;
a_0, a_1, a_2, a_3 and a_4: model parameters.

The role of OOPSF is to estimate the values of these parameters, this is done by the following steps:-

i. Create an object from class RegressionModelingProblem
ii. Create an object from class ContinousConfiguration
iii. Create an object from class Generation
iv. Call the method : createInitialGeneration() defined in class Generation
v. Call the method: update() defined in class Generation, this method will continue to run until a certain criteria is met, then the final result is declared.

The results of executing PSO using OOPSF are reported as follows:

- The best value of fitness reached is 94.1.
- The values of the coefficients declared by OOPSF are:

a_0=0.86, a_1=0.63, a_2=0.83, a_3=0.05, a_4=-0.45

Fig. 3 shows the value of the best fitness for each iteration.

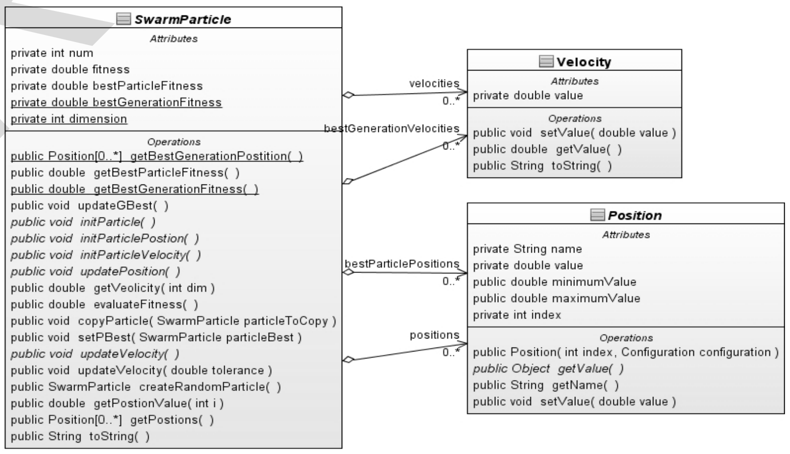
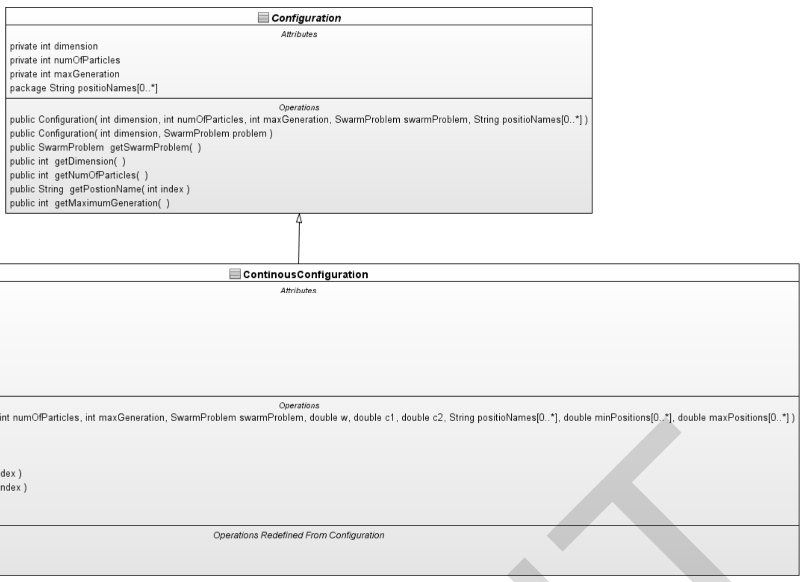

SwarmParticle
Attributes
private int num
private double fitness
private double bestParticleFitness
private double bestGenerationFitness
private int dimension
Operations
public Position[0..*] getBestGenerationPostition( )
public double getBestParticleFitness( )
public double getBestGenerationFitness( )
public void updateGBest( )
public void initParticle( )
public void initParticlePostion( )
public void initParticleVelocity( )
public void updatePosition( )
public double getVeolicity(int dim)
public double evaluateFitness( )
public void copyParticle(SwarmParticle particleToCopy)
public void setPBest(SwarmParticle particleBest)
public void updateVelocity( )
public void updateVelocity(double tolerance)
public SwarmParticle createRandomParticle( )
public double getPostionValue(int i)
public Position[0..*] getPostions( )
public String toString( )
velocities
0..*
bestGenerationVelocities
0..*
bestParticlePositions
0..*
positions
0..*
Velocity
Attributes
private double value
Operations
public void setValue(double value)
public double getValue( )
public String toString( )
Position
Attributes
private String name
private double value
public double minimumValue
public double maximumValue
private int index
Operations
public Position(int index, Configuration configuration)
public Object getValue( )
public String getName( )
public void setValue(double value)

Fig. 1. UML Class Diagram for class SwarmParticle



```
Configuration
Attributes
private int dimension
private int numOfParticles
private int maxGeneration
package String positioNames[0..*]
Operations
public Configuration( int dimension, int numOfParticles, int maxGeneration, SwarmProblem swarmProblem, String positioNames[0..*] )
public Configuration( int dimension, SwarmProblem problem )
public SwarmProblem  getSwarmProblem(  )
public int  getDimension(  )
public int  getNumOfParticles(  )
public String  getPostionName( int index )
public int  getMaximumGeneration(  )

ContinousConfiguration
Attributes
private int dimension
private int numOfParticles
private double w = 0.5
private double c1 = 0.5
private double c2 = 0.5
private double minPositions[0..*]
private double maxPositions[0..*]
Operations
public ContinousConfiguration( int dimension, int numOfParticles, int maxGeneration, SwarmProblem swarmProblem, double w, double c1, double c2, String positioNames[0..*], double minPositions[0..*], double maxPositions[0..*] )
public double  getW(  )
public double  getC1(  )
public double  getC2(  )
public double  getPostionMinmumValue( int index )
public double  getPostionMaximumValue( int index )
public double[0..*]  getMinmumPostions(  )
public double[0..*]  getMaximumPostions(  )
Operations Redefined From Configuration
public int  getDimension(  )
public int  getNumOfParticles(  )
```

Fig. 2. Uml Diagram of class Configuration and class AdaptiveConfiguration

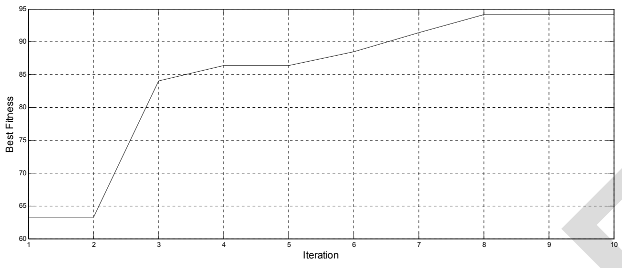


Fig. 3. Best Fitness versus Iteration

V. Conclusion and Future Work

The author presented an Object Oriented Particle Swarm Framework (OOPSF) for particle swarm optimization as a Java Application Programming interface. OOPSF is based on object oriented principles, and encapsulate the standard application of PSO with some of the main modifications recommended by researched to the PSO. OOPSF is not limited to solve a certain type of PSO optimization problems, it can be easily adapted to solve various problems.

Problems would only have to be modeled as a sub class (child) of the parent SwarmProblem class which is provided in the API. Although not all the variations and modifications of PSO are included in the proposed framework, innovative ideas like PSO with local neighborhood or adding an acceleration effect to PSO can easily be implemented within the framework. A demonstration case study for solving a time series regression modeling problem using OOPSF is presented to demonstrate the usage and efficiency of the API, the framework succeeded in finding satisfactory results on reasonable duration. In the future, users of the framework can easily extend classes and modify its behavior by overriding existing methods or defining new classes and new methods, different new techniques can be evaluated.

References

[1] L.A. Zadeh, Soft Computing and Fuzzy Logic. *IEEE Software*, *vol. 11* n.6, 1994, pp. 48-56.

[2] W. Gao, Study on genetic algorithm and evolutionary programming. *The 2nd IEEE International Conference on Parallel Distributed and Grid Computing (PDGC)*, Dec 6-8, 2012, Coimbatore, India.

[3] X. Yang, A New Metaheuristic Bat-Inspired Algorithm, In J. R. Gonzalez et al (Eds.), *Studies in Computational Intelligence*, (Berlin: Springer, 2010, 65-74).

[4] X. Yang, S. Deb, Cuckoo search via Lévy flights. *World Congress on Nature & Biologically Inspired Computing, Dec 9-11,*2009, Coimbatore, India

[5] G. Fornarelli, *Swarm Intelligence for Electric and Electronic Engineering* (IGI Global,2012).

[6] TIOBE Programming Community Index for March 2013 from http://www.tiobe.com/index.php/content/paperinfo/tpci /index.html. 2013.03.028.

[7] M. Weisfeld , *The Object-Oriented Thought Process* (Addison-Wesley Professional, 2013).

[8] H. Rania, C. Babak, A comparison of particle swarm optimization and the genetic algorithm. *Structures, Structural Dynamics, and Materials Conference*, April 18-21,2005, Austin, Texas.

[9] E. Chang, J. Singh, Prediction of Short Term Traffic Variables Using Intelligent Swarm-Based Neural Networks, *IEEE Transactions on Control Systems Technology vol. 21*, 2013, pp.1942-1945.

[10] D. Rocha, J. Tome, A discrete evolutionary PSO based approach to the multiyear transmission expansion planning problem considering demand uncertainties. *International Journal of Electrical Power & Energy Systems, vol. 45* n. 1, 2013, pp.427-442.

[11] R. Bansal, P. Sehgal, Using PSO in a spatial domain based image hiding scheme with distortion tolerance. *Computers & Electrical Engineering*, 2 Feb 2013, from http://dx.doi.org/10.1016/ j.compeleceng. 2012.12.021

[12] L. Yanbin, G. Youhua, Optimum design for beam-pumping unit based on chaos particle swarm optimization algorithm, *Second International Conference on Mechanic Automation and Control Engineering (MACE)*, July 15-17 ,2011,pp. 59635965

[13] C. Chun-Mei, Comparative study of financial distress prediction via optimized SVM. *International Conference on Machine Learning and Cybernetics,* Jul 15- 17, 2012, Shaanxi, China.

[14] M. Qian, G. Tan, Z.Ma, H.Zhang. Improved Chaotic Particle Swarm Optimization Algorithm and Its Application in the Design of Robust Controller. *International Journal on Information Technology*, *vol* .7 n. 1 (Part A), 2012, pp. 23-30

[15] Q. Dong, J.Lu, Y.Gui. Integrated Production Planning and Scheduling Model and Algorithm for Permutation Flowshop, *International Journal on Information Technology, vol* .7 n. 1 (Part B), 2012, pp. 374-381

[16] D. Merkle, *Swarm Intelligence: Introduction and Applications* (Springer**,**2010**)**

[17] A. Soroudi, M. Afrasiab, Binary PSO-based dynamic multi-objective model for distributed generation planning under uncertainty. *Renewable Power Generation, vol.6* n. 2, 2012, pp.67-78.

[18] Particle Swarm Optimization, Methods, Taxonomy and Applications, *International Journal of Computer Theory and Engineering, vol. 1* n. 4, 2009,pp.1793-8201

[19] D.K. Behera, Cooperative swarm based clustering algorithm based on PSO and k-means to find optimal cluster centroids. *National Conference of Computing and Communication Systems (NCCCS)*,Nov 21-22, 2012, West Bengal, India

[20] Y. Yan, A Multi-swarm Based Hybrid Optimization Algorithm in Dynamic Environments. *Second International Conference on Computer Modeling and Simulation, Jan 7-9.*, 2010, Mumbai, India

[21] L. Cheng-Jian, A self-organizing neural network using hierarchical particle swarm optimization. *The International Joint Conference on Neural Networks (IJCNN),* June 10-15, 2012, Brisbane ,Australia

[22] L. Yun, H.S. Chung, Adaptive Particle Swarm Optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, *vol.39,*n. 6, 2009,pp.1362-1381

[23] K. Deep, P. Chauhan, M. Pant, A new fine grained inertia weight Particle Swarm Optimization. *Congress on Information and Communication Technologies*, Dec 11-14, 2011, Mimbai, India.

[24] J. Zh, and X. Cai,. A guaranteed convergence dynamic double particle swarm optimizer. *Congress on Intelligent Control and Automation,June 15-19* 2004, Hangzhou, China

[25] DataMarket, Annual employment. From http://datamarket.com/data/set/22v2/annual-employment-us-1890-to-1970 .2013.02.012.

Authors' information

Walid Mohamed Aly has acquired his Ph.D. from faculty of Engineering, Alexandria University, Egypt. He is currently working as associate professor at the College of Computing and Information Technology (CCIT), Arab Academy for Science, Technology & Maritime Transport (AASTMT). His research interests include intelligent systems, soft computing, modeling and simulation and machine learning.
E-mail: walid.ali@aast.edu

International Journal on Information Technology (IREIT)

Aims and scope

The ***International Journal on Information Technology (IREIT)*** **(www.praiseworthyprize.com/ireit.htm)** is a peer-reviewed journal that publishes original theoretical and applied papers on all aspects regarding Information Technology. It is intended to be a cross-disciplinary international journal aimed at disseminating results of research on information technology. The topics to be covered include but are not limited to:
strategy, infrastructure, human resources, sourcing system development and implementation, communications, technology developments, technology futures, national policies and standards, Emerging advances in IT and new applications, Process Automation, Intelligent Mechatronics and Robotics, Intelligent Automation, Intelligent Social Agents, Intelligent Control, Intelligent Information Systems, Industrial Applications of Artificial Intelligence, intelligent electronic devices and systems, reengineering tools and methodologies, Geographical Information Systems/ Global Navigation Satellite Systems (GIS/GNSS), Information Technology in Healthcare, Strategic Decision Support Systems, Natural Language Interfaces to Intelligent Systems, Coordination in Multi-Agent Systems, Agent Oriented Modelling and Development, Process improvement tools, Business process support systems, Management Information Systems, Security and Information Assurance, e-commerce, information technology and electrical, mechanical, chemical, industrial, civil and communications applications.

Instructions for submitting a paper

The journal publishes invited tutorials or critical reviews; original scientific research papers (regular papers), letters to the Editor and research notes which should also be original presenting proposals for a new research, reporting on research in progress or discussing the latest scientific results in advanced fields; short communications and discussions, book reviews, reports from meetings and special issues describing research on all aspects regarding Information Technology.
All papers will be subjected to a fast editorial process.
Any paper will be published within two months from the submitted date, if it has been accepted.

Papers must be correctly formatted, in order to be published.
Formatting instructions can be found in the last pages of the Review.
An ***Author guidelines*** template file can be found at the following web address:
www.praiseworthyprize.com/Template_of_IREIT.doc

Manuscripts should be sent via e-mail as attachment in .doc and .pdf formats to:

editorialstaff@praiseworthyprize.com

The regular paper page length limit is defined at **15** formatted Review pages, including illustrations, references and author(s) biographies.
Pages 16 and above are charged 10 euros per page and payment is a prerequisite for publication.

Abstracting and Indexing Information:

Academic Search Complete - EBSCO Information Services
Cambridge Scientific Abstracts - CSA/CIG
Index Copernicus (Journal Master List)

Praise Worthy Prize

2281-2911(201305)1:3;1-2