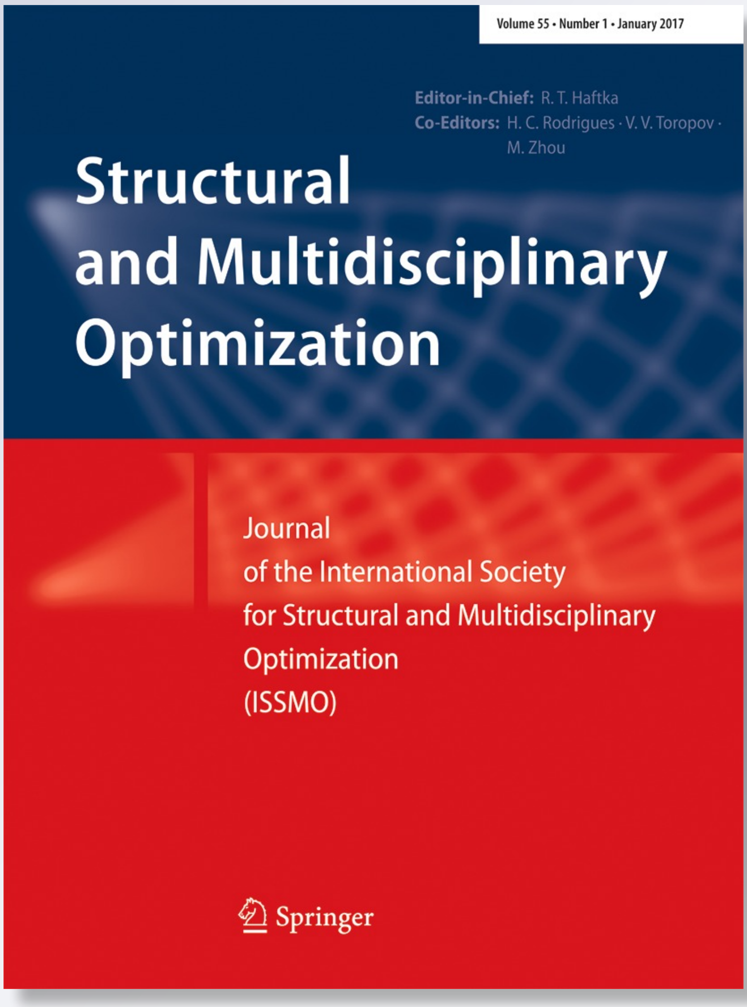
Demonstration of effective global optimization techniques via comparative analysis on a large analytical problem set

E. J. Inclan & G. S. Dulikravich

Structural and Multidisciplinary Optimization

ISSN 1615-147X
Volume 55
Number 1

Struct Multidisc Optim (2016)
55:179-204
DOI 10.1007/s00158-016-1485-3


Volume 55 • Number 1 • January 2017
Editor-in-Chief: R. T. Haftka
Co-Editors: H. C. Rodrigues · V. V. Toropov · M. Zhou
Structural and Multidisciplinary Optimization
Journal of the International Society for Structural and Multidisciplinary Optimization (ISSMO)
Springer

Struct Multidisc Optim (2017) 55:179–204
DOI 10.1007/s00158-016-1485-3

RESEARCH PAPER

Demonstration of effective global optimization techniques via comparative analysis on a large analytical problem set

E. J. Inclan[1] · G. S. Dulikravich[1]

Received: 21 February 2015 / Revised: 2 May 2016 / Accepted: 11 May 2016 / Published online: 6 June 2016

Abstract Many modern global optimization algorithms are inspired by natural phenomena rather than classes of mathematical functions. Theorems such as No Free Lunch imply that an algorithm's performance on an objective function is determined by the compatibility between the function and its structure. Grouping by inspiration blurs the distinction between algorithms, making it harder to study compatibility. Therefore, this work treats algorithms as sequential sampling algorithms, and groups them by sampling scheme: 1. perturb every design (e.g., Particle Swarm), 2. perturb a subset of designs (e.g., rand/1/bin Differential Evolution), 3. perturb a single design (e.g., best/2/bin Differential Evolution), 4. deterministically modify and then perturb the design (e.g., Quantum Particle Swarm). Using 295 analytical test cases, the structure and performance of 38 biologically inspired algorithms (major and minor variations of 5 algorithms) are compared by group. The groups are evaluated by 1. how performance scales with dimensionality, and 2. trends in mean convergence rates and accuracy. Controlling for sample size, number of algorithms/group, convergence criteria, and tuning parameters, Groups 2 and 3 demonstrate superior accuracy and convergence rates on 80 % of test cases combined, implying greater overall compatibility than other groups, and scale much better than other groups on 2nd and 4th order polynomials up to 100-dimensions, converging to minima 3–6 orders of magnitude lower. Statistical significance testing reveals overlap in the behavior of certain Group 2 and 3 algorithms on 52 test cases. Group 3 is a special case of Group 2, further implying structural compatibility with certain test cases.

Keywords Global optimization · Comparative analysis · Single-objective optimization

Nomenclature

BAT	Bat-inspired algorithm
CKO	Cuckoo search algorithm
DE	Differential evolution family of algorithms
BST	best/2/bin DE variant
DN3	Donor3 DE variant
STD	rand/1/bin DE variant
TDE	Trigonometric DE variant
FFA	Firefly algorithm
MQP	Modified quantum-behaved particle swarm algorithm
QPS	Quantum-behaved particle swarm algorithm
PSO	Particle swarm optimization algorithm
PRD	Particle swarm with random differences

1 Introduction

Biologically inspired algorithms are usually analyzed as independent entities. However, it has been observed that steps from one algorithm can often appear in another, making several of these algorithms similar, sometimes to an extreme (Weyland April-June 2010). In the case of algorithms introduced as a collection, such as Differential Evolution (DE), the similarity is intentional

✉ G. S. Dulikravich
dulikrav@fiu.edu; http://maidroc.fiu.edu

1 Multidisciplinary Analysis, Inverse Design, Robust Optimization and Control (MAIDROC) Lab., Department of Mechanical & Materials Eng, Florida International University, 10555 West Flagler St., Miami, FL 33174, USA

(Storn and Price 1997). In other cases, it is the result of merging two algorithms into one, as in PSO-DV from (Das et al. 2008). In still other algorithms, it appears unintentional. An example could be a comparison of the "difference vector" in PSO-DV to the "empty nest" operation in (Yang and Deb 2010) discussed below.

Since most of these methods are zeroth order (no gradient information is used), the authors must rely on roughly the same set of mathematical operations in order to develop their algorithm, and similarities naturally emerge. For example, they may use relative objective function values, simple linear combinations of vectors, or statistical measurements, which partially explains the prevalence of ranking techniques, weighted averages, or random perturbations (such as those presented below).

In light of this trend, this paper will examine 38 algorithms (variations of five principal algorithms) grouped by their common traits, and connect these traits to trends in their performance. In order to identify broad trends, the algorithms will be executed on 295 analytical test cases from the Schittkowski & Hock standard test case collection (SHC) (Schittkowski and Hock 1981; Schittkowski 1987). The SHC contains analytical problems ranging from 2-dimensional, unconstrained problems to 100-dimensional unconstrained problems, as well as heavily constrained problems of varying dimensionality, and continuous or discontinuous objective functions with and without symmetries. Although no set of test cases is exhaustive, this set is too diverse to perfectly tune any known algorithm on, improving the resulting analysis' utility (Ahrari et al. 2010). Readers unfamiliar with the SHC should note that the test cases are not all numbered sequentially (e.g., there are no test cases numbered 120-199), therefore, the final test case is #395.

Rather than simply compare algorithm convergence speeds, this paper seeks to address some questions of *why* a particular algorithm works well on the SHC (see (Hooker 1995)). Completely answering the question is a daunting task beyond the scope of this work (see (Culberson 1998)). An ideal metric for this would be some easily quantifiable measure of compatibility. Although Wolpert, Macready and other authors have demonstrated that an algorithm's relative superior performance on a problem is caused by its compatibility with that problem (Wolpert and MacReady 1997; Droste et al. 2002; Radcliffe and Surry 2005), such measures do not appear to be widely circulated, and may still be very few in number. In an effort to quantify compatibility based on algorithm characteristics, this paper will associate performance with the common traits of algorithms using special analyses including: (a) how performance scales with the number of design variables, (b) how types of algorithms (based on their common features) fare on the SHC set of test problems as a whole, and (c) how optimization algorithms of a certain type compare to one-another on the SHC.

2 Optimization algorithms

Classical optimization algorithms are often developed by making assumptions about the nature of the objective function. Many modern global optimization algorithms, however, are based on observations from biology or physics (these will be referred to as MGOAs) and make few or no assumptions about the problem (Culberson 1998). Since the MGOAs examined here stochastically update their designs within some vector space, the algorithms will be treated as sequential sampling algorithms with no regard given for their sources of inspiration. As such, these algorithms have three key features: (a) a sample size, (b) an equation for generating a new sample, and (c) convergence criteria. This research focuses on the equation for generating new samples, and Section 2.1 discusses how algorithms will be grouped based on similarities in these equations. The remainder of this subsection will briefly discuss the remaining key features of these algorithms.

As with any sampling algorithm, the sample size is crucial to the accuracy of the statistic being estimated. MGOA authors have suggested everything from increasing the sample size exponentially, linearly, or even logarithmically with problem dimension, to suggesting values that do not require scaling the sample size with problem dimension (Das et al. 2008; Gendreau and Potvin 2010; Yang 2010a). This ambiguity does not exist in methods that guarantee the identification of the global minimum (referred to as "exact optimization algorithms," EOAs, in the literature (Rothlauf 2011; Rardin and Uzsoy 2001)). An exhaustive search algorithm can solve a combinatorial black-box minimization problem with a sample size equal to a full factorial of the problem dimension (the sample size is infinite if the problem is continuous). In cases when the objective function is known, EOAs can be proven to require polynomial time/memory, exponential time, etc. The ambiguity in MGOAs suggests that the sample size required by an algorithm on a known problem can be correlated with its compatibility to that problem, provided that the MGOA has a non-zero probability of identifying the global minimum. The question then becomes, how can this correlation be used to measure an increase or decrease in compatibility between algorithms? Following that thought to its extreme: if an algorithm is systematically altered such that its compatibility successively increases, is it not possible, in principal, to reverse engineer an exact optimization algorithm? In this paper the sample size for all algorithms is determined by (1), a nonlinear function developed for (Inclan 2014),

$$T = ceil\left(\frac{ceil\left(12.59\sqrt[3]{\dim^2}\right) + ceil\left(14.142\sqrt{\dim}\right) + ceil(65.5\log_{10}\dim)}{3}\right) \tag{1}$$

where T is the sample size, dim is the dimension of the design space, and *ceil* is a function that rounds a number up to the nearest integer. Currently, no theoretical justification exists in the literature for why an MGOA ought to have a particular sample size for a particular problem. All recommendations are empirical and case-specific, including (1), which was developed using the SHC. Although sample size is often treated as a tuning parameter, it is a control variable in this work. Standardizing tuning parameters in this way enables statements about performance to be more easily attributed to an algorithm's structure. Readers will find that the performance of the MGOAs discussed here will vary with sample size on individual problems within the SHC as a direct consequence of the No Free Lunch Theorem (Wolpert and MacReady 1997).

A variety of convergence criteria exist in the literature, which, like optimization algorithms (Droste et al. 2002), can be "deceived" when applied to an incompatible problem, resulting in premature or poor convergence. For example, convergence based on the improvement in the value of the objective function can fail when the objective function is nearly flat. Convergence based on satisfying the Karush-Kuhn-Tucker conditions will fail when the gradients of the constraints are linearly dependent at the point of interest. Some authors have incorporated multiple convergence criteria along with a rule system that decides which criterion to activate in an effort to avoid this issue (e.g., Dulikravich et al. 1999), but even a hybrid convergence criteria is "a" convergence criteria that can be deceived by some unforeseen circumstance. Poorly selected convergence criteria can make an algorithm that is compatible with a problem appear incompatible, thereby biasing the analysis of the algorithm. Although it is impossible to decouple convergence criteria from an algorithm, the algorithms in this paper are simply stopped after 200 iterations so that any bias can be attributed exclusively to premature convergence.

For constrained optimization, some additional mechanism is needed to guide the algorithm's sample into the feasible design space and prevent it from escaping. Constraint enforcement techniques typically either modify the algorithm or the objective function (Coello Coello 2002). The method chosen here is a penalty function developed for (Inclan 2014) given by (2):

$$P\left(\vec{x}\right) = \sum_{i=1}^{I}\left|h_i\left(\vec{x}\right)\right| + \sum_{j=1}^{J}\max\left[0, g_j\left(\vec{x}\right)\right] + kF\left(\vec{x}\right), \quad k \le I + J \tag{2}$$

where k is the number of violated constraints, and $I + J$ is the total number of constraints. Note that the objective function, f, is translated according to (3):

$$F\left(\vec{x}\right) = f\left(\vec{x}\right) - f_{\min} \tag{3}$$

where $f_{\min}$ is the published global minimum value. The complete objective function is,

$$U\left(\vec{x}\right) = F\left(\vec{x}\right) + P\left(\vec{x}\right) \tag{4}$$

Thus, the constrained objective function's global minimum is always zero, which facilitates debugging.

Side constraints, which bound a design variable, are enforced by projecting the violating point to the edge of the boundary it violates (see "Nearest" method in (Helwig and Branke 2013)). Some of the SHC are unbounded in one or both directions for some or all of the dimensions of the design space. Although the algorithms presented here do not require explicit bounds, the random number generator used to create the initial set of designs requires them (Sobol's algorithm, see (Burkardt 2009; Bratley and Fox March 1988; Sobol 1976)), therefore unbounded problems were truncated to some "large interval." With few exceptions, unbounded domains were set to [−300,300] with no special consideration given to how it affects problem symmetry (if any).

2.1 Algorithm groups

Wherever possible, this paper will preserve the notation utilized by the authors in their original papers. However, in order to avoid confusion, all of the algorithms described in the proceeding subsections will utilize the notation that follows. Let X^g be the set of all vectors in the algorithm's sample during a given iteration, and the vectors, $\vec{x}^g$, are elements of X^g. The superscript, g, is the symbol for iteration number because MGOA literature often uses the term "generation" to mean iteration. Let Y^g be a proper subset of X^g of size $S < T$.

$$\left\{\vec{x}^g \mid \vec{x}^g \in X^g\right\} \tag{5}$$

$$Y^g \subset X^g \tag{6}$$

Let the subscripts i and j denote an element of X^g and Y^g, respectively. Let the subscript c denote a vector in X^g that contains some special characteristic, such as being the vector with the lowest objective function value in X^g (it need not be unique).

$$\left\{\vec{x}_i^g \mid \vec{x}_i^g \in X^g \quad \forall i \in T\right\} \tag{7}$$

$$\left\{\vec{x}_j^g \mid \vec{x}_j^g \in Y^g \quad \forall j \in S\right\} \tag{8}$$

$$\left\{\vec{x}_c^g \mid \vec{x}_c^g \in X^g, \quad c \in T\right\} \tag{9}$$

The algorithm categories are all based on the general formula in (10):

$$\vec{x}_i^{g+1} = \vec{d}^g + \vec{p}^g \tag{10}$$

This means that these algorithms generate a new sample for the next iteration ($g+1$), by selecting a vector $\vec{d}$, and perturbing it using a vector $\vec{p}$. The vector, $\vec{d}$, is computed deterministically, while the perturbation vector, $\vec{p}$, is computed stochastically. Another interpretation of (10) is that these algorithms iteratively sample a region bounded by $\vec{p}$. If $\vec{p}$ has a mean value of zero, then this region is also centered at $\vec{d}$. Let us define one more vector, $\vec{w}^g$, as a vector that is not an element of X^g,

$$\left\{\vec{w}^g \mid \vec{w}^g \notin X^g\right\} \tag{11}$$

Although $\vec{w}^g$ is not an element of X^g, it may be deterministically constructed from elements of X^g. Similarly, $\vec{p}$ may be constructed from elements of X^g. Finally, the algorithm groups are,

Group 1 $$\vec{x}_i^{g+1} = \vec{x}_i^g + \vec{p}^g \tag{12}$$

Group 2 $$\vec{x}_i^{g+1} = \vec{x}_j^g + \vec{p}^g \tag{13}$$

Group 3 $$\vec{x}_i^{g+1} = \vec{x}_c^g + \vec{p}^g \tag{14}$$

Group 4 $$\vec{x}_i^{g+1} = \vec{w}^g + \vec{p}^g \tag{15}$$

Group 1 algorithms produce a new sample by perturbing every vector in the current sample. In this sense, Group 1 algorithms operate like a Random Walk algorithm (Yang 2010a) with an initial sample size greater than one. Group 2 algorithms produce a new sample, by perturbing a subset of the vectors in the current sample. The subset may be selected randomly or deterministically. Group 3 algorithms are a special case of a Group 2 algorithm. Here, a specific element is selected to be perturbed, usually because of some desired attribute. The vector possessing this attribute may change from one iteration to the next. Group 3 algorithms include older methods such as Grid Search and Random Search (Vanderplaats 2005). Group 4 methods generate a vector using some formula, and perturb that vector. For the purposes of this paper, methods containing equations from multiple groups will be classified as hybrids, and their resemblance to other methods will be discussed.

Some of the algorithms discussed here will contain two other features that significantly affect their performance. The first is a recursive perturbation vector that takes the form,

$$\vec{p}^g = f\left(\vec{p}^{g-1}\right) \tag{16}$$

Recursion introduces a form of "memory" to the algorithm, which moves it away from the blind-search approach common to many MGOAs (Culberson 1998). The second feature is elitism, which, in the simplest case, takes the form,

$$Y^{g+1} \subset X^g \tag{17}$$

That is, some members of the new sample are simply copied over from the previous sample (as often seen in Genetic Algorithms (Gendreau and Potvin 2010)). The algorithms here, however, contain operations that function like elitism without explicitly copying over old vectors. They include: (a) comparative updates, in which the vector in the new sample only replaces the vector in the old sample if it is superior to the old vector, and (b) reference points, in which the perturbation equation includes vectors specifically selected to "guide" the search (usually due to having lower objective function values than the rest of the sample). Comparatives updates is a strong form of elitism, whereas reference-point elitism is a weak form (it is possible for the reference point to disappear from the sample). The precise impact of recursion and elitism within groups and across groups will be the subject of future investigations.

2.2 Particle swarm optimization (PSO)

PSO (Kennedy and Eberhart 1995) is a very popular algorithm that has been the topic of several individual and comparative studies (for example (Das et al. 2008; Vesterstrom and Thomsen 2004; Angeline 1998; van den Bergh and Engelbrecht 2006)), and has inspired numerous variations including some described below. It is a Group 1 algorithm with recursion and reference-point elitism that uses (18–19):

$$\vec{x}_i^{g+1} = \vec{x}_i^g + \vec{p}_i^g \tag{18}$$

$$\vec{p}^g = \alpha \vec{p}_i^{g-1} + \beta R_1 \left(\vec{x}_{best,i}^g - \vec{x}_i^g \right) + \gamma R_2 \left(\vec{x}_{best,G}^g - \vec{x}_i^g \right) \tag{19}$$

where the perturbation vector is known as the "velocity vector," α, β and γ are user defined scalars, and R_1 and R_2 are uniformly distributed random numbers $\in [0,1]$. Some authors cast the equation in such a way that the coefficients α, β and γ are related to another scalar called the "constriction factor" (Das et al. 2008; van den Bergh and Engelbrecht 2006), which can be tuned to control the overall behavior of the perturbation term. Here, the scalars are tuned independently. The recursive term is called the "inertia," that reflects a swarm's resistance to changing direction. In problems where the global minimum is located at the boundary of the search domain, the inertia term can cause PSO to repeatedly overshoot the domain boundary. Since side constraints are enforced by projecting the violating vector component back to the boundary, PSO's inertia term enables it to identify the global minimum faster (Helwig and Branke 2013). The vector $\vec{x}_{best,i}$ corresponds to the best value ever held by the i^{th} design vector (referred to as the "individual best"). If $\gamma = 0$, the algorithm can be referred to as "individuality-only PSO," because each vector's search direction becomes decoupled from the other vectors (the "individual's" behavior is independent of the others). The vector $\vec{x}_{best,G}$ is the best solution ever found by the algorithm (referred to as the "global best"). If $\beta = 0$, the algorithm can be referred to as "sociability-only PSO," indicating that the vectors all move toward the same point (i.e., "social" behavior). In this work, the global best is updated at the end of a full iteration, rather than immediately after a superior design is identified. The algorithm can be described as follows:

```
1. Initialize sample
2. Initialize set of individual best vectors (copy of initial sample)
3. Store global best vector
4. Initialize perturbation vector
5. While convergence criteria not met…
  a. For every design in the sample …
    i. Apply (19) to every design in sample (generates new sample)
    ii. If new point is superior to old individual best, replace old
        individual best with new point
  b. If any point in new sample is superior to global best, replace global
     best with new point
6. Check convergence criteria
```

Thus, we see that due to PSO's recursive nature it must store the sample in memory, in addition to a set of individual best vectors, the global best vector, and a set of perturbation vectors.

2.3 Quantum-behaved particle swarm (QPS)

Drawing some inspiration from PSO, QPS borrows the concept of a set of individual best vectors and also utilizes the global best vector (Inclan et al. 2013; Sun et al. 2004), but it is a fundamentally different algorithm. QPS does not use recursion (no inertia term) but does use a form of reference-point elitism. QPS is a Group 4 algorithm that uses (20–23),

$$\vec{x}_i^{g+1} = \vec{0} + \vec{p}_i^g = \vec{p}_i^g \tag{20}$$

Original:

$$\vec{p}_i^g = \phi\, \vec{x}_{best,i}^g + (1-\phi)\, \vec{x}_{best,G}^g + \alpha_U \left| \vec{C}^g - \vec{x}_i^g \right| \tag{21}$$

$$\phi = \frac{\beta R_1}{\beta R_1 + \gamma R_2} \tag{22}$$

$$\alpha_U = \pm \alpha \ln\left({}^{1}/_{R_3} \right) \tag{23}$$

Relaxed:

$$\vec{p}_i^g = \phi\, \vec{x}_{best,i}^g + (1-\phi)\, \vec{x}_{best,G}^g + \alpha_U \left(\vec{C}^g - \vec{x}_i^g \right) \tag{24}$$

where $\vec{C}^g$ is the "mean best value" which is simply the arithmetic mean of the individual best vectors for each iteration. The first two terms of (21) (the terms scaled by ϕ) are collectively called the "local attractor." Note that (20) is centered at zero, which means that this method samples probabilistically throughout the domain, with no deterministic focus on one region. The scalars R_1, R_2 and R_3 are uniformly distributed random numbers $\in [0,1]$, and α, β and γ are user-defined scalars. The original form of the perturbation equation in (Sun et al. 2004) is (21), which computes the difference between $\vec{C}^g$ and $\vec{x}_i^g$ within an absolute value rather than parentheses. The $\pm$ operation in (23) is resolved simply by assigning a 50 % probability that the term will be either positive or negative. Since the absolute value function and the $\pm$ operation serve similar purposes, then (24), which is the relaxed form of (21), will be examined in order to draw attention to the characteristics of α_U. Since the scalars ϕ and α_U are functions of randomly generated numbers, their probability distributions are no longer uniform.

Thus $\phi \in [0,1]$, while $\alpha_U \in (-\infty,\infty)$. Note that the x-axis of Fig. 1b is logarithmic. The majority of α_U observations range from −100 to 100, which can cause large variations in the perturbation vector. Therefore, this sampling scheme (i.e., the location of the perturbation vector for each new sample) is somewhere along the line segment connecting $\vec{x}_{best,i}$ to $\vec{x}_{best,G}$, and then displaced by some modest, but potentially large value along the line connecting the current sample point to the mean individual best. If all of the individual best points are clustered close together, (23) can still cause sampling far from this cluster, which may or may not be desirable for a given problem.

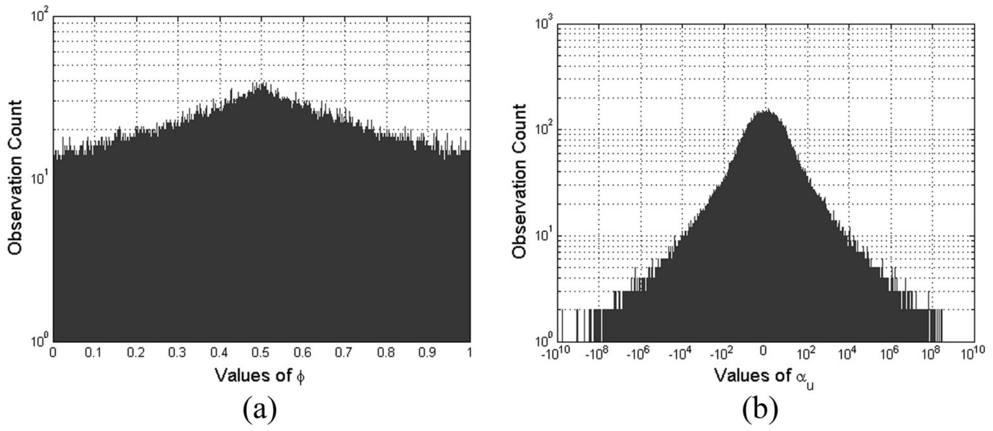


Fig. 1 Histogram of **a** ϕ, for $\beta=\gamma=2$ and **b** α_U, for $\alpha=1$, based on 10 million random samples. These plots highlight the way in which standard distributions can be combined to produce new behavior. Critically, α_U can take on enormous values, which can dramatically alter the performance of QPS in unbounded domains

The algorithm can be described as follows:

```
1. Initialize sample
2. Initialize set of individual best vectors (copy of initial sample)
3. Store global best vector
4. Compute/store mean of individual best vectors
5. While convergence criteria not met…
  a. For every design in the sample …
    i. Use (21) to generate new design
    ii. If new point is superior to old individual best, replace old
        individual best with new point
  b. If any point in new sample is superior to global best, replace global
     best with new point
6. Check convergence criteria
```

Due to the lack of an inertia term, QPS requires less memory than PSO.

2.4 Modified quantum-behaved particle swarm (MQP)

The only difference between the MQP algorithm presented here (Sun et al. 2007), and QPS, is the local attractor. Recall (24) from QPS,

$$\vec{p}_i^{\,g} = \phi\,\vec{x}^{\,g}_{best,i} + (1-\phi)\,\vec{x}^{\,g}_{best,G} + \alpha_U\left(\vec{C}^g - \vec{x}_i^{\,g}\right)$$

In MQP, the second term is replaced by (25–26),

$$\vec{p}_i^{\,g} = \phi\,\vec{x}_{best,i} + (1-\phi)\,\vec{G} + \alpha_U\left(\vec{C}^g - \vec{x}_i^{\,g}\right) \tag{25}$$

$$\begin{aligned} &\mathit{if}\ \ U\left(\vec{x}^{\,g}_{best,j}\right) < U\left(\vec{x}^{\,g}_{best,i}\right)\\ &\quad \vec{G}^g = \vec{x}^{\,g}_{best,j}\\ &\mathit{else}\\ &\quad \vec{G}^g = \vec{x}_{best,G} \end{aligned} \tag{26}$$

where $\vec{x}^{\,g}_{best,j}$ is the individual best of a randomly selected vector such that $j \neq i$ (Sun et al. 2007). The rest is identical to QPS. Since the change is applied only to the perturbation vector, MQP is also a Group 4 algorithm.

2.5 Firefly algorithm (FFA)

Although it has been demonstrated that FFA can reduce to a special form of PSO (Yang 2009), FFA distinguishes itself from all other algorithms here due to its relative complexity. Each design in the sample is compared to every other design in the sample. This structure causes a naïve implementation of the algorithm to be $O(T^2)$ with respect to the number of comparisons it performs, while remaining $O(T)$ in terms of the number of objective function evaluations per iteration. All other algorithms here are $O(hT)$, where h is a constant. When applied to problems with high dimension (dim >50) and rapidly computed objective functions, the difference in performance is noticeable even on modern computers. Additionally, FFA is classified as Group 4 without recursion, and with reference-point elitism based on its current implementation, but can be easily converted into a Group 1 algorithm. The equations are:

$$\vec{x}_i^{\,g+1} = \vec{w}_i^{\,g} + \vec{p}_i^{\,g} \tag{27}$$

$$\vec{w}_i^{\,g} = a_0\,\vec{x}_i^{\,g} + \sum_{j=1}^{m} a_j\,\vec{x}_j^{\,g} \tag{28}$$

$$\vec{p}_i^{\,g} = b\,\vec{u} \tag{29}$$

$$\vec{u} = \sum_{k=1}^{\dim} \alpha R_1 L_k \hat{e}_k \tag{30}$$

where m is a number less than T that varies per design vector, a_0, a_j, and b are scalar functions that vary for each design at each iteration, and $\vec{u}$ is a randomized vector comprised of the user-defined constant α, the uniformly distributed random number R_1 ϵ [−0.5,0.5], and the constant L_k, which is the width of the domain along a coordinate direction of the design space, denoted by $\hat{e}_k$. The scalars a_0, a_j, and b are derived from the expressions,

$$\begin{aligned} &\mathit{if}\ \ U\left(\vec{x}_j^{\,g}\right) < U\left(\vec{x}_i^{\,g}\right)\\ &\quad \vec{x}_i^{\,g,new} = \vec{x}_i^{\,g} + \beta_{ij}\left(\vec{x}_j^{\,g} - \vec{x}_i^{\,g}\right) + \vec{u} \end{aligned} \tag{31}$$

$$\beta_{ij} = (\beta_0 - \beta_{\min})e^{-\gamma r_{ij}^2} + \beta_{\min} \tag{32}$$

where β_0, β_{min}, γ are user-defined scalars, and r_{ij} is the Euclidean distance between $\vec{x}_j^g$ and $\vec{x}_i^g$. Note that $\vec{x}_i^{g,new}$ is not the same as $\vec{x}_i^{g+1}$. FFA checks the condition in (31) for all pairs of design vectors $i \neq j$, and displaces $\vec{x}_i^g$ each time the condition is true. FFA only evaluates the objective function after each design has been displaced according to (31).

For example, suppose $T=3$, and the sample is sorted from lowest to highest objective function value. We now loop over the sample to apply (27). In order to compute (27), each design must be checked against every other design and (31) must be applied as many times as the condition holds. For $\vec{x}_1^g$, the condition always fails, therefore it is not displaced. For $\vec{x}_2^g$, however, the condition is true once (when it is compared to $\vec{x}_1^g$). Therefore, $\vec{x}_2^{g,new}$ becomes,

$$\vec{x}_2^{g,new} = \vec{x}_2^g + \beta_{21}\left(\vec{x}_1^g - \vec{x}_2^g\right) + \vec{u} \tag{33}$$

Now consider $\vec{x}_3^g$. In this case, the first time the condition is triggered ($\vec{x}_3^g$ vs. $\vec{x}_2^g$), $\vec{x}_3^{g,new}$ becomes,

$$\vec{x}_3^{g,new} = \vec{x}_3^g + \beta_{32}\left(\vec{x}_2^g - \vec{x}_3^g\right) + \vec{u} \tag{34}$$

When $\vec{x}_3^g$ is compared to $\vec{x}_1^g$, the condition is triggered again (recall that the objective function was not evaluated on $\vec{x}_3^{g,new}$). Let us call the previous result from (34), $\vec{x}_3^*$

$$\vec{x}_3^{g,new} = \vec{x}_3^* + \beta_{31}\left(\vec{x}_1^g - \vec{x}_3^*\right) + \vec{u} \tag{35}$$

$$\vec{x}_3^{g,new} = \vec{x}_3^g + \beta_{32}\left(\vec{x}_2^g - \vec{x}_3^g\right) + \vec{u} + \beta_{31}\left(\vec{x}_1^g - \vec{x}_3^g - \beta_{32}\left(\vec{x}_2^g - \vec{x}_3^g\right) - \vec{u}\right) + \vec{u} \tag{36}$$

Therefore, the reader can confirm that,

$$\vec{x}_1^{g+1} = \vec{x}_1^g \tag{37}$$

$$\vec{x}_2^{g+1} = \underbrace{\beta_{21}\vec{x}_1^g + (1-\beta_{21})\vec{x}_2^g}_{\vec{w}} + \vec{u} \tag{38}$$

$$\vec{x}_3^{g+1} = \underbrace{\beta_{31}\vec{x}_1^g + (1-\beta_{31})\beta_{32}\vec{x}_2^g + (1-\beta_{32}+\beta_{31}\beta_{32}-\beta_{31})\vec{x}_3^g}_{\vec{w}} + \underbrace{(2-\beta_{31})}_{b}\vec{u} \tag{39}$$

The algorithm's reference-point elitism is made explicit in (37). PSO's reference point elitism would operate the same way were it not for the inertia term. Notice also in (38) and (39) that the magnitude of the perturbation vector is also a function of the design vector's rank (in terms of its objective function value). This function becomes approximately linear when the sample is spaced very far apart. Therefore, if the design vector in question is located far from the rest of the sample, and the sample size is large, it can experience very large fluctuations (especially if it is the worst design in the sample), but if the design vector is close to the rest of the sample, the perturbations are governed by the values of α and L_k. The algorithm can be summarized as follows:

```
1. Initialize sample
2. While convergence criteria not met…
   a. For every design in the sample …
      i. Compare "this" design to "every other" design using the condition
         in (31)…
         A. If the condition is true, apply (31)
3. Check convergence criteria
```

As stated earlier, it is the inner loop (step *i*) that drives the computational expense of this algorithm. A MATLAB implementation of this algorithm can be downloaded from (Yang 2010b).

2.6 Differential evolution (DE)

DE is a family of algorithms. Although these algorithms contain the same essential structure, each of the variants discussed here fall into a different category. All of the DE algorithms use comparative updates, and none use recursion. A comparative update takes the form,

$$\begin{aligned} &if\ \ U\left(\vec{x}_i^{new}\right) < U\left(\vec{x}_i^g\right)\\ &\quad \vec{x}_i^{g+1} = \vec{x}_i^{new}\\ &else\\ &\quad \vec{x}_i^{g+1} = \vec{x}_i^g \end{aligned} \tag{40}$$

Strictly speaking, since every method uses a condition to form its new design vectors, these methods could be classified as hybrids. However, since the alternative of each condition is simply the unperturbed vector, it would not properly fall into the Group 1 category. Therefore, these methods will be classified based on their primary equation, which is a reasonable classification when the condition is triggered at a high frequency. The four forms of DE reviewed here include the so-called rand/1/bin (STD), proposed in (Storn and Price 1997), which is a Group 2 algorithm,

$$\vec{x}_i^{new} = \vec{x}_j^g + \vec{p}^g \tag{41}$$

$$\begin{aligned} &if \;\; R < CR, \;\; or \;\; k = k* \\ &\quad x_{i,k}^{new} = x_{j,k}^g + p_k^g \\ &else \\ &\quad x_{i,k}^{new} = x_{i,k}^g \end{aligned} \tag{42}$$

$$p_k^g = F\left(x_{a,k} - x_{b,k}\right) \tag{43}$$

best/2/bin (BST) also proposed in (Storn and Price 1997), which is a Group 3 algorithm,

$$\vec{x}_i^{new} = \vec{x}_c^g + \vec{p}^g \tag{44}$$

$$\begin{aligned} &if \;\; R < CR, \;\; or \;\; k = k* \\ &\quad x_{i,k}^{new} = x_{bestG,k}^g + p_k^g \\ &else \\ &\quad x_{i,k}^{new} = x_{i,k}^g \end{aligned} \tag{45}$$

$$p_k^g = F\left(x_{a,k} + x_{b,k} - x_{c,k} - x_{d,k}\right) \tag{46}$$

Donor3 (DN3), proposed in (Fan et al. 2003), which is a Group 4 algorithm,

$$\vec{x}_i^{new} = \vec{0} + \vec{p}^g \tag{47}$$

$$\begin{aligned} &if \;\; R < CR, \;\; or \;\; k = k* \\ &\quad x_{i,k}^{new} = p_k^g \\ &else \\ &\quad x_{i,k}^{new} = x_{i,k}^g \end{aligned} \tag{48}$$

$$p_k^g = \frac{R_1 x_{a,k} + R_2 x_{b,k} + R_3 x_{c,k}}{R_1 + R_2 + R_3} + F\left(x_{b,k} - x_{c,k}\right) \tag{49}$$

and Trigonometric DE (TRG), proposed in (Lampinen and Fan 2003), which, in this form, is a modification of rand/1/bin. This variant applies a "trigonometric mutation" equation with a certain user-defined probability, M_t, and Eqs. (42) and (43) with the complement of that probability. Since the trigonometric mutation is a *deterministic* combination of three randomly selected vectors, it will be treated as a modification of a Group 2 algorithm, rather than a hybrid of a Group 2 algorithm with a Group 4 algorithm. That is, if the same three vectors are selected for this operation, the end result will be the same. The probability of selecting the same three vectors increases as sample size decreases, therefore, in some applications, this operation can produced several copies of the same vector if used frequently. The presence of duplicates would be essentially zero if the operator contained a random number. The equations for TRG are,

$$\begin{aligned} &if \;\; R_M < M_t \\ &\quad \vec{x}_i^{new} = \vec{w} \\ &else \\ &\quad Apply \;\; (42), (43) \end{aligned} \tag{50}$$

where,

$$\begin{aligned} \vec{w} = \frac{1}{3}\left(\vec{x}_a^g + \vec{x}_b^g + \vec{x}_c^g\right) + (q_b - q_a)\left(\vec{x}_a^g - \vec{x}_b^g\right) \\ + (q_c - q_b)\left(\vec{x}_b^g - \vec{x}_c^g\right) + (q_a - q_c)\left(\vec{x}_c^g - \vec{x}_a^g\right) \end{aligned} \tag{51}$$

$$q' = \left|U\left(\vec{x}_a^g\right)\right| + \left|U\left(\vec{x}_b^g\right)\right| + \left|U\left(\vec{x}_c^g\right)\right| \tag{52}$$

$$\begin{aligned} q_a &= \left|U\left(\vec{x}_a^g\right)\right| / q' \\ q_b &= \left|U\left(\vec{x}_b^g\right)\right| / q' \\ q_c &= \left|U\left(\vec{x}_c^g\right)\right| / q' \end{aligned} \tag{53}$$

In the preceding equations, CR is a user-defined constant called the "cross-over rate," F is a user-defined scalar, R, R_1, R_2, R_3, and R_M are uniformly distributed random numbers ϵ [0,1], the subscripts a, b, c, and d denote that the vector was randomly selected from the sample, and the subscript k denotes the component of the vector being modified. The index $k*$ is randomly selected to guarantee that at least one component of each vector in the current sample is modified according to the DE equations. The majority of the randomness in the standard form of these equations is introduced through CR. Were it not for CR, the argument could be made that these equations should all be treated as essentially deterministic formulations (similar to the argument made for the TRG variation). While not proposed for the purpose of increasing randomization, three types of modifications, presented in (Inclan and Dulikravich 2013), draw more performance out of DE while minimizing the changes made to the implementation of DE.

2.6.1 Randomly varying parameters (-R)

Randomly varying both F and CR within some interval, rather than setting them as constants, dramatically improves the convergence speed and robustness of the DE methods in many test functions. Setting F as a random number causes the perturbation vectors to fully match the definition stated for this paper. The randomized values of F and CR used here were tuned on the SHC (Inclan and Dulikravich 2013). See also (Das et al. 2005, 2008) for discussions of similar ideas.

2.6.2 Special vectors (-V)

Including special vectors in the population (such as the weighted average) after each iteration has been shown to improve performance on a variety of problems due to symmetries within the function (Inclan and Dulikravich 2013). An average vector, and a weighted average vector are used for this purpose. This modification is similar to the TRG modification in that it is a deterministic formula. However, these particular special vectors are introduced to the new sample by replacing the worst designs of the current sample.

2.6.3 Sorted comparisons (-S)

DE normally compares a newly created design to an original design based on the order in which it was generated. This technique sorts the newly created population from best to worst, and the original population from worst to best before executing the comparison. This modification was also applied to every variation of DE. For example, rand/1/bin with sorted comparisons will be denoted as STD-S.

The DE family of algorithms is structured as follows:

```
1. Initialize sample
2. Create empty container of size T to store $x^{new}$ sample
3. While convergence criteria not met…
  a. For every design in the sample …
    i. Apply (41-53 depending on type)
    ii. If new design is superior to current design, replace, as in (40)
4. Check convergence criteria
```

The use of comparative updates eliminates the need to store the global best vector separately. Therefore, DE requires more memory than FFA, which does not save a secondary sample, but less than the other algorithms presented thus far.

2.7 Bat-inspired algorithm (BAT)

The BAT algorithm discussed here was originally proposed in (Yang 2010c) (not to be confused with (Malakooti et al. 2012)). In essence, this is a method that merges together a form of PSO (Group 1) with a type of Group 3 algorithm. It also utilizes recursion (an inertia term identical in form to sociability-only PSO), reference-point elitism, and comparative updates.

$$\begin{aligned} &\mathit{if}\ \ R_1 \le r \\ &\quad \vec{x}_i^{\,new} = \vec{x}_i^{\,g} + \vec{p}_1^{\,g} \\ &\mathit{else} \\ &\quad \vec{x}_i^{\,new} = \vec{x}_c^{\,g} + \vec{p}_2^{\,g} \end{aligned} \tag{54}$$

$$\vec{p}_1^{\,g} = \vec{p}_i^{\,g-1} + f\left(\vec{x}_i^{\,g} - \vec{x}_{best,G}^{\,g}\right) \tag{55}$$

$$f = f_{\min} + (f_{\max} - f_{\min})R_2 \tag{56}$$

$$\vec{x}_c^{\,g} = \vec{x}_{best,G}^{\,g} \tag{57}$$

$$\vec{p}_2^{\,g} = A_1 R_3 \tag{58}$$

where r, A_1, f_{min}, and f_{max} are user-defined scalars, R_1, and R_2 are uniformly distributed random numbers $\in [0,1]$, and R_3 is a uniformly distributed random number $\in [-1,1]$. Note that (55) causes the algorithm to search some distance from $\vec{x}_i^{\,g}$ in the direction away from the global best (the direction opposite sociability-only PSO) when inertia is zero, while (58) causes BAT to search in the vicinity of the global best using a random walk. Similar to DE, the BAT comparative update condition is,

$$\begin{aligned} &\mathit{if}\ \ U\left(\vec{x}_i^{\,new}\right) < U\left(\vec{x}_i^{\,g}\right)\ \mathit{and}\ \ R_4 < A_2 \\ &\quad \vec{x}_i^{\,g+1} = \vec{x}_i^{\,new} \\ &\mathit{else} \\ &\quad \vec{x}_i^{\,g+1} = \vec{x}_i^{\,g} \end{aligned} \tag{59}$$

where R_4 is a uniformly distributed random number $\in [0,1]$. It is unclear from (Yang 2010c) whether or not the A_1 in (58) is different from A_2 in (59), but the MATLAB implementation of the code, found in (Yang 2012), suggests that they are different.

The structure of this algorithm brings to light an interesting feature about some of the operations used in these algorithms. Reference-point elitism and comparative updates are certainly not mutually exclusive. However, comparative updates can easily decouple the recursion term from the design vector it is supposed to reflect. In this algorithm, the perturbation term is always computed according to (55), regardless of which condition in (54) was triggered. Therefore, as the algorithm proceeds, if the second condition is triggered rather than the first, the recursive term no longer has the same meaning that it would in PSO. Additionally, if $\vec{x}_i^{\,new}$ is rejected during the comparative update, the perturbation is totally decoupled from $\vec{x}_i^{\,g}$ because it no longer reflects the true perturbation of $\vec{x}_i^{\,g}$ during the iteration prior to the comparison.

The algorithm's structure can be summarized as follows,

```
1. Initialize sample
2. Create empty container of size T to store $x^{new}$ sample
2. While convergence criteria not met…
  a. For every design in the sample …
    i. Compute $\vec{p}_1^{\,g}$ according to (55)
    ii. Execute condition in (54) and apply appropriate equation
    iii. Execute condition (59) and, if true, replace current design vector
       with new design
3. Check convergence criteria
```

This very compact algorithm utilizes the same amount of memory as DE.

2.8 Cuckoo search (CKO)

The final algorithm considered here is CKO (Yang and Deb 2010). It is a Group 1 algorithm that uses DE-style comparative updates. Recall that these have the form,

$$\begin{aligned}&if\ \ U\left(\vec{x}_i^{\,new}\right) < U\left(\vec{x}_i^{\,g}\right)\\ &\quad \vec{x}_i^{\,g+1} = \vec{x}_i^{\,new}\\ &else\\ &\quad \vec{x}_i^{\,g+1} = \vec{x}_i^{\,g}\end{aligned}$$

The main drawback of this algorithm is that it requires two objective function evaluations per iteration. The first update, called the Lévy flight operation, is executed according to (60-61),

$$\vec{x}_i^{\,new} = \vec{x}_i^{\,g} + \vec{p}_i^{\,g} \tag{60}$$

$$p_{i,k}^g = 0.01\frac{\sigma N_1 N_3}{|N_2|^{1/\beta}}\left(x_{i,k}^g - x_{bestG,k}^g\right) \tag{61}$$

where, N_1, N_2, N_3 are normally distributed random numbers centered at zero, β is a user-defined parameter, k is the dimension number, and σ is a scalar function of β, depicted in Fig. 2,

The σ is a discontinuous, complex-valued function, but for $\beta \in [1,1000]$, the real values of $\sigma \in (0,8)$ as computed using the MATLAB 7.10.0 R2010a implementation of the gamma function. For all practical purposes, is can be assumed that σ increases smoothly with β for small, positive values of β, and remains $O(1)$.

As with QPS, the combination of random numbers in (61) results in a new distribution. Let us collect the coefficients in (61) into one coefficient, σ_N. Rewriting (61),

$$p_{i,k}^g = \sigma_N\left(x_{i,k}^g - x_{bestG,k}^g\right) \tag{62}$$

$$\sigma_N = 0.01\frac{\sigma N_1 N_3}{|N_2|^{1/\beta}} \tag{63}$$

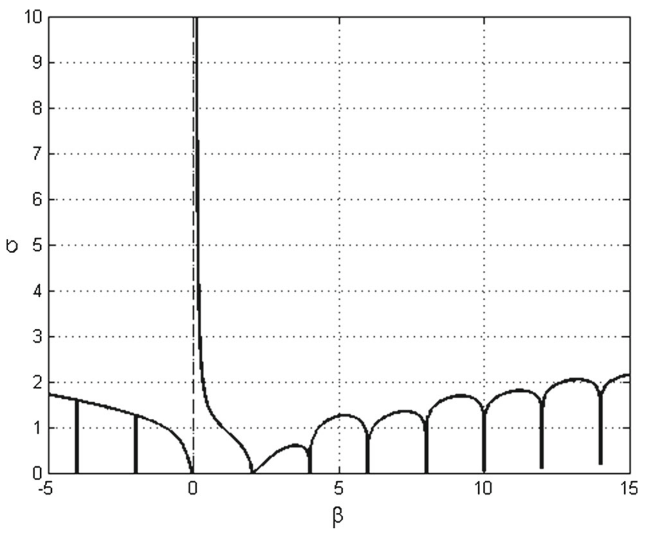


Fig. 2 Plot of σ vs. β. Complex valued σ not shown. The non-uniqueness and discontinuities of this function indicate the unusually significant impact tuning β can have on CKO performance

Although σ_N varies with β, users typically implement MGOAs with uniquely specified constants. Suppose the user sets $\beta = 1.5$, as recommended in (Yang and Deb 2010). Sampling from σ_N yields (Fig. 3).

The resulting distribution returns values very close to zero with very high frequency. The implementation of CKO used here was written in C++ and uses uniformly distributed random numbers (R_1, R_2, and R_3) bounded by [−1.1, 1.1] in the place of N_1, N_2, and N_3. This form is called σ_U, and is given by,

$$\sigma_U = 0.01\frac{\sigma R_1 R_3}{|R_2|^{1/\beta}} \tag{64}$$

The distribution of σ_U is sharper than σ_N, but produces very similar behavior near zero. Thus, in general, perturbation Eq. (61) moves the design point some very small distance toward or away from the global best. It has the same form as (55) from BAT, without recursion.

The second update in CKO, called the "empty nest" operation, is based on a condition with (65-66),

$$\begin{aligned}&if\ \ R_4 > q\\ &\quad \vec{x}_i^{\,new} = \vec{x}_i^{\,g} + \vec{p}_i^{\,g}\end{aligned} \tag{65}$$

$$\begin{aligned}&R_4 > q\\ &\quad \vec{x}_i^{\,new} = \vec{x}_i^{\,g} + \vec{p}_i^{\,g}\end{aligned} \tag{66}$$

where q is a user-defined scalar, R_4 and R_5 are uniformly distributed random numbers $\in$ [0,1], and the subscripts a and b denote vectors randomly selected from the current sample. If the same design vector is selected for both the Levy Flight and Empty Nest operations, and retained by the algorithm, then

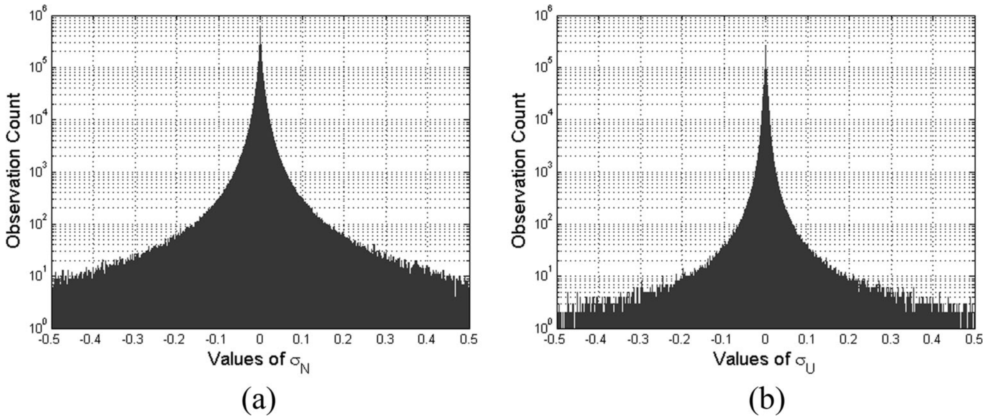


Fig. 3 Histogram of **a** σ_N *for* $\beta = 1.5$, and **b** σ_U *for* $\beta = 1.5$, based on 10 million random samples. Although **b** is more sharply peaked than **a**, the similarity in these plots demonstrates that (64) can be used in lieu of (63) without dramatically altering the behavior of CKO

the final form of the update for that design vector at the end of the iteration is,

$$\vec{x}_i^{\,g+1} = \vec{x}_i^{\,g} + \sigma_N\left(\vec{x}_i^{\,g} - \vec{x}_{bestG}^{\,g}\right) + R_5\left(\vec{x}_a^{\,g} - \vec{x}_b^{\,g}\right) \qquad (67)$$

which has the same form as the equation for the "current-to-best" variant of DE (Storn 1996). The complete CKO algorithm can be summarized as follows,

```
1. Initialize sample
2. Create empty container of size T to store $x^{new}$ sample
3. While convergence criteria not met…
  a. For every design in the sample …
    i. Apply (60),(61)
    ii. If new design is superior to current design, replace, as in (40)
    iii. Execute condition (65), and if true, apply the equation (65),(66)
    iv. If new design is superior to current design, replace, as in (40)
4. Check convergence criteria
```

The question of when the additional computational expense of separating (67) into two steps is warranted will be the subject of future research. Interested readers are referred to (Yang 2010d) for a MATLAB implementation of CKO.

3 Numerical results

3.1 Numerical experiment setup

A numerical experiment is the application of an optimization algorithm to a test case in the SHC (test cases 85, 356, 360, and 365 were omitted). For each numerical experiment, the optimization algorithm was executed for 200 iterations. In order to establish statistical significance, the numerical experiments (per test case, and per algorithm) were repeated 50 times, each with a new, randomly generated initial population.

Tuning an algorithm's parameters to a set of test cases can make it difficult to generalize conclusions drawn from those experiments (Rardin and Uzsoy 2001). To further complicate matters, recall that as a consequence of the No Free Lunch Theorem (Wolpert and MacReady 1997), "even a computer program (implementing an Evolutionary Algorithm) containing programming errors can perform better than some other highly tuned algorithms for some test functions" (Oltean 2004). Recall that this work seeks to measure performance based on group type and treats this measurement as a possible indicator of fundamental compatibility between an algorithm and a test case. An example of fundamental compatibility would be an algorithm that guarantees the identification of a descent direction on a convex optimization problem because such a property would make convergence to the global minimum inevitable for a range of user-defined parameters. An example of trivial compatibility would be any algorithm that converges to the global minimum solely because the right user-defined parameters were luckily selected. Therefore, tuning each algorithm to the whole SHC (rather than individual test cases) is used here solely for the purpose of making performance across the SHC more easily attributable to group type. However, exhaustively tuning each algorithm, as in (Pedersen 2010), is prohibitively computationally expensive for this research, outside its scope, and would require more thorough testing to determine whether or not observations based on those parameters can be generalized. The values of the user-defined parameters provided for each algorithm in Table 1. With the exception of DE variants -V, -S, and -VS, all algorithms are tuned to the SHC. The un-tuned DE variants use parameters suggested in literature because the variants that include -R are their tuned counterparts. In some cases, the tuned algorithms were close enough to their published recommended values that the published values were used instead. The references are provided to facilitate comparisons.

For the reader's convenience, the group numbers for each algorithm are provided in Table 2. Recall that the objective functions used here have a global minimum value of zero. If an algorithm converges to an objective function value of 10^{-7} or lower, it is deemed to have found the global minimum. In this sense, the objective

Table 1 User-defined parameter values for each algorithm, including modified algorithms

Algorithm/Acronym	Parameter values	References
PSO *Particle Swarm*	$\alpha = 0.5$, $\beta = \gamma = 2$ *initial perturbation = 0*	(Inclan et al. 2013)
QPS *Quantum-Behaved Particle Swarm*	α decreases linearly from 1 to 0.5, per iteration $\beta = 2.95$, $\gamma = 1.16$	(Sun et al. 2007)
MQP *Modified Quantum-Behaved Particle Swarm*	α decreases linearly from 0.83 to 0.4, per iteration $\beta = 0.96$, $\gamma = 2.71$	(Sun et al. 2007)
FFA *Firefly Algorithm*	α decreases linearly from 0.563 to 0, per iteration $\gamma = 0{,}66$, $\beta_0 = 2.88$, $\beta_{min} = 0.35$	(Yang 2009)
BAT *Bat-Inspired Algorithm*	A_2 decreases exponentially from 0.25 to 0, per iteration $f_{\min} = 0.6757$, $f_{\max} = 0.7902$, $A_1 = 0.2792$, $r = 0.1405$	(Yang 2010c, 2012)
CKO *Cuckoo Search*	$\beta = 1.6246$ ($\sigma \approx 0.6103$), $q = 0.0111$	(Yang and Deb 2010)
DE (type – modification) *Differential Evolution Family of Algorithms*	Recall modification labels are: (-R) randomized parameters, (-V) special vectors, and (-S) sorted comparisons. Randomized parameters are recalculated for each design vector, each iteration.	(Inclan and Dulikravich 2013)
STD, STD-V, STD-S, STD-VS *rand/1/bin*	$F = 0.8$, $CR = 0.9$, (no tuning performed)	
STD-R, STD-RS, STD-RV, STD-RVS *rand/1/bin*	$F \in [0.4,0.8]$, $CR \in [0.7,0.9]$	
BST, BST-V, BST-S, BST-VS *best/2/bin*	$F = 0.8$, $CR = 0.9$, (no tuning performed)	
BST-R, BST-RS, BST-RV, BST-RVS *best/2/bin*	$F \in [0.2,0.8]$, $CR \in [0.6,1]$	
DN3, DN3-V, DN3-S, DN3-VS *Donor 3*	$F = 0.8$, $CR = 0.9$, (no tuning performed)	
DN3-R, DN3-RS, DN3-RV, DN3-RVS *Donor 3*	$F \in [0.4,0.8]$, $CR \in [0.4,0.8]$	
TRG, TRG-V, TRG-S, TRG-VS *Trigonometric*	$F = 0.8$, $CR = 0.9$, (no tuning performed)	
TRG-R, TRG-RS, TRG-RV, TRG-RVS *Trigonometric*	$F \in [0.4,0.8]$, $CR \in [0.7,0.9]$	

function is being used as a measure of accuracy, although, the true measure of accuracy is the Euclidean distance between the converged design vector and the true global minimum design vector. Since this paper focuses on large trends over hundreds of test cases, the objective function measure of accuracy will suffice.

3.2 General results

3.2.1 Highest accuracy and convergence rate by group – all algorithms

The first performance metric to be considered is the highest mean accuracy and highest convergence rate (HAHC). This metric measures performance on a given objective function by identifying the algorithm that was both the fastest (converged in the fewest number of objective function evaluations), and most accurate (lowest objective function value) on average for a given problem. The algorithm with HAHC is the algorithm that attained the lowest mean objective function value for a given objective function. If multiple algorithms attained the same result, the algorithm with the fastest convergence rate is selected as the one with HAHC for that objective function. The algorithm's convergence is measured based on the objective function value of the global best design vector after each objective function evaluation. Recall that for CKO this means that its convergence rate *per iteration* is halved. Note, also, that this metric does not incorporate a Kolmogorov-Smirnov test to determine whether the converged results are statistically distinguishable (see Section 3.2.2 for additional comments on this).

Table 2 Algorithms listed by group number

Group Number	Algorithms
1	PSO, CKO
2	STD, TRG
3	BST
4	QPS, MQP, FFA, DN3
Hybrid of 1,3	BAT

Figure 4 shows the proportion of test cases for which any algorithm from a given group attained the HAHC. The most successful strategies were in Groups 2 and 3. Figure 5 shows the proportion of test cases for which the algorithms in each group attained the HAHC (the total of the bars in each sub-plot is the height of the bar for the corresponding group in Fig. 4). The single most successful algorithm was BST-RV (best/2/bin with randomized parameters that also copied the sample mean, and sample weighted mean into the sample for the next iteration). The five best performing variations of BST each attained

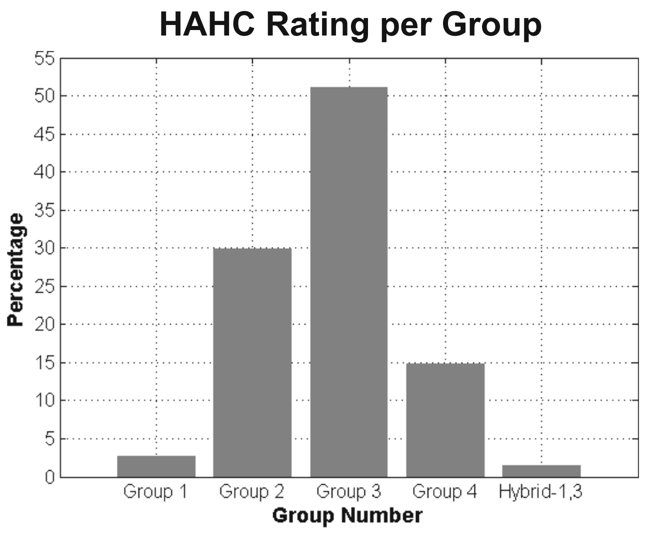


Fig. 4 Percentage of objective functions for which group attained highest mean accuracy and highest mean convergence rate (HAHC), averaged over 50 trials. Although based on mean values without regard for higher moments or statistical significance testing between algorithms, these general trends are found to hold in subsequent, more detailed tests. These large-scale trends suggest compatibility between Groups 2 and 3, and the SHC exists

success rates higher than any other single algorithm. Four of those five included randomized parameters, which clearly demonstrates the advantage of increasing the stochastic nature of this algorithm.

More importantly Figs. 4 and 5 suggests that of all the update equations presented here, there are two general strategies capable of solving a wide range of problems: (a) iteratively searching around the neighborhood of the global best design with some random distribution, and (b) iteratively searching around the neighborhood of subsets of the population with some random distribution. It is not necessary (as done in PSO and CKO) to perturb every design in the population, nor does using complicated sampling schemes necessarily improve overall performance.

Now consider the case where the algorithms are compared based on their form of elitism. Figure 6a demonstrates that in the case of reference-point elitism, Group 4 algorithms dominate PSO in performance, which further reinforces the conclusion that it is not necessary to perturb every design, every iteration to obtain good performance over a broad set of problems. However, none of the Group 4 algorithms utilize recursion, therefore more experimentation is required to distinguish its impact from that of reference-point elitism.

When comparative updates are considered (Fig. 6b), it appears that Group 1 algorithms can perform competitively against the other groups. However, since the CKO equation can be viewed as a special case of a DE variant, it is unclear

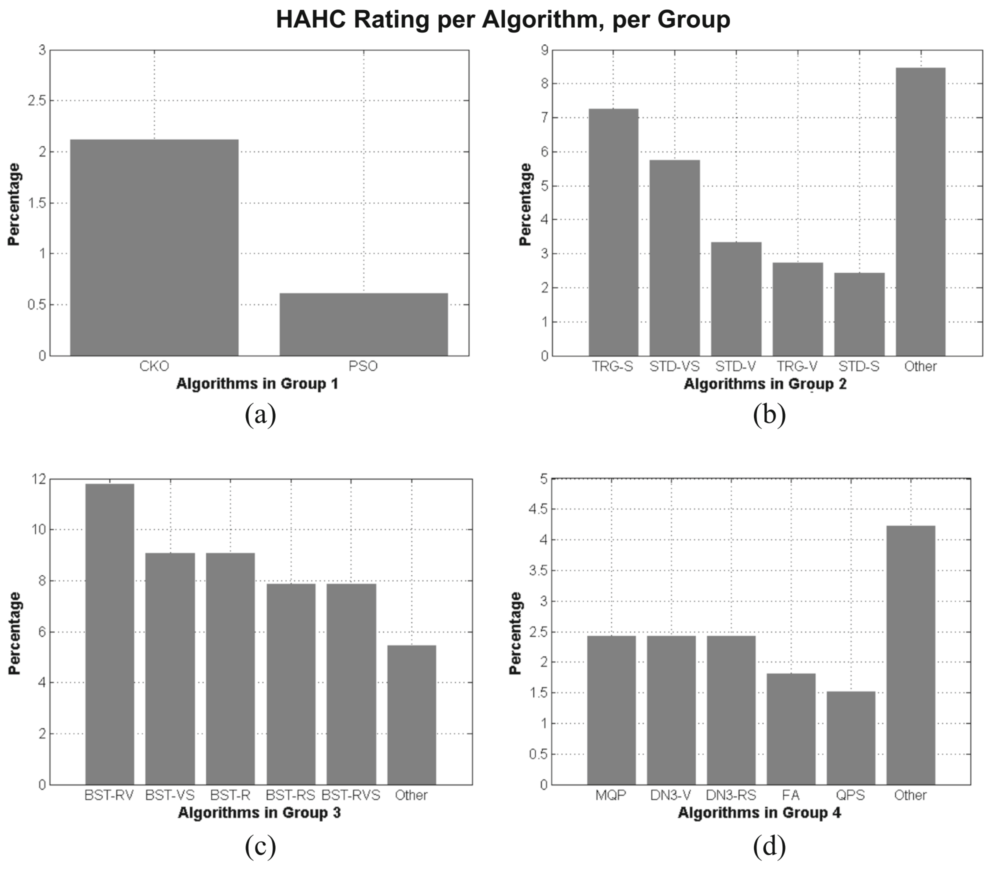


Fig. 5 Percentage of objective functions for which an algorithm attained the HAHC (averaged over 50 trials), listed by group number. These are a breakdown of the results summarized in Fig. 4. The performance of Group 3 is dominated by a minority of its algorithms, while Group 4 performance is more evenly distributed. Group 2 performance is also dominated by a minority of algorithms, but contains more variants than Group 3

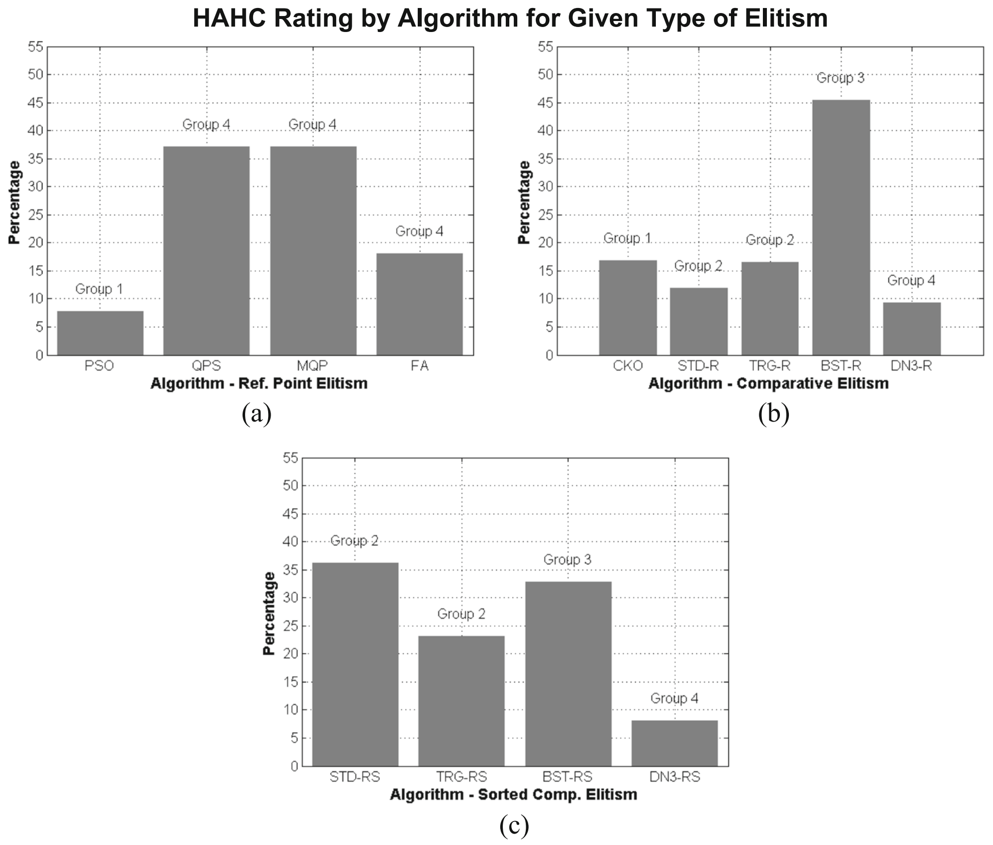


Fig. 6 Percentage of objective functions for which a subset of algorithms attained the HAHC (averaged over 50 trials), controlling for elitism: **a** reference point, **b** comparative updates, and **c** sorted comparative updates. Cases **b** and **c** also control for randomized parameters, and exclude special vectors in the DE algorithms. The success of Group 3 over Group 2 in Fig. 4 is due to algorithms with unsorted comparative elitism. Other forms of elitism do not significantly impact trends

whether its performance is competitive because it is Group 1, or because it is approaching the performance of DE on the SHC. Sorted comparisons, controlled also for randomized parameters and special vectors, (Fig. 6c) suggest that Group 2 and Group 3 algorithms perform strongly against Group 4. That is, searching around known points is likely to perform better than searching around a randomly-weighted average of points. Since only two algorithms utilize recursion, conclusions cannot be drawn for this discussion.

3.2.2 Highest accuracy convergence rate by group – single algorithm

Based on the analysis in 3.2.1, it is possible that the groups appear to do better than they ought, because there are so many small variations of DE in some groups, while other groups contain only one or two members. These slight variations may spread the group's viability over larger subsets of the SHC. An alternative approach is to use a single algorithm from each group to represent the performance of that group. Based on the information in Fig. 5, the top four algorithms from each group were selected (where possible) and compared to one-another using the HAHC rating. In addition to the HAHC rating, the Kolmogorov-Smirnov (KS) test was used to ensure that the algorithm with the apparent HAHC rating is statistically distinguishable from all other algorithms being evaluated. Here "statistically distinguishable" means that the null hypothesis of the KS-test is rejected for the distribution of the converged objective function values and is also rejected for the distribution of the number of iterations required to achieve convergence with a confidence interval of 5 %. Figure 7 summarizes the results of these analyses. In each plot, the only algorithms compared are those explicitly listed. If the KS test null hypothesis is not rejected for a particular algorithm, it is labelled "similar" to another algorithm. Due to similarities in performance, the HAHC rating bars do not sum to 100 %. In general, the trends in Fig. 7 strongly resemble those of Fig. 4 suggesting that the distinction between groups is founded, and holds at least for the top performing algorithms. In each case, Group 2 and 3 perform best on roughly 80 % of all test cases. Figure 7b shows the greatest overlap in the performance of two groups. In this case, STD-VS and BST-VS are statistically indistinguishable on 52 objective functions. Although this might suggest that the –VS modifications drive their performance more than their group structure, such similarities are also facilitated by the fact that Group 3 is a proper subset of Group 2.

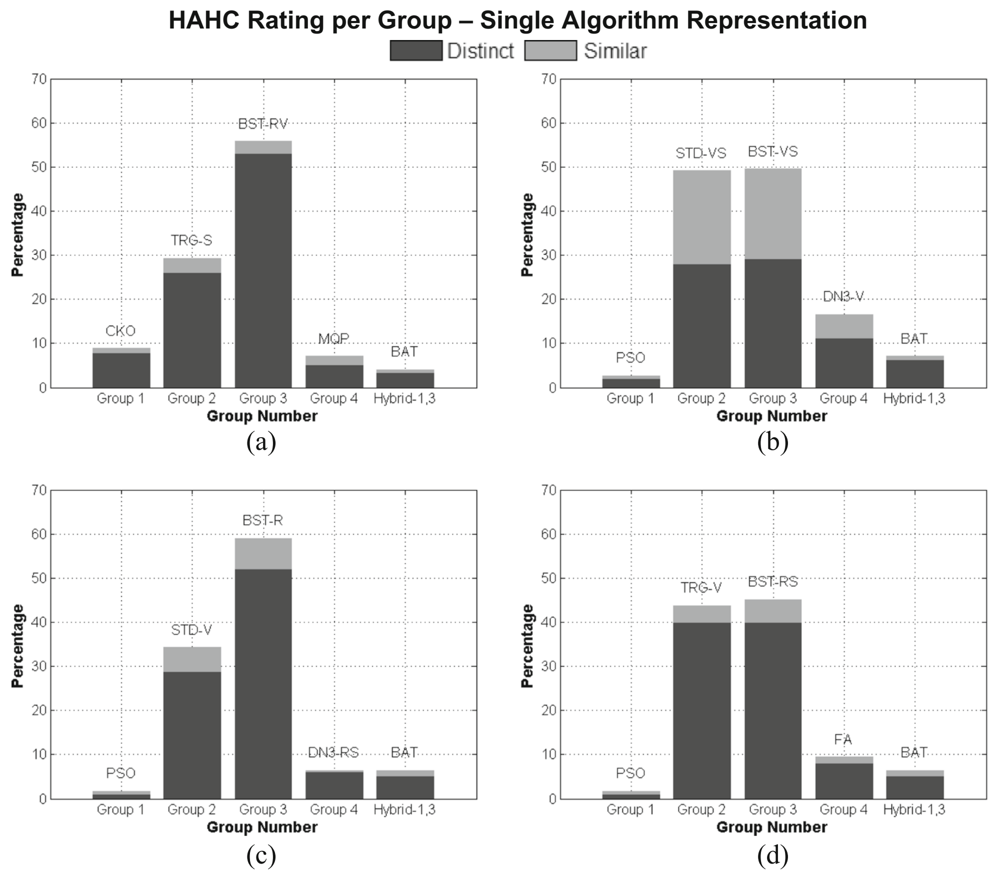


Fig. 7 Percentage of objective functions for which an algorithm, representative of its group, attained the HAHC (averaged over 50 trials). *Light grey* represents the test cases where performance in one group is statistically indistinguishable from another group. The algorithms in **b** use the VS modifications, and since Group 3 is a subset of Group 2, there is substantial overlap in performance. Trends consistent with Fig. 4 do not depend on algorithm

Other examples of similarity are much more specific. For example, the BAT algorithm is "similar" to BST-R, BST-RS, and BST-RV on test cases 45 and 234, as well as being similar to BST-VS on test case 234. In this case, the similarity is a direct consequence of the incorporation of Group 3 behavior into the BAT algorithm using (57).

3.3 Trends with problem dimension

The second performance metric to be considered is the manner in which a group's mean performance scales with problem dimension. Full histograms of each group's performance are provided in the Appendix, along with brief supplementary discussion. Consider a set of unconstrained second and fourth order polynomials with domains of [−300,300] in each dimension.

It is clear from Fig. 8 that Group 1 algorithms scale poorly on these four sets of test cases. Although some deterioration in performance is expected due to the curse of dimensionality (Kleywegt and Shapiro 2001), this poor performance can be attributed to incompatibility with the objective function. Whatever the performance of Groups 2-4 may be, they appear to scale much better with the exception of Group 3 on cases 303-305. Thus, we may begin to say that on even polynomials it is not necessary to iteratively perturb every point. The BAT algorithm does not outperform any method in this set. Furthermore, on test cases 294-299 it consistently converges to the same local minima, suggesting that the objective function "perfectly deceives" this algorithm, as pointed out in Section 2. More detailed statements will require further experimentation. For example, these experiments do not consider a group's sensitivity to the value of the function's coefficients. Furthermore, future experiments can use the BAT algorithm (or similar Hybrid 1,3 algorithms) to explore the transition between Group 1 performance and Group 3 performance on these functions to determine if the transition is smooth, or otherwise informative.

Sets of test cases for which the problem dimension increases by small factors reveal a pattern of behavior more complex than those indicated in Fig. 8. The BAT algorithm displays strongly non-monotonic behavior in Fig. 9a that, at first glance, appears to depend on whether or not the problem is of even or odd dimension. Such a dependency is uncommon, but can occur if an even dimension imposes symmetry that the algorithm implicitly exploits. If this hypothesis holds, then this convergence behavior can be attributed to trivial compatibility when the function is of even dimension. Further testing is required to determine if this is the case, and if so, why it does not appear to affect the other groups. Group 2 and 3 algorithms demonstrate the same superior

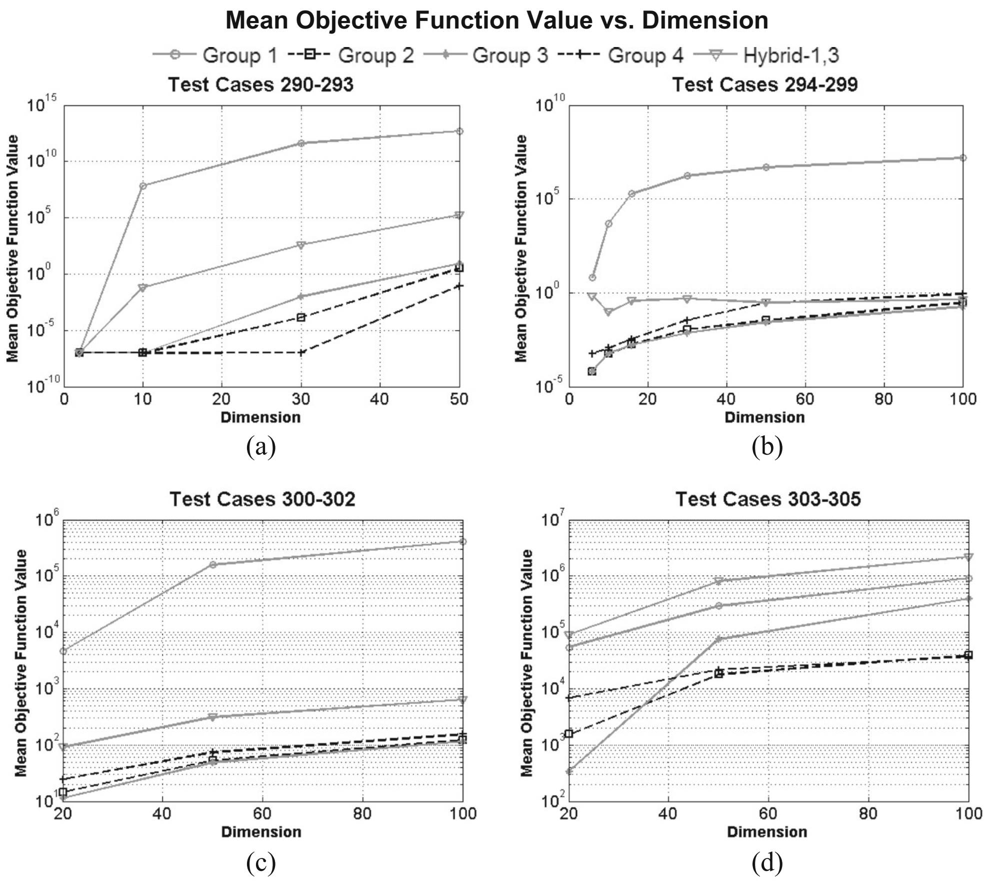


Fig. 8 Mean objective function value (averaged over 50 trials) obtained by groups on objective functions designed to scale with problem dimension (see Table 3). Values based on best-performing algorithm within group for each test case. Groups 2 and 4 show the lowest sensitivity to problem dimension, suggesting better compatibility. Hybrid-1,3 performs consistently in **b** suggesting it is trapped in local minima (the function is "deceptive")

performance in Fig. 9b and c that they did in the even, unconstrained polynomials of Fig. 8. However, the penalty function for Fig. 9c contains a linear term, causing the overall form of test cases 277–280 is a polynomial with step discontinuities. This appears to reduce the compatibility between Groups 2 and 3, and the objective functions as indicated by the increase in slope in their respective curves.

The results of this and the previous subsection appear to support the idea Group 3 algorithms in particular possess some advantage over the other groups on a large set of functions. A brief thought experiment will produce a hypothesis regarding why this may be the case: Suppose a second order polynomial of one variable with positive leading coefficient were sampled using a set of points (the precise distribution is immaterial), and that Jensen's inequality (a necessary condition of convexity) were computed on this sample. For any sample, the resulting test would yield "convex."

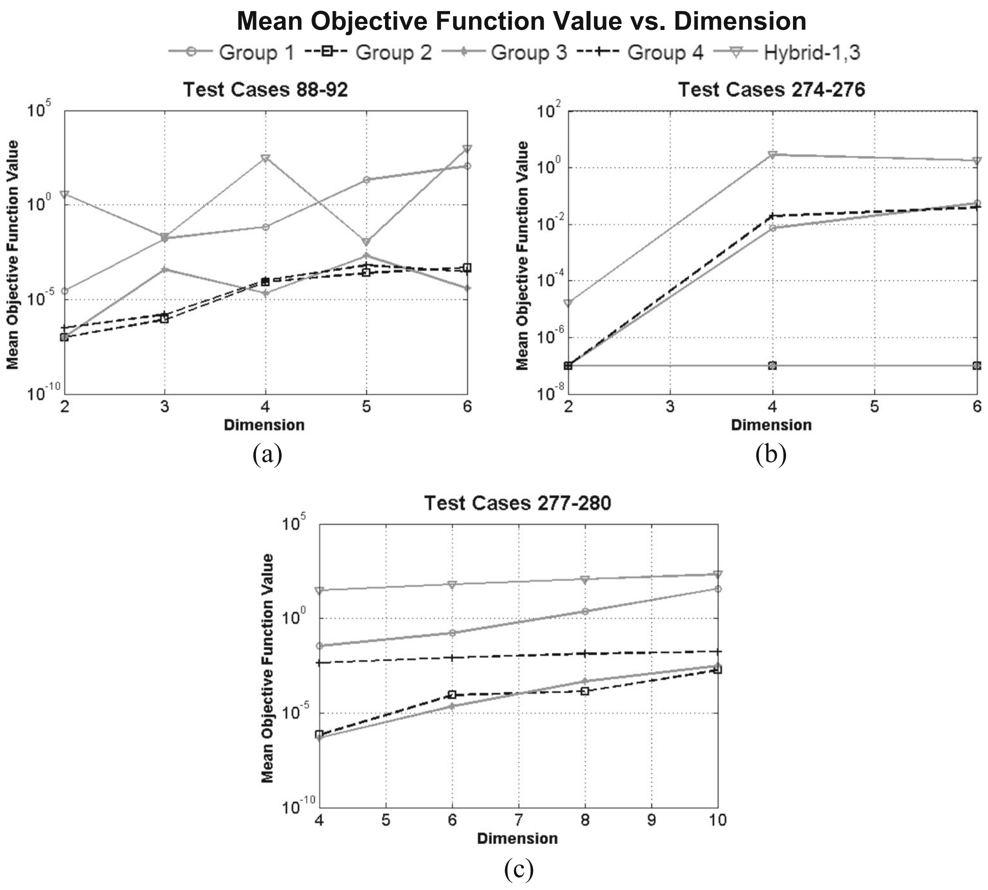
Now suppose that this process was repeated for a fourth order polynomial with some finite sample. Independent of the sample, and no matter which form of the fourth order polynomial, if the sample points are spaced "far enough" apart, there will always be a sample that will pass Jensen's inequality. In other words, if the sample points are spaced far enough apart, all fourth order polynomials appear convex to the human eye (and to Jensen's inequality) even when they are not truly convex.

The same can be said for all even polynomials and extended to large groups of functions with exponential terms. Note that the Group 3 algorithms used here all

Table 3 Objective functions depicted in Fig. 8. Note that (c) is the only second-order polynomial, and that all polynomials tend to positive infinity far from the origin

$U(\vec{x}) = \left(\sum_{i=1}^{\dim} i x_i^2\right)^2$	$U(\vec{x}) = \frac{1}{10{,}000}\sum_{i=1}^{\dim-1} 100\left(x_{i+1}-x_i^2\right)^2 + (1-x_i)^2$
(a)	(b)
$U(\vec{x}) = \dim + x_1^2 - 2x_1 + 2\sum_{i=2}^{\dim} x_i^2 - x_{i-1}x_i$	$U(\vec{x}) = \frac{1}{4}\left(\sum_{i=1}^{\dim} i x_i\right)^2 + \frac{1}{16}\left(\sum_{i=1}^{\dim} i x_i\right)^4 + \sum_{i=1}^{\dim} x_i^2$
(c)	(d)

Fig. 9 Mean objective function value (averaged over 50 trials) obtained by groups on objective functions designed to scale with problem dimension (see Table 4). Values based on best-performing algorithm within group for each test case. Due to their low sensitivity to problem dimension, Groups 2 and 3 are most compatible with these objective functions. Since Group 3 is a subset of Group 2, the incompatibility of other groups is further reinforced



sample around the global best design. One can conclude from intuition that such an algorithm will monotonically converge to the true global minimum of a convex function (assuming properly specified parameters, including sample size). Following intuition, one can imagine that such Group 3 algorithms possess superior convergence properties in non-deceptive situations where an objective function "appears" convex. One can also suppose that these algorithms will retain this advantage until the sample converges to a region in which the non-convexity becomes appreciable. The other groups, which do not have this advantage, may perform better in the nonconvex region, but due to the initial size of the search space simply fail to converge to that region. Thus, it appears Group 3 algorithms possess an advantage for functions that appear convex throughout the majority of their domain. However, to confirm that this is the case will require more studies to test a variety of hypotheses addressing questions such as: (a) Do all Group 3 algorithms outperform all other algorithms on convex problems, or it is a result of the algorithm's elitism, or its choice to perturb the global best rather than some other vector? (b) Which metrics capture this apparent convexity, and how can such metrics be applied to known functions to provide a statistical measure of this appearance?

Table 4 Objective functions depicted in Fig. 9. Note that (c) is the constrained version of (b), which causes a pronounced change in performance for Groups 2 and 3

$U(\vec{x}) = P(\vec{x}) - 1.3626568 + \sum_{i=1}^{\dim} x_i^2$ Constraint: discontinuous, heavily nonlinear, inequality (a)	$U(\vec{x}) = \vec{x} \bullet (H\vec{x})$ where H is the Hilbert matrix, and • is the dot product operation (b)
$U(\vec{x}) = P(\vec{x}) - f_{\min} + \vec{x} \bullet (H\vec{x})$ where H is the Hilbert matrix, and • is the dot product operation Constraint: linear inequality (c)	

3.4 Additional remarks

This preliminary investigation is an extension of research presented in (Inclan 2014). More studies are required, utilizing the guidelines presented in (Rardin and Uzsoy 2001), in order to quantify the contributions of elitism, group classification, and recursion with greater statistical rigor. However, this work highlights some fundamental challenges to the analysis being performed. For example, this work seeks to identify compatibility between algorithm and objective function without access to widely accepted metrics that quantify that compatibility. While not totally prohibitive, it means that there is no guarantee that any of the test cases used here are fundamentally compatible with any of the algorithms selected. Therefore, the conclusions drawn here from the analysis of broad trends will need to be analyzed in greater detail. For example, when examining how an algorithm scales with problem dimension, future studies should consider computing two values: (a) the trend in probability of attaining the global minimum with problem dimension, and (b) the trend of computational cost of successful instances with problem dimension. This will further quantify compatibility, since such a measure must reflect the actual probability of success. It would be informative to see how sensitive an algorithm's probability of success is to variation of the objective function's coefficients.

Another fundamental challenge to this analysis is the number and type of optimization algorithms included in each group. In addition to the "results-smearing" issue highlighted in 3.2.2, it was found during the course of this research that very different parameter settings can cause the same algorithm to perform well over large subsets of the SHC (e.g., QPS $\alpha_{max}=0.8$, $\beta=0.25$, $\gamma=0.19$). Clearly this suggests that more algorithms, such as the current-to-best DE variant, ought to be included. This also raises a number of possibilities for future work: (a) Consider whether or not to treat successful, but widely different parameter settings on an algorithm as a different variation of that algorithm, (b) construct additional variations based on the modifications already available including CKO-S, BAT-S, QPS with recursion, or PSO without recursion, etc., (c) construct variants of the same algorithm such that they fall into different groups, such as Group 1 FFA, (d) construct variants that are hybrids with tuning parameters that enable a smooth transition from full group A behavior to full group B behavior, and (e) construct algorithms that apply the various forms of elitism and convergence criteria to pure random-number generators using well-known distributions in order to further study their effects. The next concern will be how to decide which collection of variants to use in a particular analysis, how many, and why.

Once the set of algorithms has been determined, questions of statistical significance arise. For example, are there problems for which entire groups of algorithms appear statistically indistinguishable, and how does this relate to problem compatibility? If compatibility between a group/algorithm and a subset of the SHC is suspected (a subset representative of a larger class), these tools can be used to search for an algorithm from another group whose performance is statistically indistinguishable from the group/algorithm in question by using the probability of rejecting the null hypothesis as an objective function. If a second group is found to be indistinguishable from the first, it would suggest that some trait other than group type is leading to improved performance (possible examples of this are seen in Fig. 7). A second, and important aspect of statistical significance is the estimation of algorithm performance metrics within some confidence. A robust approach, outlined in (Shilane et al. 2008) was prohibitive for this work, but will be considered for future studies. For this work, the common approach of multiple executions was utilized not only due to practical considerations, but also because such hypothesis testing requires that specific algorithm traits and performance metrics be easily measureable so they can be correlated. This work is a step toward such experiments.

4 Conclusions and future work

By examining similarities in various global single-objective optimization algorithms and measuring their relative performance on a set of 295 analytical test cases, this research has shown that a small set of commonly used heuristics can produce relatively good global search behavior on a large set of distinct optimization problems. Specifically, the top performing heuristics on 80 % of the test cases are based on: (a) iteratively searching the neighborhood of the global best design with a random distribution of designs (that is, randomly perturbing the current best design), (b) iteratively searching the neighborhood of random subsets of the population with some random distribution (that is, randomly perturbing a subset of the population); and (c) retaining designs based on some superiority comparison measure (that is, a form of elitism).

Given the relationship between the individual heuristics and their effectiveness on different sets of problems, it will now become possible to begin answering the following questions: What mathematical qualities can be identified that define a problem as deceptive, or not, with respect to the heuristics presented here? Do Group 3 algorithms demonstrate superior performance on all of the convex problems in the SHC? If so, do the nonconvex problems on which Group 3 algorithms perform well also demonstrate some apparent convexity? If the "curse of dimensionality" is a symptom of algorithm-problem incompatibility, can we expect compatible algorithms to exhibit polynomial scaling, exponential scaling, or some other scaling?

With these questions in mind, possible avenues for future research are also presented such as incorporating more algorithms in each group, constructing algorithms with the ability to transition from one group to another, and utilizing statistical tests to identify problems on which multiple algorithms are statistically indistinguishable. Furthermore, improvements to the classification scheme ought to be further explored.

Acknowledgments The authors are grateful for receiving early versions of the Bat Algorithm, Firefly Algorithm and the Cockoo Search Algorithm from Prof. X.-S. Yang. Early part of this work was partially funded by the US Air Force Office of Scientific Research under grant FA9550-12-1-0440 monitored by Dr. Ali Sayir. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the US Air Force Office of Scientific Research or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon.

Appendix

To supplement the discussion in Section 3.3, this section presents the complete set of data used to create the plots in histogram form. The source codes of the algorithms, test cases, and MATLAB scripts used to generate these results will be available for download at http://maidroc.fiu.edu. Note that the distributions shown are that of the best algorithm in each group, for each objective function. In general, Group 3 algorithms appear to provide the best performance on second and fourth order, convex polynomials. The occasional success of Hybrid-1,3 suggests there is room for improvement over Group 3 on this class of objective functions (Figs. 10, 11, 12, 13, 14, 15, 16).

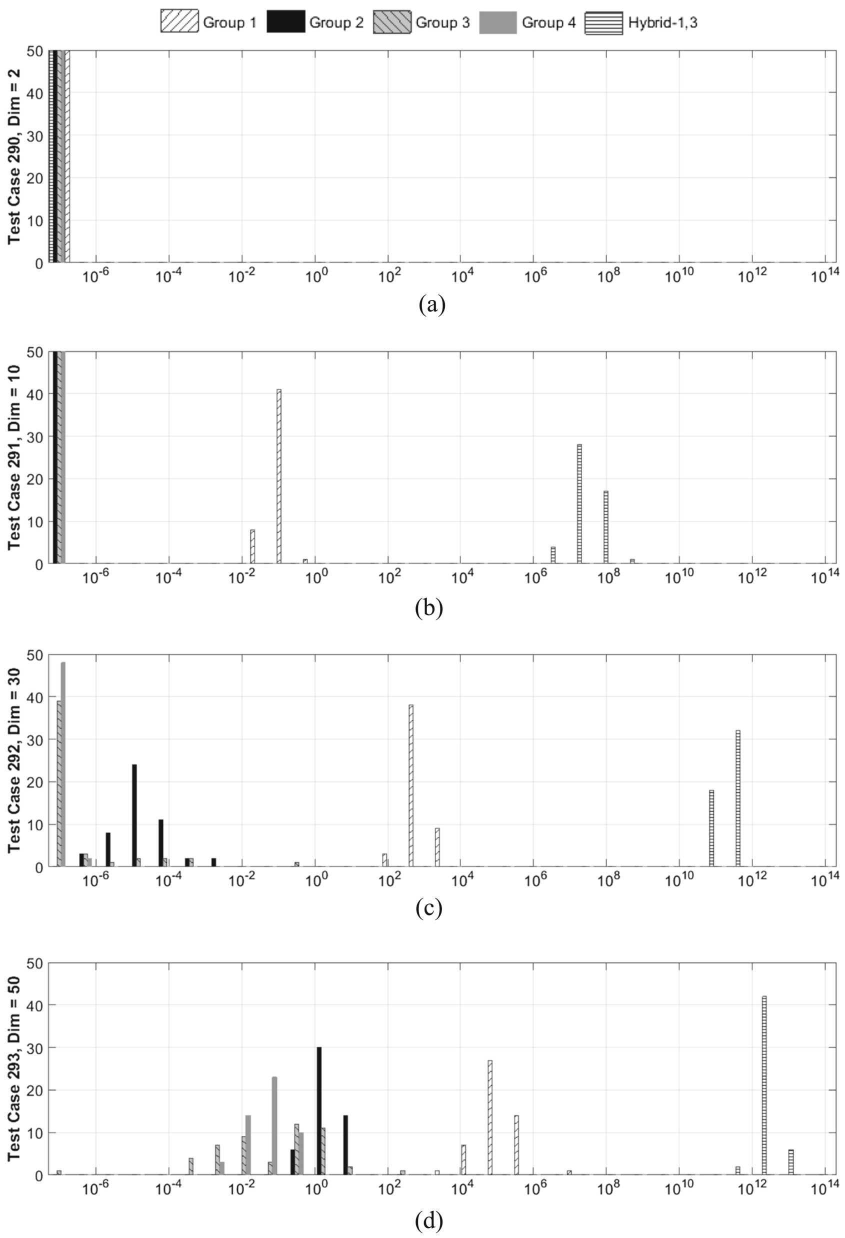


Fig. 10 Histograms corresponding to data shown in Fig. 8a. Count vs. objective function value. Distributions of converged objective function values for Groups 1–5 on a set of test cases that scale with dimension. Group 1 and the Hybrid-1,3 are very sensitive to problem dimension as shown by their rapid deterioration in performance relative to Groups 2–4. Groups 3 and 4 compete for best performance with the support for Group 3's distribution spreading much wider than that of Group 4. Group 3 retains its ability to find better solutions than Group 4, but at the cost of robustness

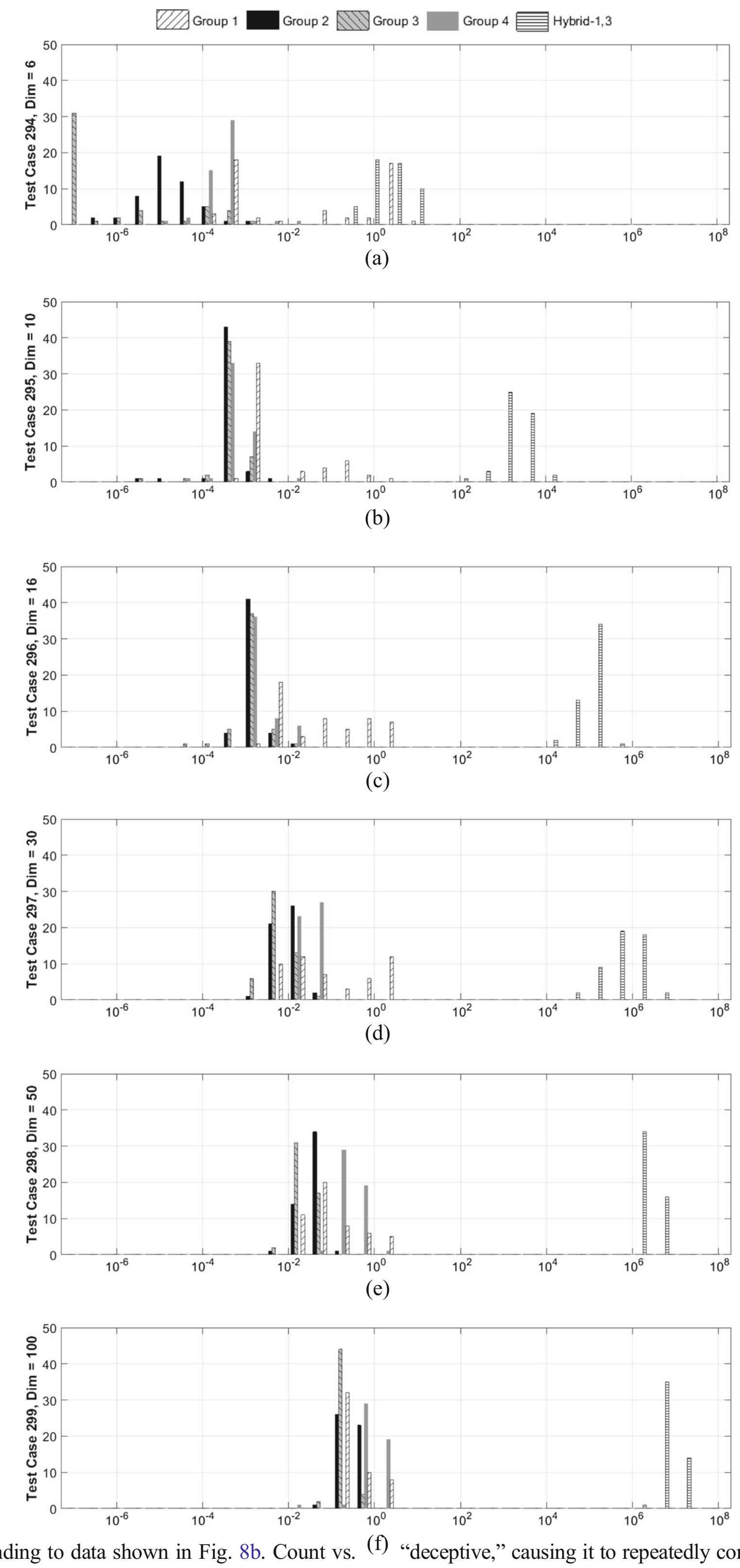


Fig. 11 Histograms corresponding to data shown in Fig. 8b. Count vs. objective function value. Distributions of converged objective function values for Groups 1–5 on a set of test cases that scale with dimension. Hybrid-1,3 converges consistently to the same range of objective function values (10^{-4} – 10^{0}). This suggests that the objective function is "deceptive," causing it to repeatedly converge to local minima. Group 1 scales very poorly, indicating incompatibility. The trends in Groups 2–4 strongly suggests that Group 3 behavior (searching only in the vicinity of the global best) is the most compatible strategy

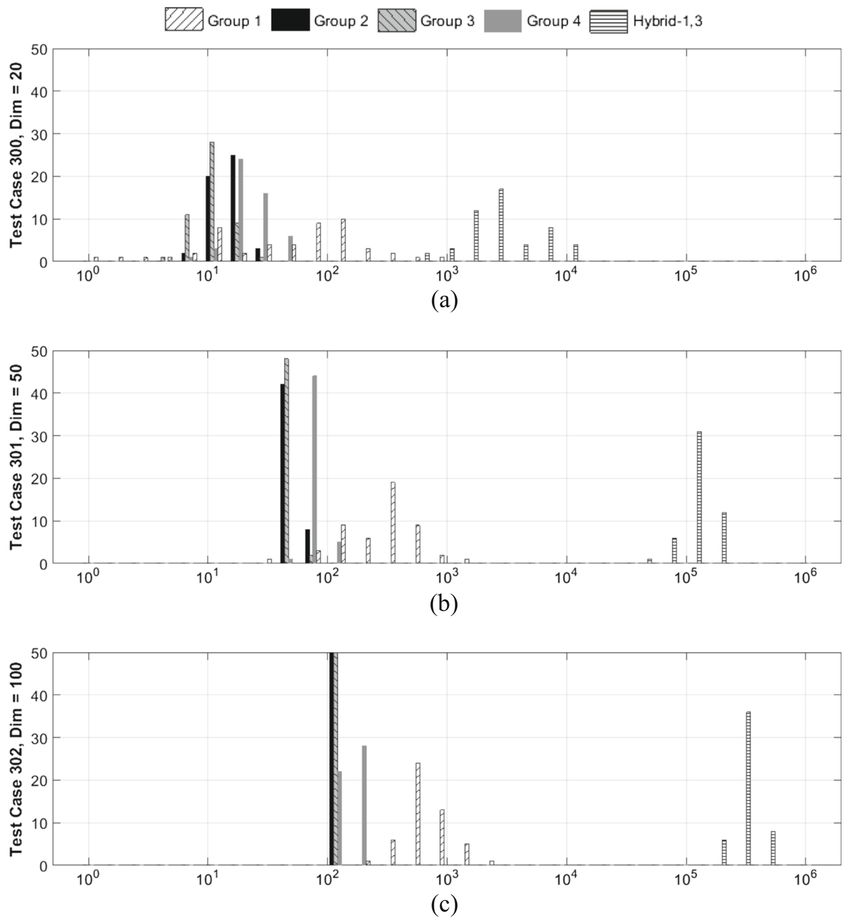


Fig. 12 Histograms corresponding to data shown in Fig. 8c. Count vs. objective function value. Distributions of converged objective function values for Groups 1–5 on a set of test cases that scale with dimension. Groups 2–3 have largely overlapping supports. Hybrid-1,3 has a much wider distribution, indicating worse robustness, but is the only algorithm capable of outperforming Group 3 in 20D and 50D. This suggests it may be a better starting point for designing an algorithm specifically tuned to this objective function than Group 3

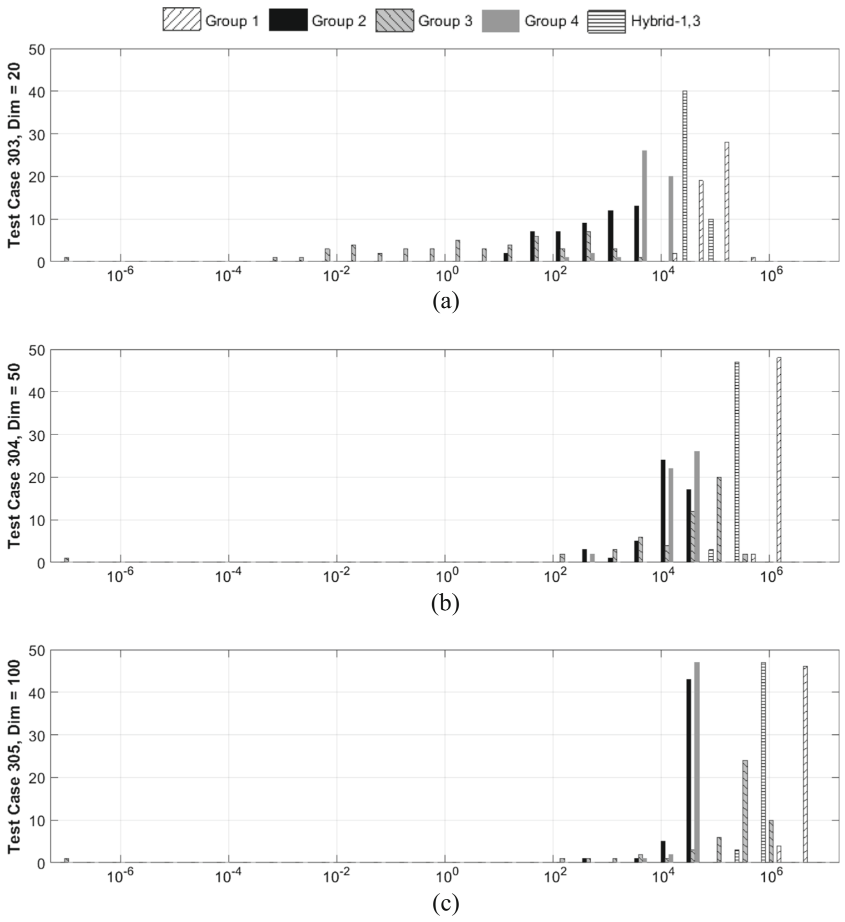


Fig. 13 Histograms corresponding to data shown in Fig. 8d. Count vs. objective function value. Distributions of converged objective function values for Groups 1–5 on a set of test cases that scale with dimension. Remarkably, the Group 3 algorithms consistently identify solutions near the global minimum in a very small percentage of cases. Generally, Group 3 rapidly deteriorates in performance with increasing problem dimension, while all other algorithms begin with poor performance and deteriorate further

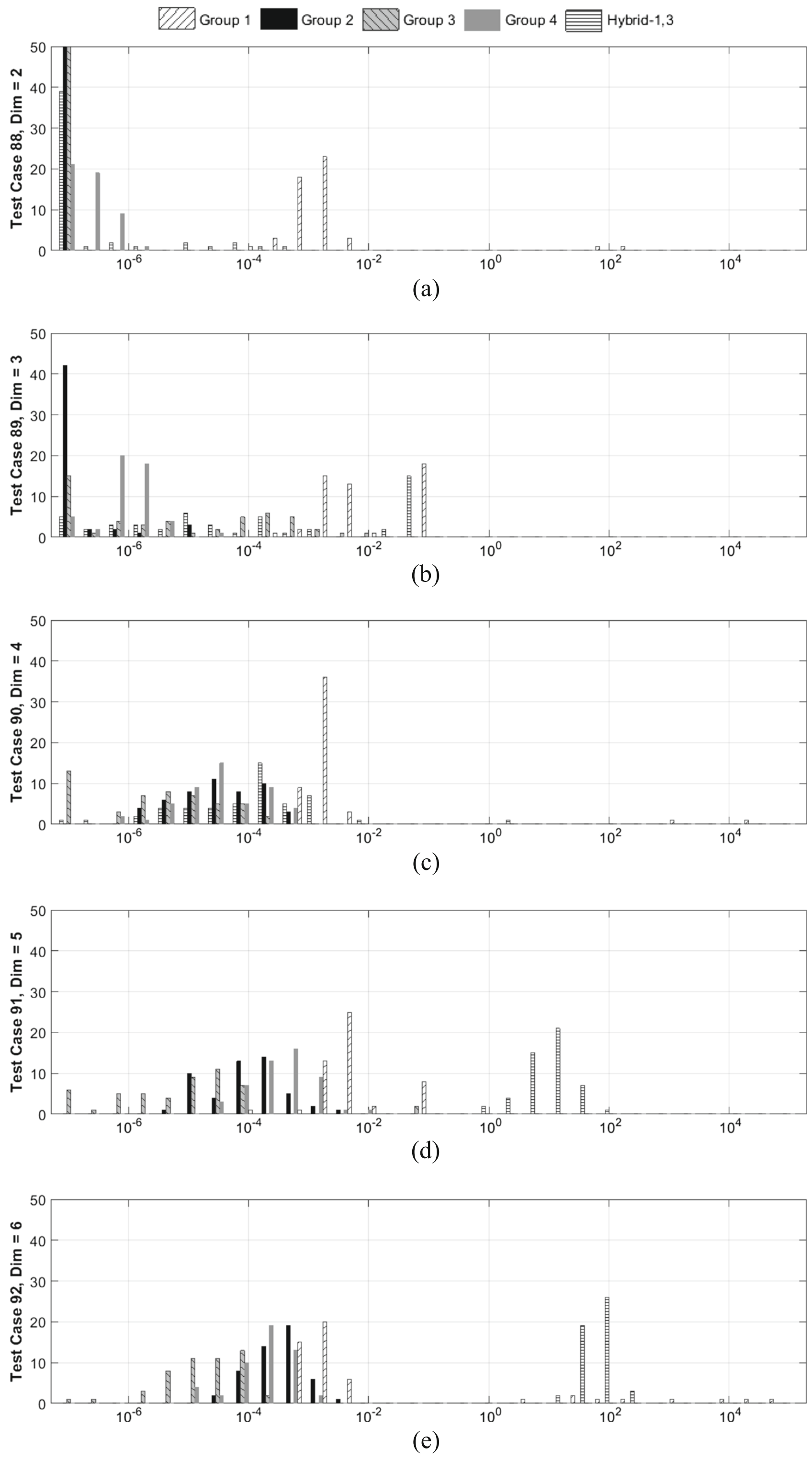


Fig. 14 Histograms corresponding to data shown in Fig. 9a. Count vs. Objective Function Value. Distributions of converged objective function values for Groups 1–5 on a set of test cases that scale with dimension. The presence of a highly nonlinear penalty function added to the quadratic polynomial objective function causes the distributions of every Group to spread wider relative the distributions seen in Figs. 10, 11, 12 and 13. An odd characteristic of Hybrid-1,3 on this problem is that its distribution contains outliers at very high objective function values for even-dimension problems only, suggesting that the algorithm is sensitive to problem symmetry

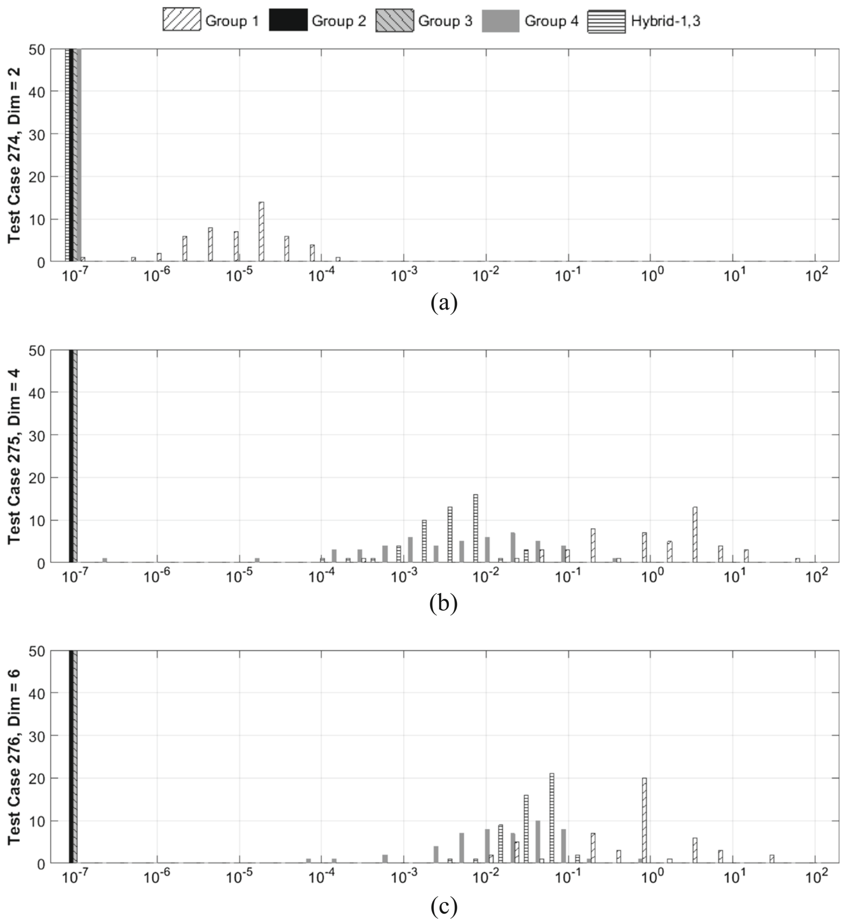
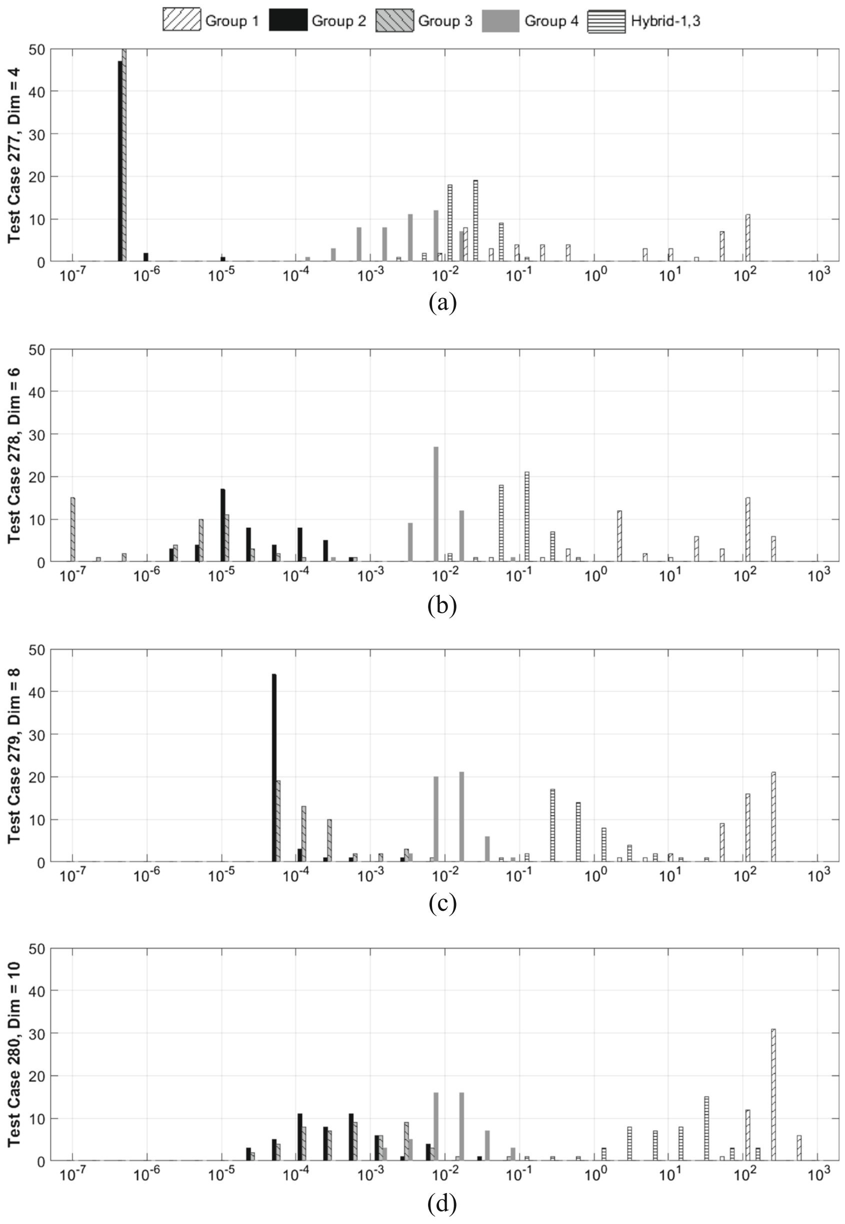


Fig. 15 Histograms corresponding to data shown in Fig. 9b. Count vs. objective function value. Distributions of converged objective function values for Groups 1–5 on a set of test cases that scale with dimension. Groups 2–3 appear ideally suited to the objective function based on this range of problem dimension. Groups 1, 4, and the Hybrid-1,3 are very sensitive to dimension. The behavior of Group 1 is consistent with previous unconstrained even-order polynomials, while Group 4 appears unusually sensitive compared to Groups 2 and 3

Fig. 16 Histograms corresponding to data shown in Fig. 9c. Count vs. objective function value. Distributions of converged objective function values for Groups 1–5 on a set of test cases that scale with dimension. The behavior of Groups 2 and 3 is very similar, even exhibiting a pattern of expansion-contraction in their distributions with dimension. Group 1 and Hybrid-1,3 drift further with dimension than Groups 2–4, indicating worse compatibility relative to Groups 2–4. Being the constrained version of Fig. 9b, Group 4 consistently converges to the 10^{-3}-10^{-1} range, suggesting that the objective function is "deceptive" for this group



References

Ahrari A, Saadatmand MR, Shariat-Panahi M, Atai AA (2010) On the limitations of classical benchmark functions for evaluating robustness of evolutionary algorithms. Appl Math Comput 215:3222–3229

Angeline PJ (1998) Evolutionary optimization versus particle swarm optimization: philosophy and performance differences. In: Proceedings of the 7th International Conference, EP98, San Diego

Bratley P, Fox BL (1988) Algorithm 659 implementing Sobol's quasirandom sequence generator. ACM Trans Math Softw 14(1): 88–100

Burkardt J (2009) SOBOL - the sobol quasirandom sequence:, 12 December 2009. [Online]. Available: http://people.sc.fsu.edu/~jburkardt/cpp_src/sobol/sobol.html. Accessed 1 Jan 2013

Coello Coello CA (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Methods Appl Mech Eng 191:1245–1287

Culberson JC (1998) On the futility of blind search: an algorithmic view of "no free lunch". Evol Comput 6(2):109–127

Das S, Konar A, Chakraborty UK (2005) Two improved differential evolution schemes for faster global search. In: GECCO '05 Proceedings of the 7th annual conference on Genetic and evolutionary computation, New York

Das S, Abraham A, Konar A (2008) Particle swarm optimization and differential evolution algorithms: technical analysis, applications and hybridization perspectives. In: Advances of Computational Intelligence in Industrial Systems, vol 116, Berlin/Heidelberg, Springer, pp 1–38

Droste S, Jansen T, Wegener I (2002) Optimization with randomized search heuristics—the (A)NFL theorem, realistic scenarios, and difficult functions. Theor Comput Sci 287:131–144

Dulikravich GS, Martin TJ, Dennis BH, Foster NF (1999) Multidisciplinary hybrid constrained GA optimization. Chapter 12 in EUROGEN'99 - Evolutionary Algorithms in Engineering and Computer Science: Recent Advances and Industrial Applications, Jyvaskyla, Finland, 30 May–3 June, pp 231–260

Fan H-Y, Lampinen J, Dulikravich GS (2003) Improvements to mutation donor formulation of differential evolution. In: EUROGEN2003 - International Congress on Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, Barcelona, Spain, September 15–17

Gendreau M, Potvin J-Y (eds) (2010) Handbook of metaheuristics, 2nd edn. Springer, New York

Helwig S, Branke J (2013) Experimental analysis of bound handling techniques in particle swarm optimization. IEEE Trans Evol Comput 17(2):259–271

Hooker JN (1995) Testing heuristics: we have it all wrong. J Heuristics 1: 33–42

Inclan E (2014) The development of a hybrid optimization algorithm for the evaluation and optimization of the asynchronous pulse unit. FIU Electronic Theses and Dissertations, Miami, December 2014

Inclan EJ, Dulikravich GS (2013) Effective modifications to differential evolution optimization algorithm. In: V International Conference on Computational Methods for Coupled Problem in Science and Engineering, Santa Eulalia, Ibiza, Spain, June 17–19

Inclan EJ, Dulikravich GS, Yang X-S (2013) Modern optimization algorithms and particle swarm variations. In: 4th Symposium on Inverse Problems, Design and Optimization-IPDO2013, Albi, France, June 26–28

Kennedy J, Eberhart R (1995) Particle swarm optimization. Neural Networks, 1995. Proceedings., IEEE International Conference on, pp 1942–1948

Kleywegt AJ, Shapiro A (2001) Stochastic optimization. In: Salvendy G (ed) Handbook of industrial engineering, 3rd edn. WIley, New York, pp 2625–2650

Lampinen J, Fan H-Y (2003) A trigonometric mutation operation to differential evolution. J Glob Optim pp 105–129

Malakooti B, Kim H, Sheikh S (2012) Bat intelligence search with application to multi-objective multiprocessor scheduling optimization. Int J Adv Manuf Technol 60(9–12):1071–1086

Oltean M (2004) Searching for a practical evidence of the No Free Lunch theorems. In: Biologically inspired approaches to advanced information technology, Lausanne, Springer, pp 472–483

Pedersen MEH (2010) Tuning & simplifying heuristical optimization. Hvass Laboratories, Southampton

Radcliffe NJ, Surry PD (2005) Fundamental limitations on search algorithms: evolutionary computing in perspective. In: Computer science today, vol 1000, Berlin, Springer, pp 275–291

Rardin RL, Uzsoy R (2001) Experimental evaluation of heuristic optimization: a tutorial. J Heuristics 7:261–304

Rothlauf F (2011) Optimization methods. In: Design of modern heuristics, Berlin, Springer Berlin Heidelberg, pp 45–102

Schittkowski K (1987) More test examples for nonlinear programming codes. Springer, Berlin

Schittkowski K, Hock W (1981) Test examples for nonlinear programming codes - all problems from the Hock-Schittkowski-collection. Springer, Bayreuth

Shilane D, Martikainen J, Dudoit S, Ovaska SJ (2008) A general framework for statistical performance comparison of evolutionary computation algorithms. Inf Sci 178:2870–2879

Sobol IM (1976) Uniformly distributed sequences with an additional uniform property. USSR Comput Math Math Phys 16(5):236–242

Storn R (1996) On the usage of differential evolution for function optimization. In: Proceedings of the 1996 Biennial Conference of the North American Fuzzy Information Processing Society - NAFIPS, Berkeley

Storn R, Price K (1997) Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 341–359

Sun J, Feng B, Xu W (2004) Particle swarm optimization with particles having quantum behavior. In: Congress on evolutionary computation, 2004. CEC2004., Portland

Sun J, Lai CH, Xu W, Chai Z (2007) A novel and more efficient search strategy of quantum-behaved particle swarm optimization. ICANNGA '07 Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms, pp 394–403

van den Bergh F, Engelbrecht AP (2006) A study of particle swarm optimization particle trajectories. Inf Sci 937–971

Vanderplaats GN (2005) Numerical optimization techniques for engineering design, 4th edn. Vanderplaats Research & Development, Inc., Colorado Springs

Vesterstrom J, Thomsen R (2004) A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In: Congress on Evolutionary Computation, 2004. CEC2004, Portland

Weyland D (2010) A rigorous analysis of the harmony search algorithm - how the research community can be misled by a "novel" methodology. Int J Appl Metaheuristic Comput 1–2:50–60

Wolpert DH, MacReady WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82

Yang X-S (2009) Firefly algorithms for multimodal optimization. Stochastic Algorithms: Foundations and Applications, SAGA 2009, vol 5792, pp 169–178

Yang X-S (2010a) Nature-inspired metaheuristic algorithms. Luniver Press, Frome

Yang X-S (2010) Firefly algorithm - file exchange - MATLAB central, 13 December 2010. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/29693-firefly-algorithm. Accessed 26 Sept 2015

Yang X-S (2010) A new metaheuristic bat-inspired algorithm. Nature Inspired Cooperative Strategies for Optimization (NISCO 2010), vol 284, pp 65–74

Yang X-S (2010) Cuckoo search (CS) algorithm - file exchange - MATLAB central, 22 December 2010. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/29809-cuckoo-search–cs–algorithm. Accessed 26 Sept 2015

Yang X-S (2012) Bat algorithm (demo) - file exchange - MATLAB central, 20 July 2012. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/37582-bat-algorithm–demo-. Accessed 26 Sept 2015

Yang X-S, Deb S (2010) Engineering optimisation by cuckoo search. Int J Math Model Numer Optim 1(4):330–343