Engineering Applications of Artificial Intelligence 54 (2016) 62–79

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Yin-Yang-pair Optimization: A novel lightweight optimization algorithm

Varun Punnathanam, Prakash Kotecha *

Department of Chemical Engineering, Indian Institute of Technology Guwahati, Guwahati 781039, Assam, India

ARTICLE INFO

Article history:
Received 20 November 2015
Received in revised form
18 April 2016
Accepted 19 April 2016
Available online 25 May 2016

Keywords:
Yin-Yang-Pair Optimization
Evolutionary computation
Single objective optimization
Metaheuristic
Congress on Evolutionary Computation 2013

ABSTRACT

In this work, a new metaheuristic, Yin-Yang-Pair Optimization (YYPO), is proposed which is based on maintaining a balance between exploration and exploitation of the search space. It is a low complexity stochastic algorithm which works with two points and generates additional points depending on the number of decision variables in the optimization problem. It has three user defined parameters that provide flexibility to the users to govern its search. The performance of the proposed algorithm is evaluated on the set of problems used for the Single Objective Real Parameter Algorithm competition that was held as part of the Congress on Evolutionary Computation 2013. The results are compared with that of other traditional and recent algorithms such as Artificial Bee Colony, Ant Lion Optimizer, Differential Evolution, Grey Wolf Optimizer, Multidirectional Search, Pattern Search and Particle Swarm Optimization. Based on nonparametric statistical tests, YYPO is shown to provide highly competitive performance relative to the other algorithms while having a significantly lower time complexity. In addition, the performance of YYPO is showcased on three classical constrained engineering problems from literature.

1. Introduction

Metaheuristics are frequently employed to solve optimization problems in various fields due to their inherent advantages over traditional techniques (Deb, 2001). Many techniques such as Genetic Algorithm (GA) (Holland, 1992) and Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995) are well established in literature, and variants are regularly developed (Khatib and Fleming, 1998; Ratnaweera et al., 2004; Gülcü and Kodaz, 2015; Tsai, 2015; Kim et al., 2007; Chatterjee and Siarry, 2006; Moayedikia et al., 2015; Idris et al., 2015) to improve or modify certain aspects of the algorithm. This is possible because most metaheuristics are remarkably simple and very flexible. Additionally, being derivative-free, these techniques do not require any prior knowledge on the physics of the problem and hence are attractive options for a wide range of problems (Gosselin et al., 2009; Ramteke and Srinivasan, 2012; Lee et al., 2008; Babu and Angira, 2006; Li and Shao, 2016; Zhao et al., 2016; Ling et al., 2016; Thomas et al., 2015). Single objective metaheuristics also form the basis for more complex algorithms such as multi-objective optimization algorithms. Several popular single objective algorithms have been extended to accommodate multiple objectives in literature (Mirjalili et al., 2016; Deb et al., 2002; Dasheng et al., 2007; Chalermchaiarbha and Ongsakul, 2012; Agrawal et al., 2006).

The No Free Lunch theorem (Wolpert and Macready, 1997) states that no single algorithm can perform well on every optimization problem, encouraging the development of new metaheuristics. These techniques are generally inspired from various everyday phenomena and are predominantly nature inspired. Few such nature inspired algorithms include the Ant-Lion Optimizer (ALO) (Mirjalili, 2015) mimicking the hunting mechanisms of antlions, Artificial Bee Colony (ABC) (Karaboga and Basturk, 2007) based on the foraging behaviour of honey bees, Grey Wolf Optimizer (GWO) (Mirjalili et al., 2014) based on the hierarchy and hunting mechanism of a pack of grey wolves, Differential Evolution (DE) (Storn and Price, 1997) based on the principle of evolution (similar to GA) and Cuckoo Search (Yang and Deb, 2009) simulating the behaviour shown by some cuckoo species in laying their eggs in the nests of other birds. Additionally, various other algorithms that are inspired from miscellaneous phenomena are the Teaching-Learning Based Optimization (Rao et al., 2011) which simulates the classroom environment, the Jaya algorithm (Rao, 2016) which attempts to "approach" the best solution while "moving away" from the worst solution, the Chaotic Golden Section Search Algorithm (Koupaei et al., 2016) based on chaotic maps and the golden section search, the Gravitational Search Algorithm (Rashedi et al., 2009) based on the law of gravity and mass

* Corresponding author.
E-mail address: pkotecha@iitg.ernet.in (P. Kotecha).

http://dx.doi.org/10.1016/j.engappai.2016.04.004
0952-1976/

interactions, Simulated Annealing (SA) (Kirkpatrick et al., 1983) based on the annealing process in metallurgy and Biogeography-based Optimization (Simon, 2008) based on the distribution of biological species over geographies. All the algorithms mentioned above except SA work on a set of solutions (commonly known as population) which communicate among each other to determine an optimal solution. Conversely, SA works on a single solution, and attempts to improve on it over iterations.

The Yin-Yang Algorithm (YYA) (Tam et al., 2011) was developed specifically for the travelling salesman problem and is based on transformation operators from the Chinese classic text *I Ching*. It is a population based technique which utilizes the mutation and selection operators of GA, while employing a transformation operator based on the hexagram transformations of *I Ching* in place of crossover. YYA shows similarities with GA in many levels, but it bears no resemblance with the algorithm proposed in this work. Although both the algorithms have similar names, the similarities end there as the proposed algorithm is not population based and has been developed for bounded real parameter single objective optimization problems as opposed to the travelling salesman problem (although the proposed algorithm may be extended for the same).

Most metaheuristics model a specific phenomenon or mechanism based on which they tackle optimization problems. On the contrary, the Yin-Yang-Pair Optimization (YYPO) algorithm is not based on any specific mechanism or physical event but is designed to explicitly balance exploration and exploitation and thus attempts to be a realization of the Yin Yang philosophy of balance between conflicting forces. Thus, YYPO is the authors' implementation of how the contradicting behaviours inherent in an evolutionary optimization algorithm (exploration and exploitation) can be balanced such that they complement each other to effectively determine the optimal solution. The algorithm is designed for bounded real parameter unconstrained single objective optimization, although the framework may be extended to accommodate constrained as well as mixed integer problems by incorporating minor modifications. For instance, the algorithm can handle constrained as well as mixed integer problems by employing the appropriate techniques from literature (Deb and Agrawal, 1999; Deep et al., 2009).

The rest of this article is structured as follows. The algorithm is described in detail in Section 2 along with a study of its behaviour on a sample problem. Section 3 briefly describes the other optimization algorithms which are used for a comparative performance analysis. Section 4 discusses the benchmark problems along with the test characteristics which are used for studying the performance of YYPO. This performance, including the algorithm complexities, is discussed and compared with that of the other algorithms in Section 5. The performance of YYPO on constrained engineering problems are presented in Section 5 as well. The work is summarized in Section 6 along with a discussion on future directions.

2. Algorithm description

In this section, we discuss the inspiration for the proposed algorithm along with its detailed modelling. Additionally, the behaviour of the algorithm is studied on the two dimensional Rastrigin function for easy visualization.

2.1. Inspiration

Many aspects in the universe are governed by dualities, which refers to two opposite forces or states of conflicting nature being at work. Few examples of mundane dualities are light and darkness, the body and the mind, male and female, good and bad or simply life and death. In the field of science, the wave-particle relationship, positive and negative charges, constants and variables or the binary digits 1 and 0 form typical examples. These dualities are depicted in the Chinese philosophy as Yin and Yang, two complementary and interdependent extremes that would not exist without the other. One aspect gradually changes to the other and this cycle is constantly being repeated, thus the balance between these two aspects results in harmony.

In the field of evolutionary computing, exploitation and exploration represent two conflicting behaviours which work together for solving a problem, and a right balance between them is integral to the performance of the algorithm. The correlation of these behaviours with Yin and Yang is apparent, and this has been the inspiration behind the algorithm.

2.2. Modelling the algorithm

In this algorithm, all the decision variables are handled in their normalized form (between 0 and 1) and are scaled appropriately for functional evaluation with the help of the variable bounds. YYPO employs two points (*P1* and *P2*) among which one of the points (point *P1*) is designed to focus on exploitation, while the other (point *P2*) is designed to focus on exploration of the variable space. These points provide the flexibility to establish a balance between the exploration and exploitation, and this is expected to lead to an ideal performance. The points *P1* and *P2* act as centres to explore the hypersphere volumes in the variable space defined by radii of δ_1 and δ_2 respectively. These radii are self-adaptive such that δ_1 has a tendency to periodically decrease and δ_2 to increase. It should be noted that δ_1 and δ_2 are not user defined parameters and simulate a converging–diverging couple of hyperspheres. The algorithm consists of two main stages, the splitting stage and the archive stage. The splitting stage is encountered at every iteration and is used to explore the hypersphere with the radii (δ) around the two points whereas the archive stage is encountered at dynamic intervals of iterations (I) and updates δ_1 and δ_2 using a user-defined expansion/contraction factor (α). In the following discussion, D is used to denote the problem dimension (number of decision variables of the problem).

The algorithm begins with the generation of two random points in the domain of $[0, 1]^D$ and evaluating their fitness. The fitter of the two points is assigned as *P1* and the other as *P2*. The required parameters in terms of the minimum and maximum number of archive updates (I_{min} and I_{max}) and the expansion/contraction factor (α) are to be specified and the values of δ_1 and δ_2 are set at 0.5. The number of archive updates is randomly generated between (I_{min} and I_{max}). Subsequent to this, the iteration loop is initiated and the fitness of the two points are compared. If *P2* is fitter than *P1*, the points as well as their corresponding δ values are interchanged, ensuring that the iteration starts with the fitter point as *P1*. Both the points are stored in the archive and each point along with its δ enters the splitting stage.

2.2.1. Splitting stage

The inputs to the splitting stage are one of the points (*P1* or *P2*) along with its corresponding search radii (δ_1 and δ_2). Although both points undergo the splitting stage, only a single point (referred as *P*) along with its search radii (δ) undergoes the splitting stage at a time. The splitting stage is designed so as to generate new points in the hypersphere (around the point *P* with the radius δ) at directions as varied as possible, while maintaining a level of randomness. This is implemented by one of the following two methods and is decided based on equal probability.

– **One-way splitting**: In this method, $2D$ identical copies of the

point P are stored as S which can be considered as a matrix of size $2D \times D$. One variable of each point in S is modified using the following equations.

$$S_j^j = S^j + r\delta \quad \text{and}$$
$$S_{D+j}^j = S^j - r\delta, \quad \text{where } j = 1,\ 2,\ 3\ \ldots\ D \tag{1}$$

In the above equation, the subscript denotes the point number whereas the superscript denotes the decision variable number that is being modified while r denotes a random number between 0 and 1. It should be noted that a new r is generated for each modification in S thereby requiring a total of $2D$ random numbers.

– ***D*-way splitting**: In this method, $2D$ identical copies of the point P are stored as S which can be considered as a matrix of size $2D \times D$. A binary matrix B is generated containing $2D$ random binary strings of length D such that each string is unique. Subsequently, every variable of each point is modified using the following equations.

$$S_k^j = S^j + r\left(\delta/\sqrt{2}\right) \quad \text{if } B_k^j = 1,\ S_k^j = S^j - r\left(\delta/\sqrt{2}\right) \quad \text{else, where } k = 1,\ 2,\ 3\ \ldots\ 2D \text{ and } j = 1,\ 2,\ 3\ \ldots\ D \tag{2}$$

In the above equation, the subscript denotes the point number (or row), the superscript denotes the decision variable number (or column) and r denotes a random number between 0 and 1. It should be noted that a new r is generated for every variable in each point in S thereby requiring a total of $2D \times D$ random numbers. The binary matrix B can be generated by randomly selecting $2D$ unique integers between 0 and $2^D - 1$ and converting into binary strings of length D.

Any violations of the scaled variable in its bound (i.e., lower than 0 or greater than 1) in any of the method is rectified by assigning it a random value in the interval of (0,1). The $2D$ new points generated by either method are evaluated for their respective fitness, and the fittest point replaces the point which undergoes the splitting stage. It is worth noting that the point obtained through the splitting stage can be potentially inferior to the point that underwent the splitting stage. A unique feature of YYPO is that the number of points being generated is dependent on the problem dimension, with more points being generated for problems of larger dimensions. This adaptive behaviour is expected to aid the algorithm in handling problems of different dimensions.

Thus at the end of the splitting stage, both the points (*P1* and *P2*) have been replaced. After the specified number of archive updates (I), the archive stage is implemented otherwise the next iteration is initiated.

2.2.2. Archive stage

The archive stage is initiated after the required number of archive updates have been achieved and it is to be noted that the archive contains $2I$ points at this stage, corresponding to the two points (*P1* and *P2*) being added at each update before the splitting stage. The best point in the archive is interchanged with point *P1* if it is fitter than the point *P1*. Subsequent to this, if the best point in the archive is fitter than the point *P2*, this point replaces *P2*. It should be noted that if the point *P1* has been interchanged with a point from the archive, the previous point which has been currently included in the archive is still considered for replacing point *P2*. The archive stage thus maintains elitism and ensures monotonic convergence (i.e. the fittest point determined at any iteration is never lost) of the algorithm. This is followed by updating the search radii (δ_1 and δ_2) using the following equations.

$$\delta_1 = \delta_1 - (\delta_1/\alpha)$$
$$\delta_2 = \delta_2 + (\delta_2/\alpha) \tag{3}$$

At the end of the archive stage, the archive matrix is set to null, and a new value for the number of archive updates (I) is randomly generated within its specified bounds (I_{min} and I_{max}). The value of δ_2 is restricted to a maximum of 0.75 as a very high value of δ_2 could lead to ineffective exploration of the search space. Although a big change in this upper limit for δ_2 (such as restricting δ_2 to 0.2) is bound to affect the performance of the algorithm, setting it at a value in the neighbourhood of 0.75 was observed to provide satisfactory performance in all the scenarios considered. For instance, restricting the value of δ_2 to a low value will hinder the algorithm's capability to explore the complete search space. On the other hand, if δ_2 is unrestrained or very high (>1), the new solutions generated from point *P2* will have a higher tendency to be outside the bounded search space. Hence, for this work we consider this upper limit of δ_2 to be fixed at 0.75, even though it could potentially be tuned for each problem.

The completion of the archive stage ends the current iteration and a subsequent iteration is initiated till the satisfaction of a termination criteria provided by the user. Thus the two stages of YYPO are expected to effectively guide the algorithm to discover the optimal solution. The pseudo code for the algorithm is provided in Fig. 1 and is divided into two sections for clarity: (i) the main algorithm and (ii) the splitting stage.

The evaluation of the fitness is encountered inside the splitting stage where $4D$ points are evaluated in every iteration as each iteration executes the splitting stage twice (once for each of the points *P1* and *P2*). This implies that for a problem of dimension D and T iterations of the algorithm, the total number of fitness evaluation, including that for the two initial points, is $4DT+2$. Hence it can be seen that for a fixed number of functional evaluations, the algorithm will execute larger number of iterations for a problem with lower dimension and vice versa. The effect of having the number of functional evaluations as a function of the dimension of the problem in terms of algorithm complexity is discussed in Section 5. An additional feature of this algorithm is that it is not necessary to identify and replace any duplicate solutions.

Thus this algorithm effectively utilizes the functional evaluations and is able to quickly determine the optimal solution.

2.3. Comments on user defined parameters

The parameters required from the user are the minimum and maximum number of archive updates I_{min} and I_{max} (both of which can assume integer values between 1 and maximum number of iteration, with $I_{min} \leq I_{max}$), and the expansion/contraction factor α (which should be given positive non-zero real values). The number of iterations after which the algorithm proceeds to the archive stage (I) is randomly generated within the bounds I_{min} and I_{max}. The parameter I also defines the maximum size of the archive, which is $2I$ (as two points are added to the archive at every iteration). In this current work, a new I is generated randomly within its bounds every time the algorithm proceeds to the archive stage, but variations to this can be explored to make it more adaptive. In its simplest form, I may be given a fixed static value, resulting in a two parameter (I and α) algorithm.

It can be inferred that, as δ_1 and δ_2 are varied inside the archive stage, I defines the frequency with which the δ_1 and δ_2 values are decreased and increased respectively. The extent to which they are changed is defined by the other user defined parameter α, as can be noted from Eq. (3). A large value of α will result in smaller changes in δ values and vice versa. Thus, it is apparent that α and I

Main algorithm:

Step 1: Set user defined parameters (I_{min}, I_{max} and α), the termination criteria and the lower and upper variable bounds of the decision variables.
Step 2: Initialize two random points $P1 = \{P1^1, P1^2, P1^3 \ldots P1^D\}$ and $P2 = \{P2^1, P2^2, P2^3 \ldots P2^D\}$, where D = dimension and $0 \le (P1^j, P2^j) \le 1, j = 1, 2, 3 \ldots D$. Initialize $\delta_1 = 0.5$, $\delta_2 = 0.5$ and $i = 0$. Generate I between I_{min} and I_{max}.
Step 3: Evaluate fitness of $P1$ and $P2$.
Step 4: **do**
Step 5: **if** $P2$ is fitter than $P1$, interchange $P1$ and $P2$, and δ_1 and δ_2, **end**
Step 6: Store $P1$ and $P2$ in an archive, $i = i + 1$.
Step 7: Execute the splitting function for $P1$ and $P2$ and receive updated $P1$ and $P2$ along with their respective fitness values.
Step 8: **if** i equals I,
Step 9: **if** the fittest point in archive is fitter than $P1$,
Step 10: Fittest point in archive and $P1$ are interchanged, **end**
Step 11: **if** the fittest point in archive is fitter than $P2$,
Step 12: Fittest point in archive is the new $P2$, **end**
Step 13: Update δ_1 and δ_2 using eq. (3).
Step 14: Clear the archive, generate new I within the archive size range, set $i = 0$, **end**
Step 15: **while** termination criteria is not satisfied.

Splitting function:

Input: Point P and its corresponding δ.
Output: Updated point P along with its fitness.

Step 1: Generate random number R ($0 \le R \le 1$)
Step 2: **if** $R < 0.5$
Step 3: Generate $2D$ copies of P and store it in S.
Step 4: Modify each point in S using eq. (1).
Step 5: **else**
Step 6: Generate a random binary matrix B of size ($2D$ x D) such that every row is unique.
Step 7: Generate $2D$ copies of P and store it in S.
Step 8: Modify each point in S using eq. (2), **end**
Step 9: Evaluate fitness for every point in S.
Step 10: If any variable in S is lesser than 0 or greater than 1, replace it with a randomly generated number between 0 and 1.
Step 11: Updated P = Fittest point in S.

Fig. 1. Pseudo code for YYPO.

are interdependent, as a high or low value of both I and α simultaneously would potentially result in poor performance of the algorithm. The first case would lead to δ_1 and δ_2 undergoing small changes infrequently, while the second case would lead to a large and frequent fluctuations.

2.4. Algorithm behaviour

In this subsection, the working of the algorithm is demonstrated on the two dimensional Rastrigin function (shown below) over the domain $[-5.12, 5.12]$.

$$\text{Minimize } F = \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$$

The results presented in this section are for 600 iterations of YYPO and its detailed stepwise demonstration for two iterations is provided in the supporting material (Supplementary, 2016). A video file is provided as accompanying material which helps in visualizing the behaviour of the algorithm on this example. A MATLAB based GUI for this algorithm is also provided (Supplementary, 2016) which can be used to test YYPO on various optimization functions and also analyse the impact of the various user defined parameters. These materials will aid in the understanding of YYPO as well as verify its performance. Fig. 2 shows the status of the algorithm at four different stages of iterations whereas Fig. 3 shows the variation of δ_1 and δ_2 over the 600 iterations. It can be observed that in view of the low dimension of this problem, the algorithm is able to employ large number of iterations for a specified limit on fitness evaluation. Thus 600 iterations requires only 4802 ($=4 \times 2 \times 600 + 2$) fitness evaluations.

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.engappai.2016.04.004.

Each subplot in Fig. 2 corresponds to an iteration showing the contours of the Rastrigin function and 12 points corresponding to (i) the global optima of the problem, (ii) the two points ($P1$ and $P2$), (iii) the four points that are generated from the point $P1$ using the splitting function of the current iteration, (iv) the four points that are generated from the point $P2$ using the splitting function of the current iteration and (v) the current best solution obtained from the algorithm. Additionally the number of functional evaluations (indicated as FEs) utilized till the reported iteration is also provided in the title of each subplot.

In the earlier stages of the algorithm, the points generated from the splitting function for $P1$ and $P2$ are spread throughout the search space. It can be observed that at iteration 200, point $P1$ undergoes D-way splitting while point $P2$ undergoes one-way splitting. This can be deduced from the location of the points

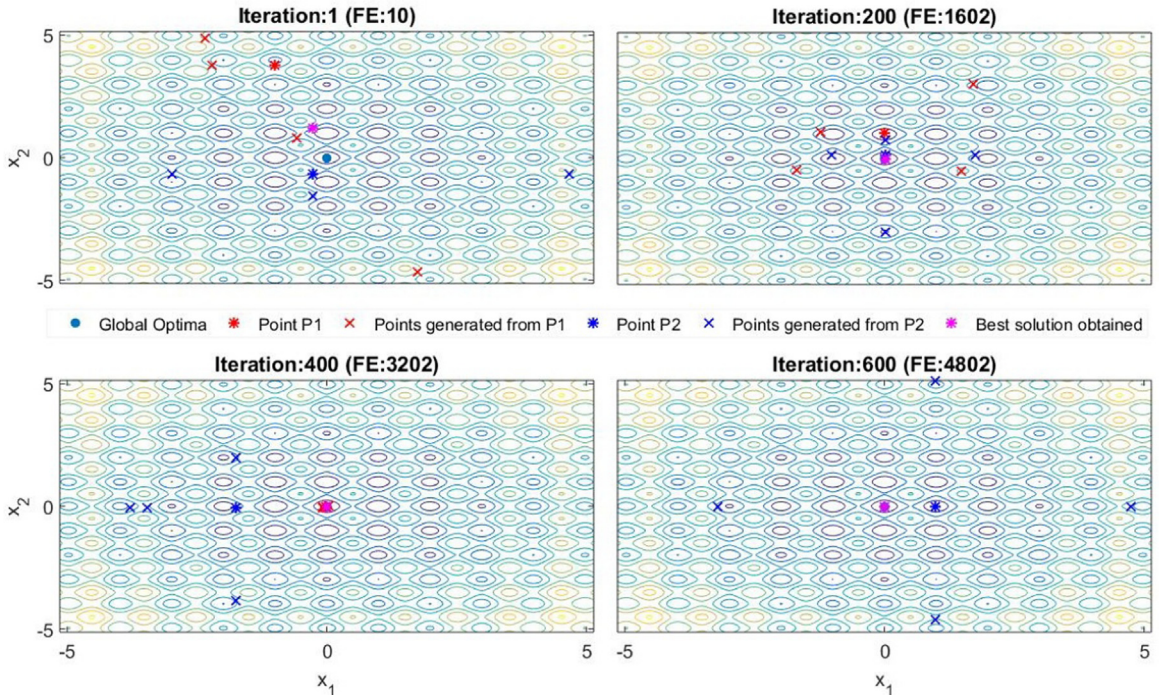

Fig. 2. Convergence of YYPO for the Rastrigin function.

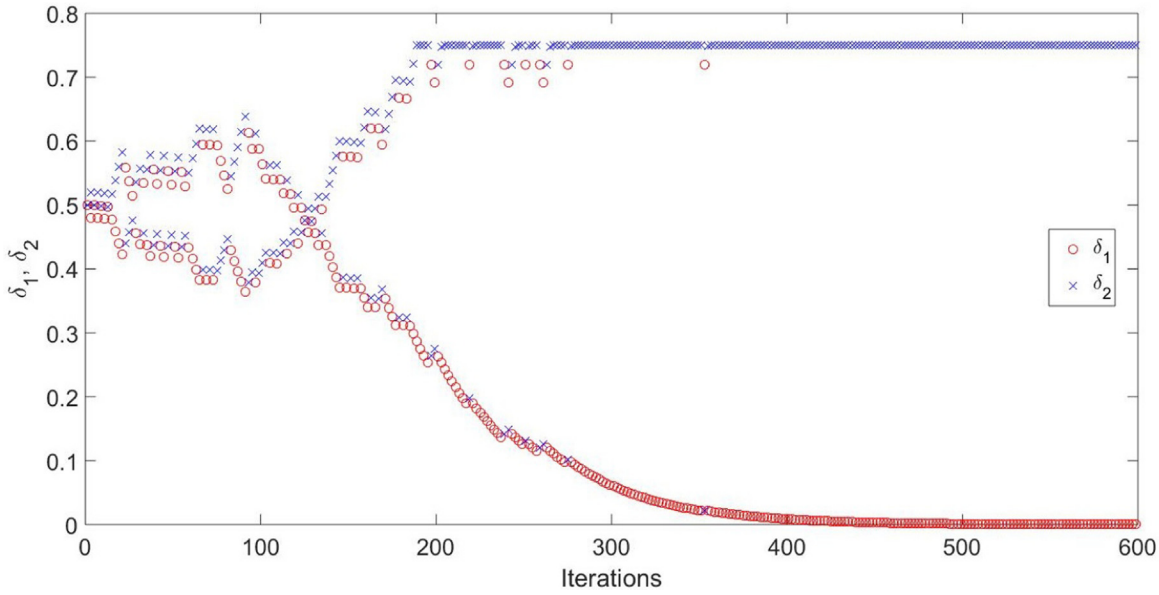

Fig. 3. Variation of δ_1 and δ_2 with iterations.

generated by *P1* and *P2*. However as the algorithm continues, point *P1* remains relatively stationary at the best point determined by the algorithm so far, while the point *P2* continues to explore the search space for a better point. In addition, the new points generated from point *P1* is very near to *P1* at iterations 400 and 600, while the ones generated from point *P2* are spread out. This is due to the variations in the δ_1 and δ_2 values. A smaller value of δ will result in all the new points generated from the corresponding *P* being within a smaller neighbourhood, hence improving its exploiting capabilities. Conversely, a larger δ helps in exploring a comparatively larger neighbourhood of the search space. If *P2* is fitter than *P1*, the points as well as the δ values are interchanged implying that the value of δ_1 may exceed δ_2. However, as one is constantly being reduced and the other increased, it is expected that by the end of the iterations δ_1 would have a value close to 0 and δ_2 would be at its maximum permitted value 0.75.

It can be seen from Fig. 3 that in the first half of the iterations, the values of δ_1 and δ_2 interchange quite often, implying that *P2* has obtained a better point than *P1* on many occasions. This suggests that exploration of the search space is predominant at this stage. Subsequently, the algorithm gradually shifts its focus to exploiting regions in the search space. In the latter half of the iterations, it is seen that δ_1 steadily converges to 0 and δ_2 to 0.75 which are their limiting values. At this stage, the algorithm stresses on searching in the region around the best obtained solution so far and attempts to improve on it (exploitation). This balance maintained by the algorithm of exploration at the initial stages and exploitation at the final stages leads to a good

performance on the Rastrigin example.

On the basis of the formulation of YYPO, the following inferences may be made:

- The search radii δ_1 and δ_2 govern how close or far apart (with respect to *P1* and *P2*) will the new points be generated in the splitting stage, and their behaviours (to increase or decrease) have the primary responsibility of maintaining the balance between exploration and exploitation.
- At the initial iterations, both the search radii have values around 0.5, hence points work on exploring the search space. This is ensured by the exchange in search radii which takes place if *P2* is fitter than *P1* at the start of an iteration. It should be noted that the exchange is the only step which can result in an apparent increase of δ_1 as well as decrease of δ_2.
- The search radii corresponding to *P1* is δ_1, and has a permanent tendency to reduce, while the search radii corresponding to *P2* is δ_2, and it has a permanent tendency to increase. Thus, the points generated by *P1* will be increasingly focused around *P1* with iterations while those generated by *P2* will be increasingly dissipated around *P2*. This corresponds to a gradual increase in the exploitative aspect of the points generated from point *P1* as well as the gradual increase in the explorative aspect of the points generated from point *P2*.
- Although the best two solutions at any iteration are discarded, they do not always participate in creating new points. For instance, consider an iteration in which the point *P1* undergoes splitting stage and is updated with an inferior point (updated *P1* is worse than the previous *P1*). The fitter *P1* was previously stored in the archive, hence is not lost, but the updated *P1* will generate new points in the next iteration. Thus, the fittest point at an iteration will not always influence the creation of new points.

2.5. Computational complexity

In this sub-section, we utilize the big O notation (worst case analysis) (Cormen, 2009) to analyse the computational complexity involved in an iteration. Every iteration of YYPO does not have the same computational complexity, hence we consider the best and worst iteration scenarios (with regard to computational complexity) separately. As the algorithm does not deal with a population of solutions, the computational complexity of an iteration is primarily a function of the problem dimension (D). The computationally best case corresponds to any iteration which satisfies the following two conditions:

- One-way splitting is employed in the splitting stage.
- The archive stage is not encountered in this iteration.

The computational complexity involved in such an iteration is at the splitting stage, and can be given by the following equation:

$$\begin{aligned} O(\text{YYPO}) &= O(O(\text{Generating } 2D \text{ points via one–way splitting}) \\ &\quad + O(\text{Scaling } 2D \text{ points for functional evaluations}) \\ &\quad + O(\text{Choosing the fittest point from } 2D \text{ points})) \\ O(YYPO) &= O(D + D^2 + D) \\ O(YYPO) &= O(D^2) \end{aligned}$$

On the other hand, the computationally worst case corresponds to an iteration which satisfies the following conditions:

- D-way splitting is employed in the splitting stage.
- The archive stage is encountered in this iteration, with $I = I_{max}$.

On considering the worst case, the computational complexity now depends on two parameters: the problem dimension and the maximum number of archive updates (user defined). The worst case complexity can be given as follows:

$$\begin{aligned} O(\text{YYPO}) &= O(O(\text{Generating } 2D \text{ points via } D\text{–way splitting}) \\ &\quad + O(\text{Scaling } 2D \text{ points for functional evaluations}) \\ &\quad + O(\text{Choosing the fittest point from } 2D \text{ points}) \\ &\quad + O(\text{Choosing the fittest point from } 2I_{max} + 1 \text{ points}) \\ &\quad + O(\text{Choosing the fittest point from } 2I_{max} \text{ points})) \end{aligned}$$

In the above equation, the last two terms corresponds to the complexity of the archive stage. The fittest point in the archive (containing $2I_{max}$ points) is compared with the current *P1*, giving the computational equivalent of choosing the fittest point from $2I_{max}+1$ points. Similarly, the updating of *P2* involves the computational equivalent of choosing the fittest point from $2I_{max}$ points. We obtain the following expression for the worst case computational complexity of an iteration:

$$\begin{aligned} O(\text{YYPO}) &= O(D^2 + D^2 + D + I_{max} + I_{max}) \\ O(\text{YYPO}) &= O(D^2) \end{aligned}$$

Thus, we observe that YYPO is a polynomial time algorithm and the complexity involved per iteration is proportional to the square of the problem dimension.

3. Other algorithms

The performance of YYPO is compared with five evolutionary algorithms (ABC, ALO, DE, GWO and PSO) and two direct search (Pattern Search (PS) and Multidirectional Search (MDS)) algorithm. In view of the numerous variants and different implementations possible in each algorithm, we have been careful to select comparative algorithms whose implementations have either been provided by their inventors or have been standardized in a software thus enabling the readers to independently verify the results presented in this article. Moreover these algorithms have not been evaluated on some of the recently proposed benchmark problems and this study provides their performance as well. In the following discussion, we restrict ourselves with a brief description of these algorithms as detailed information can be obtained from the literature.

ABC (Karaboga and Basturk, 2007) is a population based stochastic optimization algorithm which is based on the foraging behaviour of a swarm of bees. Here, the population of solutions consist of (i) employed bees, who go to their respective food sources, come back to the hive and dance in the dance area, (ii) onlookers, who watch the dances of the employed bees and subsequently select the food sources which they want to visit, and (iii) scouts, who consist of employed bees whose food source has been abandoned, and who are responsible for discovering new food sources. “Dancing” is a means of communication for the bees and is a process of information exchange.

ALO (Mirjalili, 2015) is a very recently proposed population based stochastic algorithm that has been shown to have competitive performance with other state-of-the-art optimization algorithms, which mimics the hunting mechanism of ant-lions (doodlebugs) in nature. “Ants” are points which move across the search space in a stochastic manner, affected by the traps placed by the “ant-lions”. The ant-lions are points which lay traps of sizes proportional to the corresponding ant-lion fitness. An ant becoming fitter than the ant-lion indicates the ant has been caught, causing the ant-lion to reposition itself to the last location of the ant.

DE (Storn and Price, 1997) is a well-established population

based evolutionary algorithm in which solutions are represented as vectors. In this method, a mutant vector is generated corresponding to each vector in the population (called target vector) by combining randomly selected vectors. Subsequently, these mutant and target vectors undergo crossover to generate a new vector, which replaces the target vector if it is fitter than the target vector.

GWO (Mirjalili et al., 2014) is a relatively new population based stochastic algorithm that has received considerable attention in recent literature, based on the hunting habits and hierarchy that exists in a pack of grey wolves (Canis lupus). The wolves are divided as alpha, beta, delta and omega wolves to simulate the hierarchy in a pack, with the alpha being the fittest wolf or population member. In this algorithm, the optimal solution is considered as "prey" and the exploration and exploitation of the search space is done through specially designed "Searching for prey" and "Attacking prey" stages respectively.

MDS (Torczon, 1989) is a direct search method inspired from the simplex methods proposed by Nelder and Mead among other methods. The algorithm conducts a search at multiple directions from the current vertex to find a better vertex. This search is done by one of the following three steps: the rotation step, expansion step or contraction step.

PS (Hooke and Jeeves, 1961) is a deterministic algorithm which starts with an initial guess (single point) and explores the search space around each point by creating meshes. New points generated through the mesh, if better, replaces the current point. The mesh is made to contract or expand depending on the success or failure of the current mesh.

PSO (Kennedy and Eberhart, 1995) is a population based stochastic algorithm inspired by the behaviour of a flock of birds or other such groups of organisms in which a collection of individuals, or particles, "move" in the search space and attempt to locate a target. The motion of any particle is affected by three factors: (i) the best position located till the current iteration by any particle in the swarm (Social adjustment), (ii) the best position located by the particle under consideration till the current iteration (Self adjustment) and (iii) the velocity of the particle in the previous iteration (Inertia). A comparison on few of the important aspects of these algorithms including the number of functional evaluations required in each generation/iteration is provided in Table 1.

Table 1
Brief comparison of the algorithms.

Sl. no.	Algorithm properties	ABC (Dasheng et al., 2007)	ALO (Deb et al., 2002)	DE (Agrawal et al., 2006)	GWO (Chalermchaiarbha and Ongsakul, 2012)	MDS (Deb and Agrawal, 1999)	PS (Deep et al., 2009)	PSO (Kennedy and Eberhart, 1995)	YYPO
1	Year of publication	2007	2015	1997	2014	1989	1961	1995	2015
2	Inspiration	Foraging behaviour of bees	Hunting mechanism of ant-lions	Principle of evolution	Hunting mechanism of a grey wolf pack	Simplex methods	Direct search based on patterns	Social behaviour of birds	Yin and Yang in Chinese philosophy
3	Solution representation	Bee	Ant/ Ant-Lion	Vectors	Wolf	Point	Point	Particle	Point
4	Nature	Stochastic	Stochastic	Stochastic	Stochastic	Deterministic	Deterministic	Stochastic	Stochastic
5	Monotonic convergence	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
6	Population based	Yes	Yes	Yes	Yes	No	No	Yes	No
7	No. of initial points to be generated	Half the population size	Twice the population size	Population size	Population size	One	One	Population size	Two
8	User defined parameters[a]	One[b]	One[b]	Three[c]	One[b]	Two[d]	Multiple[e]	Four[f]	Three[g]
9	Stages	Three	Five	Three	Three	Three	Two	Two	Two
10	No. of functional evaluations per generation	Population size	Population size	Population size	Population size	Not fixed	Not fixed	Population size	Four times the problem dimension

[a] Termination criteria has been excluded as a user defined parameter.
[b] Population size.
[c] Population size, weight factor and crossover probability.
[d] Initial point and the form of the initial simplex.
[e] Includes options for polling and searching methods, options for mesh size, expansion and contraction factors. Additional details available at MathWorks (2016b).
[f] Swarm size, Inertia weight, Social adjustment factor and Self-adjustment factor;
[g] Minimum and Maximum number of archive updates and expansion/contraction factor.

4. Experimental settings

In the IEEE Congress on Evolutionary Computation (CEC) 2013, a Single Objective Real Parameter Algorithm competition was organized to evaluate the performance of various optimization algorithms, both in terms of their ability to determine the optimal solution and the algorithm complexity (Liang et al., 2013). The test suite consists of 28 scalable real parameter single objective benchmark functions which represent a set of difficult optimization problems, and have been used to compare algorithm performances (Kommadath et al., 2015.). The names of the functions along with their global optima are provided in Table 2. The mathematical equations and 2D contour plots corresponding to these functions are not presented here with a view to brevity as they are provided in the technical report for the competition (Liang et al., 2013). The functions are discussed in brief below:

- Unimodal functions (CEC1–CEC5): These functions are primarily characterized by their single stationary point. The five functions in this category represent a selection of unimodal problems which take into account properties such as being non-separable, quadratic ill-conditioned, rotated, asymmetrical with one sensitive direction, having smooth local irregularities and having a

Table 2
Benchmark functions.

F	Function name	f^*	F	Function name	f^*
1	Sphere Function	−1400	15	Rotated Schwefel's Function	100
2	Rotated High Conditioned Elliptic Function	−1300	16	Rotated Katsuura Function	200
3	Rotated Bent Cigar Function	−1200	17	Lunacek bi-Rastrigin Function	300
4	Rotated Discus Function	−1100	18	Rotated Lunacek bi-Rastrigin Function	400
5	Different Powers Function	−1000	19	Rotated Expanded Griewank's plus Rosenbrock's Function	500
6	Rotated Rosenbrock's Function	−900	20	Rotated Expanded Scaffer's F6 Function	600
7	Rotated Schaffers F7 Function	−800	21	Composition Function 1	700
8	Rotated Ackley's Function	−700	22	Composition Function 2	800
9	Rotated Weierstrass Function	−600	23	Composition Function 3	900
10	Rotated Griewank's Function	−500	24	Composition Function 4	1000
11	Rastrigin's Function	−400	25	Composition Function 5	1100
12	Rotated Rastrigin's Function	−300	26	Composition Function 6	1200
13	Non-continuous Rotated Rastrigin's Function	−200	27	Composition Function 7	1300
14	Schwefel's Function	−100	28	Composition Function 8	1400

f^* represents the global optimum of the function.

smooth but narrow ridge.

- Basic multimodal functions (CEC6–CEC20): These functions have numerous local optima but a single global optimum corresponding to each function. Generally, classical derivative based optimization techniques do not perform satisfactorily on such functions as they have the tendency to get stuck at a locally optimal solution. The fifteen functions in this category portray the following characteristics: being non-separable, asymmetrical, continuous everywhere but differentiable only on a set of points, continuous everywhere but differentiable nowhere, non-continuous, rotated, having a narrow valley from local optimum to global optimum, having a very large number of local optima and having the best and the second best optimal solutions far apart.
- Composition functions (CEC21–CEC28): Each composition function, as its name suggests, is made up of a combination of multiple basic functions. All the eight composition functions are multi-modal, non-separable and asymmetrical, and have different properties around different local optima. The basic functions corresponding to each composition function is given below:

(a) Composition function 1: Rotated Rosenbrock's Function, Rotated Different Powers Function, Rotated Bent Cigar Function, Rotated Discus Function and Sphere Function.
(b) Composition function 2: Three variants of the Schwefel's Function.
(c) Composition function 3: Three variants of the Rotated Schwefel's Function.
(d) Composition function 4: Rotated Schwefel's Function, Rotated Rastrigin's Function and Rotated Weierstrass Function.
(e) Composition function 5: A variant of Composition function 4.
(f) Composition function 6: Rotated Schwefel's Function, Rotated Rastrigin's Function, Rotated High Conditioned Elliptic Function, Rotated Weierstrass Function and Rotated Griewank's Function.
(g) Composition function 7: Rotated Griewank's Function, Rotated Rastrigin's Function, Rotated Schwefel's Function, Rotated Weierstrass Function, Sphere Function.
(h) Composition function 8: Rotated Expanded Griewank's combined with Rosenbrock's Function, Rotated Schaffers F7 Function, Rotated Schwefel's Function, Rotated Expanded Scaffer's F6 Function and Sphere Function.

The variable bounds for all the functions are fixed as $[-100, 100]^D$ and each function has a corresponding global optimum value which does not change with the dimension of the problem. The C codes for the functions are freely available for downloaded (link provided in Appendix A) and can be mexed to MATLAB. The competition requires that each algorithm be tested on all the 28 functions for three dimension settings ($D = 10$, 30 and 50) and each of the above described instance is to be run for 51 random seeds, resulting in a total of 4284 ($3 \times 28 \times 51$) unique instances for an algorithm. Additionally, the test functions are to be treated as black boxes and the maximum number of functional evaluations is restricted to $D \times 10^4$ per instance.

The performance of YYPO and the other algorithms are evaluated in the format specified by the CEC 2013 algorithm competition. However, the MDS and PS algorithms requires one initial point to be specified as they are deterministic in nature. In order to provide a fair comparison, we have tested these two algorithms with 51 randomly generated initial points. All instances were run on a Windows 7 64-bit PC with an Intel i7 (3.4 GHz) processor and 6 GB RAM, and MATLAB R2015a.

The links for the codes of all the algorithms are provided in Appendix A and have been selected based on their popularity in literature and recent usage (Salimi, 2015; Li et al., 2016). An initial population size of 100 was considered for ABC, ALO, GWO and PSO, while a population size of 50 was taken for DE. The weight factor and crossover constants were taken as 0.5 each for DE. A regular simplex is taken as the form of the initial simplex for MDS. The PSO and PS algorithm inbuilt in MATLAB was used at its default settings (MathWorks, 2016a, 2016b). The user defined parameters for YYPO were not rigorously tuned, and were intuitively set as: (i) $I_{min}=5$, (ii) $I_{max}=10$ and (iii) $\alpha=25$. The same settings are employed by each algorithm for all the 4284 instances and the calculation of the algorithm complexity.

5. Results and discussion

In this section, we discuss the performance of the algorithm in four parts. Firstly, the individual performance of YYPO is studied in detail and this is followed by a comparative analysis with the other four algorithms. Subsequently, the effect of the algorithm parameters are discussed followed by a comparison of the algorithm. Finally, the performance of YYPO is showcased on three constrained engineering problems. All results are presented in terms of difference between global optimum and value obtained by the algorithm (error). The terms “best” and “worst” refer to the best and worst error obtained over 51 runs; and the terms “mean”, “median” and “standard deviation” are used to refer to the mean, median and standard deviation of the error obtained over the 51 runs by an algorithm.

Table 3
Performance of YYPO on the 10*D* test functions.

	Prob. ID (Optima)	Best	Worst	Median	Mean	Std. dev.
Unimodal	CEC1 (−1400)	1.05E−10	5.95E−09	7.39E−10	1.30E−09	1.28E−09
	CEC2 (−1300)	1.45E+03	2.63E+05	3.93E+04	6.44E+04	5.30E+04
	CEC3 (−1200)	7.64E−03	1.27E+07	3.92E+04	7.43E+05	2.30E+06
	CEC4 (−1100)	2.30E+00	7.97E+02	4.89E+01	1.01E+02	1.50E+02
	CEC5 (−1000)	1.38E−06	1.37E−04	7.32E−06	1.08E−05	1.86E−05
Basic multimodal	CEC6 (−900)	1.19E−06	9.89E+00	9.82E+00	6.26E+00	4.68E+00
	CEC7 (−800)	4.06E−02	3.91E+01	7.87E−01	4.19E+00	8.26E+00
	CEC8 (−700)	2.01E+01	2.05E+01	2.04E+01	2.03E+01	7.55E−02
	CEC9 (−600)	9.26E−01	5.87E+00	2.85E+00	3.07E+00	1.30E+00
	CEC10 (−500)	6.15E−02	6.11E−01	2.12E−01	2.36E−01	1.27E−01
	CEC11 (−400)	1.81E−10	1.39E−01	1.68E−09	2.97E−03	1.95E−02
	CEC12 (−300)	5.97E+00	3.78E+01	1.99E+01	1.98E+01	8.01E+00
	CEC13 (−200)	9.10E+00	4.20E+01	2.44E+01	2.47E+01	7.89E+00
	CEC14 (−100)	6.26E−02	7.08E+00	3.24E−01	1.38E+00	1.91E+00
	CEC15 (100)	9.34E+01	1.42E+03	6.44E+02	6.47E+02	2.58E+02
	CEC16 (200)	2.28E−01	1.34E+00	8.39E−01	8.17E−01	2.28E−01
	CEC17 (300)	1.33E−06	1.13E+01	1.04E+01	9.55E+00	3.00E+00
	CEC18 (400)	8.17E+00	4.05E+01	2.40E+01	2.50E+01	6.68E+00
	CEC19 (500)	2.96E−02	9.60E−01	4.32E−01	4.49E−01	2.11E−01
	CEC20 (600)	8.40E−01	3.50E+00	2.59E+00	2.53E+00	6.96E−01
Composition	CEC21 (700)	1.00E+02	4.00E+02	4.00E+02	3.73E+02	6.96E+01
	CEC22 (800)	5.44E−04	2.14E+02	1.02E+02	8.68E+01	7.97E+01
	CEC23 (900)	3.40E+01	1.37E+03	9.63E+02	9.14E+02	2.70E+02
	CEC24 (1000)	1.10E+02	2.19E+02	2.01E+02	1.86E+02	3.62E+01
	CEC25 (1100)	1.18E+02	2.16E+02	2.05E+02	2.00E+02	2.12E+01
	CEC26 (1200)	1.04E+02	2.00E+02	1.23E+02	1.31E+02	2.35E+01
	CEC27 (1300)	3.00E+02	5.20E+02	3.02E+02	3.36E+02	5.75E+01
	CEC28 (1400)	1.00E+02	6.50E+02	3.00E+02	2.86E+02	1.14E+02

Table 4
Performance of YYPO on the 30*D* test functions.

	Prob. ID (Optima)	Best	Worst	Median	Mean	Std. dev.
Unimodal	CEC1 (−1400)	7.67E−10	3.57E−08	3.78E−09	5.07E−09	5.38E−09
	CEC2 (−1300)	3.34E+05	4.23E+06	1.17E+06	1.39E+06	8.36E+05
	CEC3 (−1200)	1.21E+06	1.65E+09	3.46E+07	1.21E+08	2.77E+08
	CEC4 (−1100)	3.64E+02	8.07E+03	1.95E+03	2.28E+03	1.59E+03
	CEC5 (−1000)	5.86E−06	1.32E−03	1.65E−05	5.52E−05	1.90E−04
Basic multimodal	CEC6 (−900)	5.85E+00	1.37E+02	4.05E+01	4.75E+01	3.36E+01
	CEC7 (−800)	1.63E+01	1.24E+02	6.59E+01	6.29E+01	2.52E+01
	CEC8 (−700)	2.08E+01	2.10E+01	2.10E+01	2.09E+01	5.83E−02
	CEC9 (−600)	1.45E+01	3.04E+01	2.16E+01	2.19E+01	4.16E+00
	CEC10 (−500)	7.40E−03	2.31E−01	2.71E−02	3.91E−02	3.97E−02
	CEC11 (−400)	2.28E−09	2.81E+00	1.58E−07	3.63E−01	6.50E−01
	CEC12 (−300)	7.06E+01	2.96E+02	1.67E+02	1.73E+02	5.06E+01
	CEC13 (−200)	1.06E+02	3.04E+02	1.88E+02	1.93E+02	4.53E+01
	CEC14 (−100)	1.41E+00	1.53E+01	4.74E+00	5.24E+00	2.74E+00
	CEC15 (100)	2.79E+03	5.81E+03	4.03E+03	4.04E+03	7.25E+02
	CEC16 (200)	5.91E−01	2.11E+00	1.36E+00	1.38E+00	3.66E−01
	CEC17 (300)	3.07E+01	3.36E+01	3.19E+01	3.21E+01	7.54E−01
	CEC18 (400)	7.62E+01	2.07E+02	1.40E+02	1.43E+02	2.95E+01
	CEC19 (500)	6.03E−01	3.00E+00	1.57E+00	1.62E+00	4.59E−01
	CEC20 (600)	9.40E+00	1.45E+01	1.15E+01	1.16E+01	9.07E−01
Composition	CEC21 (700)	2.00E+02	4.44E+02	3.00E+02	3.10E+02	9.31E+01
	CEC22 (800)	1.07E+01	2.32E+02	1.22E+02	1.22E+02	5.47E+01
	CEC23 (900)	3.41E+03	6.14E+03	4.69E+03	4.71E+03	6.36E+02
	CEC24 (1000)	2.24E+02	3.01E+02	2.63E+02	2.62E+02	1.65E+01
	CEC25 (1100)	2.65E+02	3.14E+02	2.90E+02	2.89E+02	1.23E+01
	CEC26 (1200)	2.00E+02	2.00E+02	2.00E+02	2.00E+02	3.08E−02
	CEC27 (1300)	6.53E+02	1.07E+03	8.83E+02	8.89E+02	1.06E+02
	CEC28 (1400)	1.00E+02	1.44E+03	3.00E+02	3.18E+02	1.63E+02

Table 5
Performance of YYPO on the 50*D* test functions.

	Prob. ID (Optima)	Best	Worst	Median	Mean	Std. dev.
Unimodal	CEC1 (−1400)	2.82E−08	6.57E−07	1.12E−07	1.45E−07	1.21E−07
	CEC2 (−1300)	1.13E+06	7.26E+06	3.69E+06	3.55E+06	1.48E+06
	CEC3 (−1200)	1.21E+07	2.51E+09	3.38E+08	5.43E+08	6.10E+08
	CEC4 (−1100)	1.83E+03	1.25E+04	4.55E+03	5.40E+03	2.63E+03
	CEC5 (−1000)	6.71E−04	4.43E−03	3.74E−03	3.58E−03	6.95E−04
Basic multimodal	CEC6 (−900)	4.34E+01	1.41E+02	4.72E+01	6.36E+01	2.80E+01
	CEC7 (−800)	5.75E+01	1.44E+02	1.04E+02	1.01E+02	2.07E+01
	CEC8 (−700)	2.10E+01	2.12E+01	2.11E+01	2.11E+01	2.98E−02
	CEC9 (−600)	2.77E+01	6.14E+01	4.50E+01	4.50E+01	7.51E+00
	CEC10 (−500)	4.17E−06	9.60E−02	2.96E−02	3.47E−02	2.13E−02
	CEC11 (−400)	3.57E−07	4.97E+00	1.99E+00	1.78E+00	1.18E+00
	CEC12 (−300)	1.63E+02	6.08E+02	3.69E+02	3.76E+02	1.22E+02
	CEC13 (−200)	2.53E+02	6.45E+02	4.07E+02	4.28E+02	8.82E+01
	CEC14 (−100)	1.59E+00	1.96E+01	1.01E+01	1.02E+01	3.84E+00
	CEC15 (100)	6.12E+03	1.07E+04	8.25E+03	8.29E+03	1.02E+03
	CEC16 (200)	9.67E−01	2.98E+00	1.79E+00	1.84E+00	4.37E−01
	CEC17 (300)	5.34E+01	6.26E+01	5.75E+01	5.77E+01	2.17E+00
	CEC18 (400)	2.03E+02	4.42E+02	3.24E+02	3.22E+02	6.49E+01
	CEC19 (500)	1.57E+00	4.20E+00	3.16E+00	3.10E+00	5.78E−01
	CEC20 (600)	1.93E+01	2.40E+01	2.10E+01	2.12E+01	1.08E+00
Composition	CEC21 (700)	2.00E+02	1.12E+03	8.36E+02	7.78E+02	3.94E+02
	CEC22 (800)	2.17E+01	2.45E+02	3.21E+01	8.78E+01	8.68E+01
	CEC23 (900)	7.21E+03	1.15E+04	9.34E+03	9.47E+03	1.04E+03
	CEC24 (1000)	2.64E+02	3.83E+02	3.28E+02	3.27E+02	2.47E+01
	CEC25 (1100)	3.23E+02	4.45E+02	3.91E+02	3.87E+02	2.74E+01
	CEC26 (1200)	2.00E+02	2.08E+02	2.00E+02	2.01E+02	1.29E+00
	CEC27 (1300)	1.02E+03	1.95E+03	1.56E+03	1.55E+03	2.00E+02
	CEC28 (1400)	4.00E+02	3.71E+03	4.00E+02	8.29E+02	1.09E+03

5.1. Performance of YYPO

The performance of YYPO on the benchmark problems at dimensions 10, 30, and 50 are provided in Tables 3, 4 and 5 respectively. From Table 3, it can be observed that YYPO is able to determine values equal or very near to the global optima for eighteen of the twenty unimodal and basic multi-modal problems, the exceptions being CEC2 and CEC15. Considering the difficulty of the composition problems, values reasonably close to the optima are obtained for seven of the eight problems (all except CEC27). The standard deviation observed is low for 22 problems, the exceptions being CEC2–CEC4, CEC15, CEC23 and CEC26. Additionally for CEC1, CEC4–CEC14, CEC16–CEC20, CEC22, CEC24, CEC26 and CEC27 (21 problems), the medians obtained are very close to their corresponding best values, implying that the algorithm is consistently able to perform near to its best on these problems despite the stochastic nature of the algorithm.

In the 30 dimensional case, values equal or very near to the global optima are obtained for 13 problems (CEC1, CEC5–CEC11, CEC14, CEC16, CEC19, CEC20 and CEC22), and values reasonably close are obtained for six of the eight composition problems (all except CEC23 and CEC27). Similar to the 10*D* case, the medians for 17 problems (CEC1, CEC5–CEC11, CEC14, CEC16, CEC17, CEC19–CEC22, CEC24 and CEC26) are very close to their corresponding best values.

The ability of the algorithm to determine the global optima or a close enough value is observed to deteriorate with increase in the problem dimension, but it is still able to determine such values for 14 problems (CEC1, CEC5–CEC11, CEC14, CEC16, CEC17, CEC19, CEC20 and CEC22). Additionally, the standard deviation is reasonably low and does not vary significantly for all the above mentioned problems in addition to problems CEC13, CEC18 and CEC24–CEC26.

5.2. Comparative performance

For pairwise comparisons, the Wilcoxon signed rank test (Derrac et al., 2011) is a frequently used procedure which is utilized to check whether two samples statistically represent two different populations, i.e. there exist significant differences between them. The mean errors obtained by each algorithm (over 51 runs) for each problem are taken as the observational data for the

Table 6
Wilcoxon signed rank test results.

YYPO vs.	10*D*			30*D*			50*D*		
	R+	*R*−	*p*-Value	*R*+	*R*−	*p*-Value	*R*+	*R*−	*p*-Value
ABC	235	171	4.66E−01	284	122	6.51E−02	284	122	6.51E−02
ALO	400	6	7.26E−06	390	16	2.06E−05	278	128	8.77E−02
DE	361	45	3.21E−04	319	87	8.25E−03	311	95	1.39E−02
GWO	351	55	7.51E−04	307	99	1.79E−02	318	88	8.83E−03
MDS	385	21	3.41E−05	361	45	3.21E−04	359	47	3.82E−04
PS	393	13	1.51E−05	361	45	3.21E−04	345	61	1.22E−03
PSO	388	18	2.52E−05	315	91	1.08E−02	324	82	5.86E−03

Wilcoxon signed rank test. The mean errors are provided in Appendix B and the test results are presented in Table 6, where $R+$ is sum of ranks for the problems in which YYPO outperforms the competing algorithm.

It is observed that the null hypothesis, which states that both samples represent the same population, is rejected considering a significance value of $\alpha=0.1$ for all cases except for the 10D comparison with ABC. Additionally, the null hypothesis is consistently rejected considering a significance value of $\alpha=0.05$ for all comparisons with DE, GWO, MDS, PS and PSO over the three dimensions.

For the purpose of ranking the algorithms, we employ three nonparametric statistical ranking procedures, in particular the Friedman, Friedman Aligned and Quade tests. The Friedman test (Derrac et al., 2011) is a popularly used nonparametric multiple comparison technique in which the algorithms are ranked separately on each problem. The final rank corresponding to each algorithm is obtained by averaging these ranks over all the problems. The aligned ranking method for the Friedman test (Derrac et al., 2011) takes into consideration the identity of each observation with respect to algorithm as well as the problem. Thus, while the Friedman test allows only intra-set comparisons, the Friedman Aligned ranks are meaningful even for inter-set comparisons (Derrac et al., 2011). Both Friedman and Friedman Aligned test statistics are distributed according to a χ^2 distribution with No. of algorithm–1 degrees of freedom (8–1 in our case). The Quade test (Paulson, 2003) takes the difficulty of the problems under consideration as well by assigning importance to problems with a large range of observations. The Quade test is distributed according to an F-distribution with No. of algorithm–1 and (No. of problems–1) × (No. of algorithm–1) degrees of freedom ($8-1$ and $(28-1)\times(8-1)$in our case).

The average ranks corresponding to all the three procedures for all the problems are provided in Table 7. YYPO consistently obtains the best rank irrespective of problem dimension and statistical procedure employed while the second best rank is interchangeably obtained by PSO and ABC. YYPO obtains the best Friedman and Quade ranks on 10*D* problems, on the other hand it obtains the best Friedman Aligned rank on 50*D* problems. The low *p*-values for all cases strongly indicate that there are significant differences in the performances of the algorithms. The null hypothesis (that all the algorithms show similar performances) is consistently rejected at a significance value of 0.05, although the *p*-values are observed to have a tendency to increase with the problem dimension.

The convergence curves of all the algorithms on the 10D, 30D and 50D cases are presented in Figs. 4, 5 and 6 respectively. For the sake of brevity, the convergence curves for a select few functions of each dimension are provided in this article, while the remaining are freely available for download as supplementary material (28 plots) (Supplementary, 2016). In these plots, the ordinate is the error obtained by the algorithms while the fraction of maximum iterations is the abscissa. For a better visualization, all the plots (except for those corresponding to 10D CEC9 and 30D CEC20) are provided in the semi-log format. Due to this, convergence curves are observed to end abruptly if the algorithm obtains zero error (especially in CEC1 and CEC5).

The following are some observations that can be inferred from the 10D convergence curves:

- YYPO converges or nearly converges to a solution before termination on the problems CEC2, CEC9 and CEC10, while it is still in the process of converging on CEC4. The algorithm is observed to significantly outperform the others on CEC4.
- Although the convergence of YYPO at early iterations is observed to be slower compared to the other algorithms on the functions CEC2, CEC9 and CEC10, the algorithm outperforms the others over the course of iterations.

The following are some observations that can be inferred from the 30D convergence curves:

- YYPO has converged or nearly converged to a solution on all four functions. Although the final error obtained on CEC4 and CEC18 are of relatively high magnitude, it manages to outperform the competing algorithms.
- The algorithm shows a significantly faster and effective convergence on CEC20 in particular. On the other hand, it is observed to be sluggish at the first half of the iterations on CEC10.

The following are some observations that can be inferred from the 50D convergence curves:

- YYPO is observed to converge to a solution before termination on CEC10, CEC18 and CEC26, while it is in the process of converging on CEC14.
- YYPO and ABC are observed to have very similar convergence curves on CEC14 and CEC26. While ABC is observed to converge marginally faster, YYPO is observed to provide a marginally lower error on termination.

5.3. Parameter sensitivity of YYPO

The performances of nine different combinations of the three user-defined parameters are compared with a base case to give further insight into the effect of these parameters. We consider a base case of $I_{min}=5$, $I_{max}=10$ and $\alpha=25$, and use the Wilcoxon

Table 7
Ranks achieved by the Friedman, Friedman aligned and Quade tests.

	Friedman			Friedman aligned			Quade		
	10*D*	**30*D***	**50*D***	**10*D***	**30*D***	**50*D***	**10*D***	**30*D***	**50*D***
ABC	2.91	3.71	4.07	71.27	89.14	99.82	3.10	3.78	3.97
ALO	5.00	4.86	4.36	112.86	102.57	100.75	5.09	4.80	4.19
DE	4.29	4.91	5.21	111.71	127.48	126.71	4.26	4.67	5.01
GWO	4.61	4.54	4.43	107.79	123.71	123.21	4.38	4.62	4.76
MDS	6.98	6.50	6.00	169.09	157.46	151.18	6.97	6.55	6.02
PS	6.02	5.07	5.11	164.41	130.86	125.93	6.00	5.15	5.09
PSO	3.91	3.77	3.79	93.13	98.98	105.46	4.14	3.79	3.86
YYPO	**2.29**	**2.64**	**3.04**	**69.75**	**69.79**	**66.93**	**2.06**	**2.64**	**3.09**
Statistic	77.23	43.06	27.96	66.24	37.78	30.25	11.89	5.45	3.15
***p*-Value**	2.37E−14	1.45E−07	9.31E−05	3.92E−12	1.46E−06	3.62E−05	3.61E−12	2.62E−05	8.93E−03

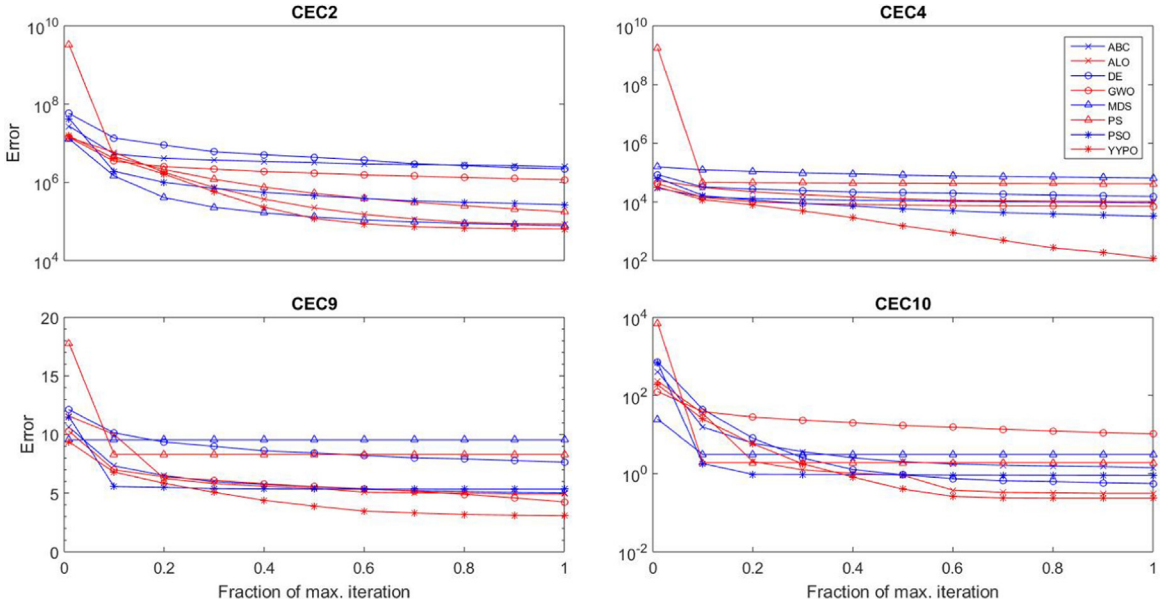

Fig. 4. Convergence curves for the 10*D* case.

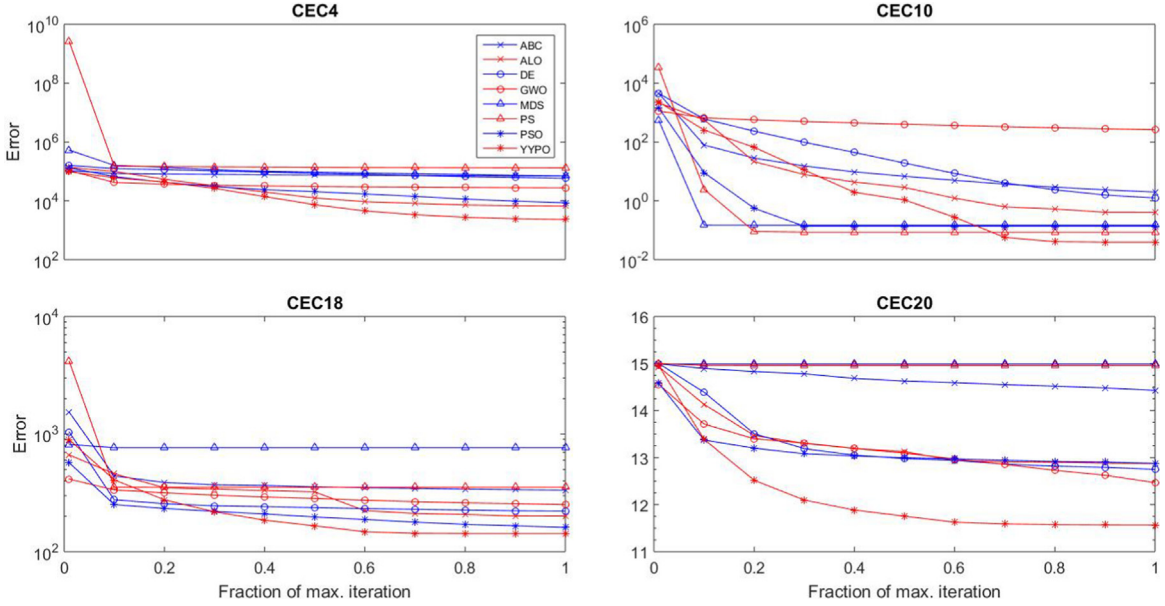

Fig. 5. Convergence curves for the 30*D* case.

signed rank test (Derrac et al., 2011) to check if the results obtained are significantly different from the base case. The inputs to the Wilcoxon test are the mean differences in obtained objective fitness with respect to global optima of each of the 28 benchmark functions over 51 runs at 30*D*. The results of this experiment are presented in Table 8, where the R+ indicate the summation of ranks where the base case outperforms the specific combination.

In line with the previous discussion, we can observe that the performance of YYPO employing (i) a relatively low value of α along with relatively high values of I_{min} and I_{max} (Combination 1–Combination 3) (ii) moderate values of α, I_{min} and I_{max} (Combination 4–Combination 6) and (iii) a relatively high value of α along with relatively low values of I_{min} and I_{max} (Combination 7) are all similar to the base case considered. This inference is evident from the large p-values obtained in all these cases. On the other hand, two instances (Combination 8 and Combination 9) are presented where we consider high values of α, I_{min} and I_{max}. This results in a significant drop in performance, as can be observed from the high $R+$ and very low p-values. The observations presented are thus consistent with our expectations on the performance of the three user-defined parameters (discussed in the Subsection 2.3). It should be noted that this by no means indicates that the base case selected represents the ideal parameter setting for this set of problems. One of the various parameter tuning procedures available in literature (Eiben and Smit, 2011; López-Ibánez et al., 2011.) may be employed for this purpose. Additionally, a more rigorous study on the effect of the user-defined parameters and the significant interactions between them can be obtained by employing techniques such as a two level or three level Design of Experiment (Montgomery, 2008).

5.4. Algorithm complexities

The CEC 2013 competition also provides a method to quantify the complexity of an algorithm (Liang et al., 2013), and this is employed in this work to study the variation of the algorithm

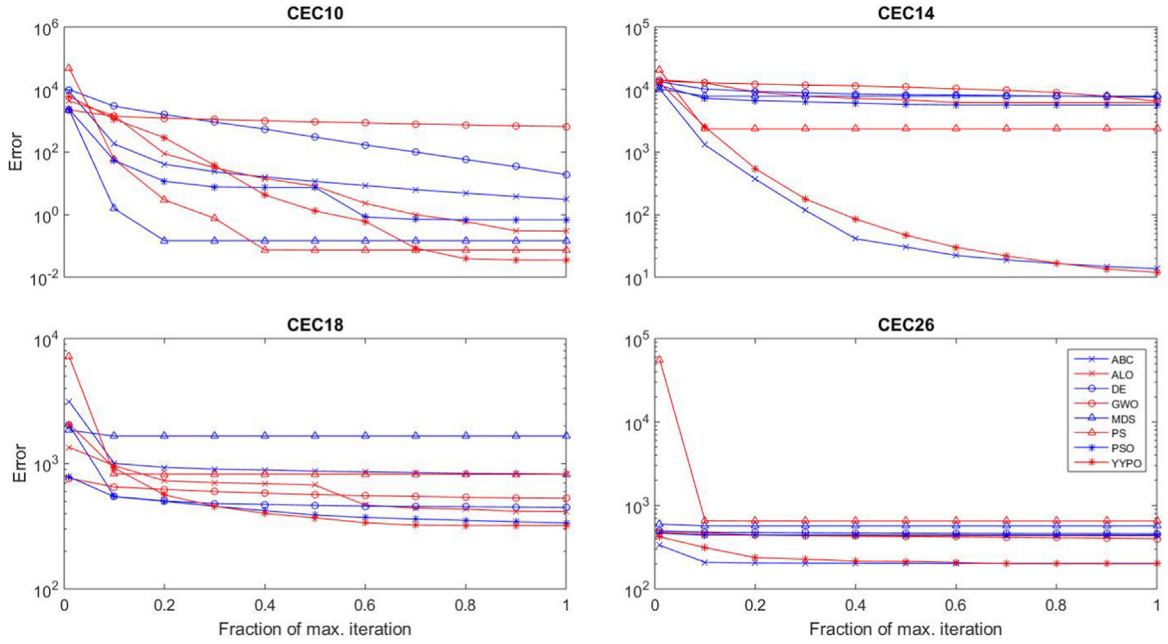

Fig. 6. Convergence curves for the 50*D* case.

Table 8
Effect of algorithm parameters.

Base case vs.	α	I_{min}	I_{max}	$R+$	*p*-Value
Combination 1	5	50	150	188	7.33E−01
Combination 2	5	75	125	187	7.16E−01
Combination 3	5	90	110	188	7.33E−01
Combination 4	10	25	75	208	9.09E−01
Combination 5	10	38	63	208	9.09E−01
Combination 6	10	45	55	183	6.49E−01
Combination 7	50	3	8	246	3.27E−01
Combination 8	50	38	63	405	4.23E−06
Combination 9	50	45	55	406	3.79E−06

Table 9
Algorithm complexities.

Algorithm	Dimension	*T0* (s)	*T1* (s)	*T2* (s)	*(T2-T1)/T0*
ABC	10	0.099	0.648	4.427	38.032
ALO				120.144	1202.819
DE				17.787	172.517
GWO				2.866	22.320
MDS				2.037	13.983
PS				14.174	136.151
PSO				3.124	24.922
YYPO				1.228	**5.837**
ABC	30		1.128	5.028	39.258
ALO				326.796	3278.102
DE				21.693	207.006
GWO				3.949	28.398
MDS				2.136	10.142
PS				13.169	121.201
PSO				3.842	27.315
YYPO				1.399	**2.728**
ABC	50		1.575	5.615	40.668
ALO				614.255	6167.096
DE				26.858	254.496
GWO				5.040	34.880
MDS				3.169	16.053
PS				13.892	123.988
PSO				4.453	28.970
YYPO				2.128	**5.566**

complexity with dimension of the problem. Although this procedure calculates the run time complexity of the algorithm, this measure is termed as algorithm complexity in the competition report (Liang et al., 2013). The complexity is calculated as *(T2–T1)/T0*, where *T0* is the time required to execute a sample code provided in the CEC report (Liang et al., 2013) for 10^6 times, *T1* is the time required for 200,000 functional evaluations of CEC14, and *T2* is the time required for an algorithm to solve CEC14 with the maximum functional evaluations being limited to 200,000 and is to be averaged over 5 runs. The algorithm complexity is shown in Table 9.

All the computations were done on the same PC, and hence the *T0* values for all the algorithm-dimension combinations and the *T1* values for each dimension setting remains the same. It is observed that the complexity for YYPO is significantly lower than that for all the others. While ALO is observed to be the slowest (highest complexity) algorithm by far, the second least complex algorithm observed (MDS) has complexities one magnitude higher than that of YYPO, thus supporting the claim of YYPO being a fast and lightweight algorithm.

Though the CEC14 function is scalable, the codes available for download allows the function to be evaluated only at discrete dimensions of $D=2$, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 variables. The algorithm complexity for each algorithm is evaluated at these dimensions and the plot between complexity and problem dimension is shown in Fig. 7. It should be noted that the ordinate is in the semi-log format to aid in better visualization.

YYPO shows a relatively high complexity at 2*D*, a possible minimum in the neighbourhood of 30*D* and a local maximum in the neighbourhood of 50*D*, after which it shows a more or less gradual increase in complexity with dimension. The reason for this high complexity at 2*D* may be deduced from the relationship between the number of functional evaluations and problem dimension. Application of YYPO on a problem of 2*D* implies 8 (4×2) functional evaluations per iteration, which results in a total of 24,999 iterations if we consider a maximum of 200,000 functional evaluations. On the other hand, a problem of 30*D* results in 120 (4×30) functional evaluations per iteration and a total of only 1666 iterations considering the same maximum functional evaluations. Thus, the number of iterations for a 2*D* problem is about

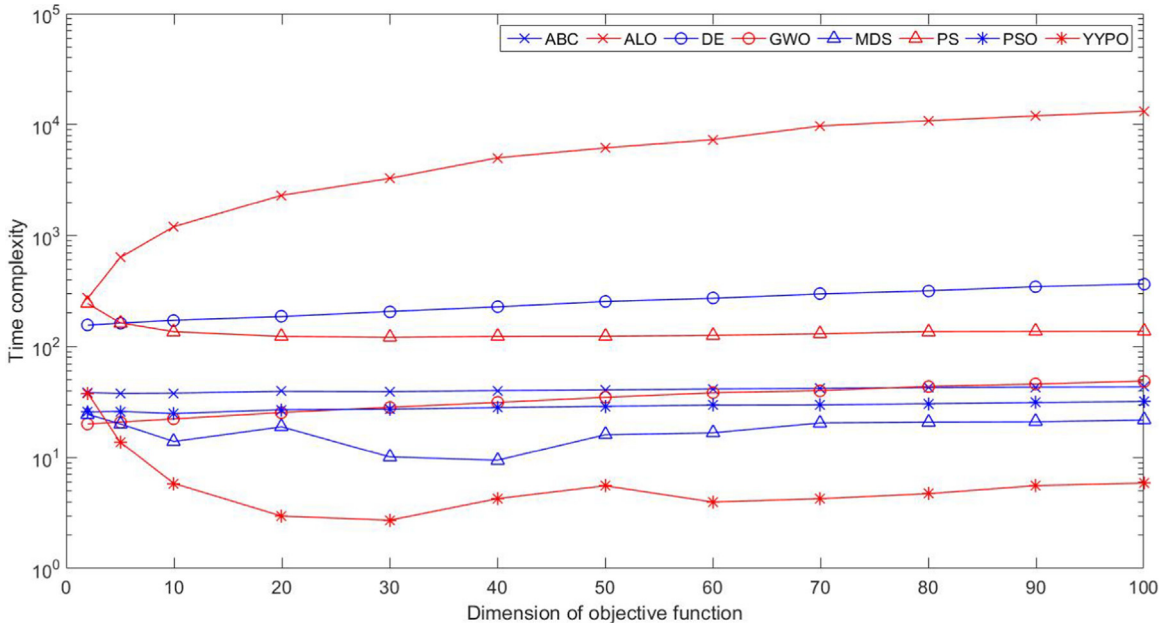

Fig. 7. Variation of algorithm complexities with dimension of the objective function.

Table 10
Experimental settings and results of YYPO on engineering problems.

Parameter		PVD	TCS	WBD
Experimental settings	α	2	2	2
	I_{min}	80	50	80
	I_{max}	100	60	90
	Max FE	50,000	50,000	50,000
Decision variables	x_1	7.8948E−01	5.1705E−02	2.0573E−01
	x_2	3.9024E−01	3.5710E−01	3.4705E+00
	x_3	4.0906E+01	1.1266E+01	9.0366E+00
	x_4	1.9200E+02	NA[a]	2.0573E−01
Constraints	g_1	0.0000E+00	0.0000E+00	−1.0454E−07
	g_2	−2.2632E−10	0.0000E+00	−7.3840E−07
	g_3	−8.3774E−05	−4.0545E+00	−2.8641E−10
	g_4	−4.8004E+01	−7.2746E−01	−3.4330E+00
	g_5	NA[b]	NA[b]	−8.0730E−02
	g_6	NA[b]	NA[b]	−2.3554E−01
	g_7	NA[b]	NA[b]	−1.2617E−03
Fitness		5.9050E+03	1.2665E−02	1.7249E+00
Average time per run (s)		1.2113E+00	1.3827E+00	1.3603E+00

[a] Problem 2 has 3 decision variables.
[b] Problem 1 and Problem 2 have 4 constraints.

15 times larger than that for a 30*D* problem, resulting in an apparently high complexity for the former case. Thus it can be seen that YYPO performs remarkably well both in its ability to determine the optimal solution and the speed with which it does so. Its performance may be attributed to the following properties:

- Most algorithms emphasize on the best population member of a specific iteration, as all the population members are directly or indirectly drawn towards it. On the contrary, the two points in YYPO are relatively unrestricted to explore the search space. In fact, the point *P2* will continue to search for a better solution irrespective of the convergence of the point *P1*.
- In almost every stochastic optimization algorithm, the best solution of a specific iteration will always be utilized to generate new points in the successive iteration, while the lesser fit points are largely neglected. While this approach may be beneficial in many cases (especially unimodal problems), it sacrifices on exploration to an extent. On the other hand, YYPO stores its best solution in an archive, but continues to generate new points based on *P1* and *P2* at that specific iteration. The contrasting factor is that although the fittest point is stored in an archive, two lesser fit points are actually utilized to generate new points. This could potentially allow the algorithm to explore the search space to a relatively larger extent.
- Population based techniques are inherently prone to be complicated and computationally intensive as compared to

techniques which are not population based. In addition, a majority of the existing algorithms perform a fixed number of functional evaluations per iteration, generally equal to the size of the population. YYPO on the other hand reduces or increases the number of functional evaluations it utilizes per iteration depending on the problem dimension. A combination of these factors allows YYPO to balance its performance with regard to solution quality and computational time required.

5.5. Performance on constrained engineering problems

In this subsection, we present the performance of YYPO on three well known constrained engineering problems:

- The pressure vessel design (PVD) problem (Li et al., 2016) consists of four variable and four constraints (g_1–g_4). The variables are (a) thickness of the vessel (x_1), (b) thickness of the head (x_2), (c) inner radius of the vessel (x_3) and (d) length of the cylindrical section (x_4). The objective of the problem is to minimize the total cost of the vessel, including the costs of material, welding and forming.
- The tension/compression spring (TCS) problem (Li et al., 2016) consists of three variables and four constraints (g_1–g_4). The variables are (a) wire diameter (x_1), (b) mean coil diameter (x_2) and (c) the length of coils (x_3). The objective is to obtain a feasible design of the spring for a minimum weight.
- The welded beam design (WBD) problem (Li et al., 2016) has four design variables: (a) width of the welded area (x_1), (b) length of the welded area (x_2), (c) depth of the main beam (x_3) and (d) thickness of the main beam (x_4). The problem has seven constraints (g_1–g_7) corresponding to various parameters such as bending stress, shear stress, maximum end deflection and loading condition. The objective is to minimize the overall cost while satisfying the constraints of the problem.

These problems are well documented in literature, and are often utilized to showcase engineering applications of metaheuristics (Salimi, 2015; Li et al., 2016). Unfortunately, minor variations in the problem definitions, the variable bounds and the termination criteria can be observed in their various implementations in literature. Additionally, the constraint handling mechanism employed is another factor which can greatly influence the algorithmic performance.

In this work, we utilize the problem definitions provided in (Li et al., 2016) to present the performance of YYPO. For the sake of simplicity, a penalty approach is employed to handle the constraints such that the fitness is penalized to a degree proportional to constraint violation (considering a penalty factor of 10^5). To account for the stochastic nature of the algorithm, we consider 20 runs of each problem, and report the best solution obtained for each problem. Table 10 presents the experimental settings as well as the results corresponding to the best solution for each problem.

The results are encouraging as YYPO is able to obtain very competitive fitness values for all the three constrained engineering problems in consideration. It can be observed the algorithm is able to satisfy all the constraints for the three problems even with a basic penalty approach, thus suggesting that it is an equally effective solver for constrained single objective optimization problems as well. Lastly, it is worth noting that the average time per run is remarkably low (lesser that 1.5 s per run for all three problems), which is yet another highlight of YYPO.

6. Conclusion

A new metaheuristic is proposed for single objective optimization based on the Yin Yang philosophy of balance. Yin and Yang depict the contradictory yet complementary extremes in Chinese philosophy and can be compared to the explorative and exploitative characteristic of an optimization algorithm. The working of YYPO is presented in detail, along with the behaviour of its self-adapting parameters and its complexity. The performance of the proposed algorithm is compared with that of well-established as well as recently proposed algorithms. In particular, these algorithms were studied on the 28 Real Parameter Single Objective Optimization problems that were proposed as part of CEC 2013. The framework provided by the CEC 2013 algorithm competition for quantifying algorithm complexity was used in this work to analyse complexities of all the five algorithms. Additionally, its performance on three constrained engineering problems were presented as well.

YYPO is observed to perform better than the other algorithms based on the nonparametric statistical tests utilized. It is seen to generally have a significantly lower algorithm complexity as compared to the rest of the algorithms. The self-adaptive nature of the algorithm results in a non-linear algorithm complexity with respect to problem dimension. This also enables the algorithm to perform consistently over different problem dimensions for a significant number of problems. On the other hand, similar to the other algorithms, YYPO is unable to determine the global optimal solution on a few benchmark problems, especially on the rotated unimodal problems in the test suite.

Future works could be directed towards reducing the number of user-defined parameters required by YYPO. At its current state, the algorithm requires the user to input three parameters to be specified by the user. This number could potentially be reduced to one user defined parameter if I and α can be correlated by an appropriate function. In addition, the algorithmic framework may be extended to accommodate multiple objectives by including appropriate sorting and diversity operators. Lastly, in view of its significantly low complexity, YYPO could be hybridized appropriately so as to further boost its performance without resulting in a slow optimizer.

Appendix A

The codes for the CEC 2013 benchmark functions can be downloaded from the following link: http://goo.gl/Ua7dSS (last accessed on April 17, 2016). The following implementations of the algorithms are considered in this work and were last accessed on April 17, 2016:

- ABC: http://mf.erciyes.edu.tr/abc/
- ALO: http://www.mathworks.com/matlabcentral/fileexchange/49920-ant-lion-optimizer–alo-
- DE: http://academic.csuohio.edu/simond/bbo/
- GWO: http://in.mathworks.com/matlabcentral/fileexchange/44974-grey-wolf-optimizer–gwo-
- MDS: http://www.mathworks.com/matlabcentral/fileexchange/2360-the-matrix-computation-toolbox/content/matrixcomp/mdsmax.m
- PS: http://in.mathworks.com/help/gads/patternsearch.html
- PSO: http://in.mathworks.com/help/gads/particle-swarm.html

Appendix B

Mean error values obtained by all the algorithms (see Tables B1–B3):

Table B1
Mean errors of 10*D* functions.

F	ABC	ALO	DE	GWO	MDS	PS	PSO	YYPO
1	0.00E+00	6.33E−10	0.00E+00	3.48E+00	2.27E−13	2.27E−13	0.00E+00	1.30E−09
2	2.47E+06	8.57E+04	2.19E+06	1.12E+06	7.82E+04	1.75E+05	2.54E+05	6.44E+04
3	9.94E+06	5.68E+06	7.89E+02	9.49E+07	1.65E+09	1.29E+09	5.46E+06	7.43E+05
4	9.24E+03	1.03E+04	1.55E+04	7.02E+03	6.52E+04	4.11E+04	2.76E+03	1.01E+02
5	1.14E−13	1.54E−03	0.00E+00	2.39E+01	1.87E−04	3.87E−10	0.00E+00	1.08E−05
6	3.02E−01	9.97E+00	9.81E+00	2.21E+01	5.09E−01	9.96E+00	9.21E+00	6.26E+00
7	3.29E+01	2.72E+01	2.57E−01	8.38E+00	2.35E+02	1.09E+02	3.01E+01	4.19E+00
8	2.04E+01	2.04E+01	2.04E+01	2.04E+01	2.06E+01	2.06E+01	2.03E+01	2.03E+01
9	5.02E+00	4.91E+00	7.66E+00	3.57E+00	9.55E+00	8.30E+00	5.34E+00	3.07E+00
10	1.43E+00	3.11E−01	5.61E−01	9.55E+00	3.09E+00	1.92E+00	8.98E−01	2.36E−01
11	0.00E+00	2.68E+01	0.00E+00	9.98E+00	5.27E+01	1.88E+01	7.16E+00	2.97E−03
12	2.23E+01	2.55E+01	2.78E+01	1.64E+01	9.81E+01	5.17E+01	1.81E+01	1.98E+01
13	3.05E+01	4.13E+01	2.75E+01	1.99E+01	1.48E+02	9.05E+01	2.65E+01	2.47E+01
14	1.24E−01	8.32E+02	1.05E+02	4.59E+02	1.47E+03	5.44E+02	3.59E+02	1.38E+00
15	6.84E+02	9.14E+02	1.51E+03	6.24E+02	1.56E+03	1.41E+03	9.36E+02	6.47E+02
16	7.79E−01	2.73E−01	1.26E+00	1.15E+00	1.35E+00	9.90E−01	6.82E−01	8.17E−01
17	8.57E+00	2.96E+01	1.10E+01	2.59E+01	7.53E+01	2.49E+01	1.86E+01	9.55E+00
18	3.97E+01	3.24E+01	3.92E+01	3.67E+01	1.16E+02	7.47E+01	2.75E+01	2.50E+01
19	4.27E−02	1.18E+00	1.20E+00	1.56E+00	3.13E+00	1.22E+00	7.73E−01	4.49E−01
20	3.24E+00	3.38E+00	3.28E+00	2.59E+00	4.72E+00	4.77E+00	3.34E+00	2.53E+00
21	1.33E+02	3.90E+02	3.92E+02	3.93E+02	3.22E+02	3.63E+02	3.75E+02	3.73E+02
22	1.23E+01	9.99E+02	3.34E+02	5.27E+02	1.64E+03	6.33E+02	3.43E+02	8.68E+01
23	1.01E+03	1.09E+03	1.49E+03	7.26E+02	1.93E+03	1.83E+03	1.08E+03	9.14E+02
24	1.33E+02	2.16E+02	2.14E+02	2.10E+02	2.17E+02	2.00E+02	2.18E+02	1.86E+02
25	1.56E+02	2.19E+02	2.13E+02	2.10E+02	2.21E+02	2.22E+02	2.18E+02	2.00E+02
26	1.30E+02	1.93E+02	1.80E+02	2.06E+02	2.58E+02	1.97E+02	1.93E+02	1.31E+02
27	3.87E+02	4.69E+02	4.70E+02	4.20E+02	5.23E+02	5.08E+02	5.46E+02	3.36E+02
28	1.30E+02	3.63E+02	3.00E+02	3.50E+02	7.60E+02	6.49E+02	3.12E+02	2.86E+02

Table B2
Mean errors of 30*D* functions.

F	ABC	ALO	DE	GWO	MDS	PS	PSO	YYPO
1	2.27E−13	9.96E−09	0.00E+00	5.78E+02	2.05E−12	6.82E−13	1.78E+01	5.07E−09
2	1.51E+07	2.31E+06	1.23E+08	1.80E+07	8.68E+05	1.10E+06	1.27E+06	1.39E+06
3	8.29E+08	2.34E+08	1.26E+06	2.57E+09	3.98E+09	1.70E+09	2.04E+08	1.21E+08
4	7.04E+04	6.65E+03	5.81E+04	2.73E+04	7.06E+04	1.33E+05	7.19E+03	2.28E+03
5	5.68E−13	6.52E−03	1.14E−13	5.43E+02	3.58E-04	8.37E−10	1.14E−13	5.52E−05
6	1.57E+01	4.80E+01	1.66E+01	1.23E+02	1.12E+01	9.96E+00	2.47E+01	4.75E+01
7	1.21E+02	1.07E+02	1.89E+01	4.13E+01	1.39E+05	1.48E+02	9.67E+01	6.29E+01
8	2.10E+01	2.10E+01	2.10E+01	2.09E+01	2.11E+01	2.11E+01	2.09E+01	2.09E+01
9	3.00E+01	2.47E+01	3.99E+01	1.89E+01	3.64E+01	3.17E+01	3.06E+01	2.19E+01
10	1.99E+00	4.06E−01	1.25E+00	2.49E+02	1.48E−01	8.57E−02	1.36E−01	3.91E−02
11	5.68E−14	1.64E+02	1.00E+02	9.16E+01	1.79E+02	5.38E+01	6.56E+01	3.63E−01
12	2.67E+02	1.67E+02	1.95E+02	1.19E+02	5.85E+02	2.91E+02	8.76E+01	1.73E+02
13	3.11E+02	2.50E+02	1.97E+02	1.65E+02	7.33E+02	4.13E+02	1.61E+02	1.93E+02
14	4.47E+00	3.56E+03	3.48E+03	2.71E+03	4.63E+03	1.56E+03	2.56E+03	5.24E+00
15	4.23E+03	4.06E+03	7.40E+03	3.35E+03	4.69E+03	4.69E+03	5.37E+03	4.04E+03
16	1.58E+00	7.36E−01	2.64E+00	2.51E+00	1.61E+00	9.83E−01	1.71E+00	1.38E+00
17	3.05E+01	1.90E+02	1.34E+02	1.48E+02	2.35E+02	8.31E+01	1.05E+02	3.21E+01
18	3.34E+02	2.01E+02	2.21E+02	2.46E+02	7.65E+02	3.53E+02	1.56E+02	1.43E+02
19	7.67E−01	8.02E+00	1.31E+01	4.36E+01	1.01E+01	3.61E+00	6.83E+00	1.62E+00
20	1.44E+01	1.29E+01	1.28E+01	1.21E+01	1.50E+01	1.50E+01	1.29E+01	1.16E+01
21	2.08E+02	3.15E+02	2.69E+02	7.50E+02	3.09E+02	3.89E+02	2.83E+02	3.10E+02
22	1.11E+02	4.75E+03	4.70E+03	2.79E+03	5.13E+03	1.54E+03	2.93E+03	1.22E+02
23	5.06E+03	4.86E+03	7.58E+03	3.68E+03	6.30E+03	6.12E+03	4.99E+03	4.71E+03
24	2.86E+02	2.78E+02	2.94E+02	2.48E+02	3.76E+02	3.01E+02	2.77E+02	2.62E+02
25	3.06E+02	3.27E+02	2.93E+02	2.68E+02	2.91E+02	3.16E+02	2.90E+02	2.89E+02
26	2.01E+02	2.87E+02	2.32E+02	3.00E+02	4.45E+02	3.70E+02	3.35E+02	2.00E+02
27	4.00E+02	1.02E+03	1.30E+03	7.90E+02	1.17E+03	1.17E+03	1.05E+03	8.89E+02
28	2.26E+02	5.24E+02	3.00E+02	9.72E+02	2.03E+03	8.72E+02	3.25E+02	3.18E+02

Table B3
Mean errors of 50*D* functions.

F	ABC	ALO	DE	GWO	MDS	PS	PSO	YYPO
1	6.82E−13	2.98E−08	2.27E−13	2.81E+03	5.91E−12	1.82E−12	1.14E−12	1.45E−07
2	2.68E+07	2.09E+06	4.61E+08	3.81E+07	1.94E+06	2.35E+06	2.07E+06	3.55E+06
3	4.52E+09	5.58E+08	2.56E+08	1.33E+10	3.81E+09	5.21E+09	7.66E+08	5.43E+08
4	1.36E+05	2.11E+01	9.08E+04	4.19E+04	8.58E+04	2.15E+05	1.01E+04	5.40E+03
5	2.73E−12	1.03E−02	1.14E−13	8.57E+02	3.95E−04	1.30E−09	5.68E−13	3.58E−03
6	4.29E+01	5.96E+01	4.35E+01	2.58E+02	4.10E+01	4.64E+01	4.62E+01	6.36E+01
7	1.77E+02	1.29E+02	6.80E+01	5.73E+01	6.01E+02	2.03E+02	1.24E+02	1.01E+02
8	2.11E+01	2.11E+01	2.12E+01	2.11E+01	2.12E+01	2.12E+01	2.11E+01	2.11E+01
9	5.93E+01	4.71E+01	7.35E+01	3.76E+01	6.77E+01	5.90E+01	5.83E+01	4.50E+01
10	3.11E+00	3.03E−01	1.92E+01	6.01E+02	1.49E−01	7.44E−02	6.91E−01	3.47E−02
11	1.36E−09	3.66E+02	2.55E+02	2.12E+02	3.46E+02	9.13E+01	1.67E+02	1.78E+00
12	7.35E+02	3.52E+02	4.06E+02	2.59E+02	1.16E+03	5.80E+02	2.19E+02	3.76E+02
13	8.01E+02	5.56E+02	4.03E+02	3.54E+02	1.61E+03	9.03E+02	3.78E+02	4.28E+02
14	1.37E+01	6.13E+03	7.58E+03	5.44E+03	7.79E+03	2.34E+03	5.62E+03	1.02E+01
15	8.65E+03	8.06E+03	1.42E+04	8.28E+03	9.01E+03	8.06E+03	9.52E+03	8.29E+03
16	2.25E+00	1.07E+00	3.49E+00	3.32E+00	2.07E+00	1.08E+00	2.38E+00	1.84E+00
17	5.10E+01	4.34E+02	3.17E+02	3.02E+02	3.81E+02	1.23E+02	2.71E+02	5.77E+01
18	8.24E+02	4.15E+02	4.48E+02	5.11E+02	1.66E+03	8.23E+02	3.33E+02	3.22E+02
19	2.33E+00	1.91E+01	2.96E+01	4.26E+02	1.58E+01	5.67E+00	2.76E+01	3.10E+00
20	2.45E+01	2.26E+01	2.27E+01	2.05E+01	2.50E+01	2.50E+01	2.20E+01	2.12E+01
21	2.54E+02	7.59E+02	5.02E+02	2.05E+03	9.16E+02	9.22E+02	8.24E+02	7.78E+02
22	4.37E+01	8.26E+03	7.66E+03	6.61E+03	8.87E+03	2.48E+03	6.19E+03	8.78E+01
23	1.04E+04	8.92E+03	1.43E+04	8.29E+03	1.16E+04	1.11E+04	9.56E+03	9.47E+03
24	3.73E+02	3.55E+02	3.79E+02	3.00E+02	4.52E+02	3.65E+02	3.46E+02	3.27E+02
25	4.15E+02	4.66E+02	3.93E+02	3.40E+02	3.66E+02	4.13E+02	3.75E+02	3.87E+02
26	2.03E+02	4.34E+02	4.58E+02	3.87E+02	5.70E+02	6.54E+02	4.42E+02	2.01E+02
27	4.07E+02	1.79E+03	2.14E+03	1.31E+03	1.95E+03	1.87E+03	1.78E+03	1.55E+03
28	4.00E+02	1.30E+03	7.16E+02	1.51E+03	2.69E+03	1.83E+03	1.68E+03	8.29E+02

Appendix C. Supplementary material

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.engappai.2016.04.004.

References

Agrawal, N., Rangaiah, G.P., Ray, A.K., Gupta, S.K., 2006. Multi-objective optimization of the operation of an industrial low-density polyethylene tubular reactor using genetic algorithm and its jumping gene adaptations. Ind. Eng. Chem. Res. 45, 3182–3199.

Babu, B.V., Angira, R., 2006. Modified differential evolution (MDE) for optimization of non-linear chemical processes. Comput. Chem. Eng. 30, 989–1002.

Chalermchaiarbha, S., Ongsakul, W., 2012. Elitist multi-objective particle swarm optimization with fuzzy multi-attribute decision making for power dispatch. Electr. Power Compon. Syst. 40, 1562–1585.

Chatterjee, A., Siarry, P., 2006. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. Comput. Oper. Res. 33, 859–871.

Cormen, T.H., 2009. Introduction to Algorithms. MIT Press, Cambridge, Massachusetts, United States.

Dasheng, L., Tan, K.C., Goh, C.K., Ho, W.K., 2007. A multiobjective memetic algorithm based on particle swarm optimization, systems, man, and cybernetics, Part B: cybernetics. IEEE Trans. 37, 42–50.

Deb, K., 2001. Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley & Sons, Hoboken, New Jersey, United States.

Deb, K., Agrawal, S., 1999. A Niched-penalty Approach for Constraint Handling in Genetic Algorithms. Artificial Neural Nets and Genetic Algorithms. Springer, Portorož, Slovenia, pp. 235–243.

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6, 182–197.

Deep, K., Singh, K.P., Kansal, M.L., Mohan, C., 2009. A real coded genetic algorithm for solving integer and mixed integer optimization problems. Appl. Math. Comput. 212, 505–518.

Derrac, J., García, S., Molina, D., Herrera, F., 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol. Comput. 1, 3–18.

Eiben, A.E., Smit, S.K., 2011. Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol. Comput. 1, 19–31.

Gülcü, Ş., Kodaz, H., 2015. A novel parallel multi-swarm algorithm based on comprehensive learning particle swarm optimization. Eng. Appl. Artif. Intell. 45, 33–45.

Gosselin, L., Tye-Gingras, M., Mathieu-Potvin, F., 2009. Review of utilization of genetic algorithms in heat transfer problems. Int. J. Heat Mass Transf. 52, 2169–2188.

Holland, J.H., 1992. Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence. MIT Press, Cambridge, Massachusetts, United States.

Hooke, R., Jeeves, T.A., 1961. "Direct Search" solution of numerical and statistical problems. J. ACM 8, 212–229.

Idris, I., Selamat, A., Thanh Nguyen, N., Omatu, S., Krejcar, O., Kuca, K., et al., 2015. A combined negative selection algorithm–particle swarm optimization for an email spam detection system. Eng. Appl. Artif. Intell. 39, 33–44.

Karaboga, D., Basturk, B., 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. Glob. Optim. 39, 459–471.

Kennedy, J., Eberhart, R., 1995. Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks – Conference Proceedings. Piscataway, NJ, 1995, pp. 1942–1948.

Khatib, W., Fleming, P.J., 1998. The stud GA: a mini revolution?. In: Eiben, A., Bäck, T., Schoenauer, M., Schwefel, H.-P. (Eds.), Parallel Problem Solving from Nature — PPSN V. Springer, Berlin, Heidelberg, pp. 683–691.

Kim, D.H., Abraham, A., Cho, J.H., 2007. A hybrid genetic algorithm and bacterial foraging approach for global optimization. Inf. Sci. 177, 3918–3937.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. Science 220, 671–680.

Kommadath, R., Punnathanam, V., Kotecha, P. Performance of Particle Swarm Optimization and Pattern Search on CEC 2013 Real Parameter Single Objective Optimization. In: Proceedings of the IEEE Workshop on Computational Intelligence: Theories, Applications and Future Directions (WCI) – Conference Proceedings, IIT Kanpur, India, 2015.

Koupaei, J.A., Hosseini, S.M.M., Ghaini, F.M.M., 2016. A new optimization algorithm based on chaotic maps and golden section search method. Eng. Appl. Artif. Intell. 50, 201–214.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M., 2011. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles Belgium.

Lee, K.H., Baek, S.W., Kim, K.W., 2008. Inverse radiation analysis using repulsive particle swarm optimization algorithm. Int. J. Heat Mass Transf. 51, 2772–2783.

Li, B., Shao, Z., 2016. Precise trajectory optimization for articulated wheeled vehicles in cluttered environments. Adv. Eng. Softw. 92, 40–47.

Li, M.D., Zhao, H., Weng, X.W., Han, T., 2016. A novel nature-inspired algorithm for optimization: virus colony search. Adv. Eng. Softw. 92, 65–88.

Liang, J.J., Qu, B.Y., Suganthan, P.N., Hernández-Díaz, A.G., 2013. Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization, pp. 1–39.

Ling, S.H., Chan, K.Y., Leung, F.H.F., Jiang, F., Nguyen, H., 2016. Quality and robustness improvement for real world industrial systems using a fuzzy particle swarm optimization. Eng. Appl. Artif. Intell. 47, 68–80.

MathWorks, 2016a. ⟨http://in.mathworks.com/help/gads/particleswarm.html#inputarg_options⟩ (last accessed 17.04.16).

MathWorks, 2016b. ⟨http://in.mathworks.com/help/gads/patternsearch.html#inpu

targ_options⟩ (last accessed 17.04.16).
Mirjalili, S., 2015. The ant lion optimizer. Adv. Eng. Softw. 83, 80–98.
Mirjalili, S., Mirjalili, S.M., Lewis, A., 2014. Grey wolf optimizer. Adv. Eng. Softw. 69; , pp. 46–61.
Mirjalili, S., Saremi, S., Mirjalili, S.M., Coelho, Ld.S., 2016. Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization. Expert Syst. Appl. 47, 106–119.
Moayedikia, A., Jensen, R., Wiil, U.K., Forsati, R., 2015. Weighted bee colony algorithm for discrete optimization problems with application to feature selection. Eng. Appl. Artif. Intell. 44, 153–167.
Montgomery, D.C., 2008. Design and Analysis of Experiments. John Wiley & Sons, Hoboken, New Jersey, United States.
Paulson, D.S., 2003. Applied Statistical Designs for the Researcher. CRC Press, Boca Raton.
Ramteke, M., Srinivasan, R., 2012. Large-scale refinery crude oil scheduling by integrating graph representation and genetic algorithm. Ind. Eng. Chem. Res. 51, 5256–5272.
Rao, R.V., 2016. Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems. Int. J. Ind. Eng. Comput. 7, 19–34.
Rao, R.V., Savsani, V.J., Vakharia, D.P., 2011. Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. Comput. Aided Des. 43, 303–315.
Rashedi, E., Nezamabadi-pour, H., Saryazdi, S., 2009. GSA: a gravitational search algorithm. Inf. Sci. 179, 2232–2248.
Ratnaweera, A., Halgamuge, S.K., Watson, H.C., 2004. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Trans. Evol. Comput. 8, 240–255.
Salimi, H., 2015. Stochastic Fractal Search: a powerful metaheuristic algorithm. Knowl. Based Syst. 75, 1–18.
Simon, D., 2008. Biogeography-based optimization. Evol. Comput. IEEE Trans. 12, 702–713.
Storn, R., Price, K., 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. 11, 341–359.
Supplementary, 2016. ⟨https://sites.google.com/site/yinyangpairoptimization/manuscript⟩ (last accessed 17.04.16).
Tam, S.C., Chio, C.H., Tam, H.K., 2011. Development of a new optimization method, yin-yang algorithm, for traveling salesman problem, System Science and Engineering (ICSSE). In: Proceedings of the 2011 International Conference on System Science and Engineering, ICSSE 2011, Macao, China. pp. 245–250.
Thomas, G., Simon, D., Michelini, J., 2015. Biogeography-based optimization of a variable camshaft timing system. Eng. Appl. Artif. Intell. 45, 376–387.
Torczon, V.J., 1989. Multi-Directional Search: A Direct Search Algorithm for Parallel Machines Ph.D. thesis. Rice University, Houston, Texas.
Tsai, H.-C., 2015. Roach infestation optimization with friendship centers. Eng. Appl. Artif. Intell. 39, 109–119.
Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1, 67–82.
Yang, X.S., Deb, S., 2009. Cuckoo search via Lévy flights. 2009 World Congress on Nature and Biologically Inspired Computing, NABIC 2009, Coimbatore, India – Proceedings, pp. 210–214.
Zhao, X., Hsu, C.-Y., Chang, P.-C., Li, L., 2016. A genetic algorithm for the multi-objective optimization of mixed-model assembly line based on the mental workload. Eng. Appl. Artif. Intell. 47, 140–146.