Title

Competitive Swarm Optimizer with Mutated Agents for Finding Optimal Designs for Nonlinear Regression Models with Multiple Interacting Factors.

Journal

Memetic Computing, 12(3)

Authors

Zhang, Zizhao
Wong, Weng Kee
Tan, Kay

Publication Date

2020-09-01

DOI

10.1007/s12293-020-00305-6

Peer reviewed

HHS Public Access

Author manuscript

Memet Comput. Author manuscript; available in PMC 2021 September 01.

Published in final edited form as:

Memet Comput. 2020 September ; 12(3): 219–233. doi:10.1007/s12293-020-00305-6.

Competitive Swarm Optimizer with Mutated Agents for Finding Optimal Designs for Nonlinear Regression Models with Multiple Interacting Factors

Zizhao Zhang[a], **Weng Kee Wong**[a,*], **Kay Chen Tan**[b]

[a]Department of Biostatistics, University of California at Los Angeles, Los Angeles, California 90095-1772, U.S.A

[b]Department of Computer Science, City University of Hong Kong, Hong Kong

Abstract

This paper proposes a novel enhancement for Competitive Swarm Optimizer (CSO) by mutating loser particles (agents) from the swarm to increase the swarm diversity and improve space exploration capability, namely Competitive Swarm Optimizer with Mutated Agents (CSO-MA). The selection mechanism is carried out so that it does not retard the search if agents are exploring in promising areas. Simulation results show that CSO-MA has a better exploration-exploitation balance than CSO and generally outperforms CSO, which is one of the state-of-the-art metaheuristic algorithms for optimization. We show additionally that it also generally outperforms swarm based types of algorithms and an exemplary and popular non-swarm based algorithm called Cuckoo search, without requiring a lot more CPU time. We apply CSO-MA to find a *c*-optimal approximate design for a high-dimensional optimal design problem when other swarm algorithms were not able to. As applications, we use the CSO-MA to search various optimal designs for a series of high-dimensional statistical models. The proposed CSO-MA algorithm is a general-purpose optimizing tool and can be directly amended to find other types of optimal designs for nonlinear models, including optimal exact designs under a convex or non-convex criterion.

Keywords

c-Optimal Design; *D*-Optimal Design; Large Scale Global Optimization; Optimal Exact Design; Swarm Optimization

1 Introduction

Swarm algorithms are increasingly used in various disciplines to optimize different types of problems. They are easy to implement and often able to find good quality solutions to

*Corresponding author: wkwong@ucla.edu (Weng Kee Wong).

complex optimization problems without many technical assumptions. For example, the objective function does not need to be differentiable or separable. The flexibility of these algorithms enables them to tackle different types of real-world optimization problems in engineering and computer science, and increasingly in other disciplines as well.

Particle Swarm Optimization (PSO), proposed in [13], is one of the most well known swarm algorithms inspired by nature. This simple algorithm is initiated by generating a swarm of particles (candidate solutions) in the user-defined search space. Particles coordinate and move to regions near the perceived optimum iteratively based on each particle's historical pathway and trajectory of the whole swarm. PSO has been applied successfully in many fields, for example, in blind signal separation, power dispatch and model variable selection [17, 23, 24, 59, 61]. The dimensions of these problems range from 5 to 30.

A limitation of PSO is that it can prematurely converge to a local optimum without having adequately explored the search space [1, 15, 36, 38, 68]. An effective algorithm explores the space sufficiently and has good exploitation properties to locate the optimum when it is in the proximity. Frequently, a trade-off between the two competing objectives is required because algorithms that are good at space exploration have limited resources to sufficiently exploit promising areas where the optimum is and algorithms with aggressive exploitation strategy can easily get stuck at a local optimum. [6, 67] showed that the premature convergence issue of PSO is more problematic when there are many variables to optimize in a high-dimensional space.

Many strategies have been proposed to improve PSO performance for tackling complicated optimization problems. They include parameter adaptation [4, 49, 53, 54, 63], hybridization with other optimization methods [20, 35, 47, 50] and swarm topological redesign [26, 27, 57, 71]. Simulations have shown that these amended PSO algorithms, among others, perform better than the original version. One of the most effective enhancements is the Competitive Swarm Optimizer (CSO), proposed by Cheng et al. (2015), to address the premature convergence issues in PSO. CSO adopts a pairwise competition mechanism to update particles at every iteration. Compared to PSO and most of its variants, CSO has a simpler structure and its updating strategy has been shown to more effective. In particular, many simulations using tests on a series of benchmark functions have shown that CSO can find significantly better solutions than PSO and other EAs for different types of problems up to 5000 dimensions [8, 39, 58, 76].

One way memetic algorithms work is by organically integrating local search strategies with evolutionary global search methods to find a global optimum. Memetic computing has been shown capable of solving complicated optimization problems with much less computing resources [7, 28, 45] and is now extensively applied in many cutting-edge research [16, 25, 64, 78]. Our work hybridizes CSO with a local search to bring about a more effective search for finding challenging optimal design problems in statistics.

More specifically, we proposes an enhancement of CSO by adding mutated agents (MA) to CSO and refers the algorithm as CSO-MA. Mutated agents are randomly selected loser particles from pairwise comparisons at each iteration in CSO-MA and they typically make

the swarm more diversified and result in a more effective local space search that inspired by memetic algorithms. An advantage of having mutated agents is that they do not disturb the swarm's movement if they cannot find promising areas. Using several test functions, our simulation results show that CSO-MA tends to outperform CSO and this would in turn imply that CSO-MA should also outperform many state-of-the-art EAs. As an application, we apply CSO-MA to find *c*-optimal designs for estimating selected coefficients in high-dimensional nonlinear regression models for count outcomes. The design methodology is general and can be applied to other regression models and design criteria.

Section 2 reviews the Particle Swarm Optimization and Competitive Swarm Optimizer. Section 3 describes how mutated agents we propose can potentially improve the algorithm's performance and introduces the new algorithm Competitive Swarm Optimizer with Mutated Agents. In section 4, we conduct a simulation and compare performance of CSO-MA with several of its competitors using benchmark functions up to 1000 dimensions. Our results suggest CSO-MA generally outperforms other popular algorithms by a wide margin and able to produce significantly better solutions. In Section 5, we apply CSO-MA to find various types of optimal designs for nonlinear statistical models, including those that require about 100 or more variables to optimize. Not all our numerical results are reported for space consideration and they all indicate that CSO-MA was effective in finding optimal designs not yet known in the literature. Section 6 contains a summary.

2 Swarm Optimization

Throughout, we assume the objective function is $f(x)$ and we want to solve the minimization problem

$$\min_{\mathbf{x} \in \mathbf{\Omega}} f(\mathbf{x}),$$

where $\mathbf{\Omega} \subset \mathbb{R}^D$ is a user-selected compact set and D is the number of variables to optimize in the problem.

Most swarm algorithms initialize by generating at random a user-selected number of, say n, particles as candidate solutions in $\mathbf{\Omega}$. These particles or points interact with one another and update their positions by some rules. For instance, the swarm could move towards current best particle positions or historically best areas by adding some random search at each particle. Our interest is in swarm-based algorithms.

2.1 Particle Swarm Optimization and Its Variants

For classic PSO, after the user randomly generated a swarm of selected size, the algorithm updates every particle x_i by referring its historical movement information and temporary global best solution. Every particle is assigned with a velocity vector $\mathbf{v}$, which is also randomly generated when initialization. Considering particle i at iteration t, it will change its velocity $\mathbf{v}_i^{t+1}$ and move to a new position $\mathbf{x}_i^{t+1}$ by

$$\mathbf{v}_i^{t+1} = \omega \mathbf{v}_i^t + \beta_1 \mathbf{R}_1 \otimes \left(pbest_i^t - \mathbf{x}_i^t\right) + \beta_2 \mathbf{R}_2 \otimes \left(gbest^t - \mathbf{x}_i^t\right)$$
$$\text{and } \mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1},$$

where $\mathbf{R}_1$, $\mathbf{R}_2$ are random vectors whose elements are independently drawn from the uniform distribution $U(0, 1)$; operation $\otimes$ means element-wise multiplication; ω, β_1, β_2 are parameters reflecting influence of different components on changing particle's next step. The best position that each particle has visited till iteration t is the personal best $pbest_i^t$, $i = 1, \cdots, n$ and the best position that the swarm of particles has ever reached till iteration t is the global best $gbest^t$. The term best refers to positions where the value of the objective function or its fitness value is smallest. These two centers influence every particle's movement. By adding stochastic components $\mathbf{R}_1$ and $\mathbf{R}_2$ into the algorithm, these particles have a chance to explore unseen areas where they might be able to capture better solutions.

How to tune parameters in a metaheuristic algorithm is a perennial issue. There are recommendations on how to tune the parameters in PSO for a more effective search; some examples are [5, 12, 53]. Some proposed that these parameters be modeled as a function of the iteration number t or the recent best objective values or as topological distances among the particles [14, 40]. Others argued for having a set of constant parameters, such as $\omega \in [0.8, 1.2]$, $\beta_1 = \beta_2 = 2$ and showed that they frequently worked well [13].

It is known that PSO is prone to premature convergence [21, 31, 79]. Typically, PSO does the exploration in a few iterations and then proceeds to exploit [42], possibly resulting in a decrease of the quality of the solution. This premature convergence phenomenon is likely due to choice of tuning parameters or its strong connection with the two centers *pbest* and *gbest*, which may be exerting undue influence and not changing frequently enough during iterations, see, for example, [8, 43, 70].

2.2 Competitive Swarm Optimizer

[8] proposed CSO to tackle the premature convergence issue by recasting the updating formulas. Like PSO, CSO first generates a swarm of n particles at positions $\mathrm{x}_1, \cdots, \mathrm{x}_n$ with random velocities $\mathbf{v}_1, \cdots, \mathbf{v}_n$ in Ω. In each iteration, we randomly divide them into $\left\lfloor \frac{n}{2} \right\rfloor$ pairs and compare their objective function values. We identify $\mathbf{x}_i^t$ as winner and $\mathbf{x}_j^t$ as loser if these two are competed at iteration t and $f(\mathbf{x}_i^t) < f(\mathbf{x}_j^t)$. Winner retains status quo and the loser learns from the winner. The two defining equations for CSO are

$$\mathbf{v}_j^{t+1} = \mathbf{R}_1 \otimes \mathbf{v}_j^t + \mathbf{R}_2 \otimes (\mathbf{x}_i^t - \mathbf{x}_j^t) + \phi \mathbf{R}_3 \otimes (\bar{\mathbf{x}}^t - \mathbf{x}_j^t)$$
$$\text{and } \mathbf{x}_j^{t+1} = \mathbf{x}_j^t + \mathbf{v}_j^{t+1},$$

where $\mathbf{R}_1$, $\mathbf{R}_2$, $\mathbf{R}_3$ are all random vectors whose elements are drawn from $U(0, 1)$; operation $\otimes$ also represents element-wise multiplication; vector $\bar{\mathbf{x}}^t$ is simply the swarm center at iteration t; social factor ϕ controls the influence of the neighboring particles to the loser and

a large value is helpful for enhancing swarm diversity (but possibly impacts convergence rate). This process iterates until some stopping criteria are met.

There are 3 tuning parameters ω, β_1, β_2 in PSO and only one parameter ϕ in CSO, suggesting that it is simpler to tune CSO. Further, the transitory data PSO needs to keep track of are stored in a $n \times$ D matrix $\mathbf{x}$, a $n \times$ D matrix $\mathbf{v}$ and a $n \times$ D matrix *pbest* whereas CSO needs two of these implying that a smaller memory space is required to run CSO.

Simulation results have shown that CSO either outperforms or is competitive with many state-of-the-art swarm algorithms, such as PSO with Constriction Factor (PSO-CO), Gaussian Bare Bones PSO (GBBPSO), or Quantum PSO (QPSO). This conclusion was arrived at after comparing CSO performance with state-of-the-art algorithms using a variety of benchmark functions with dimensions up to 5000 [8, 39, 58, 75, 76, 77]. They showed that CSO was frequently not only the winner but also required significantly less runtime.

CSO is relatively new but has many exciting applications. For example, [18] applied CSO to select variables for high-dimensional classification models; [69] used CSO to study a power system economic dispatch, which is typically a complex nonlinear multivariable strongly coupled optimization problem with equality and inequality constraints, and [29] employed CSO to find the optimal installation of multiple distributed generation units in radial distribution network.

3 Competitive Swarm Optimizer with Mutated Agents

Genetic algorithm (GA) is another important branch of evolutionary algorithms. Almost all versions of GA operate based on biological-inspired behaviors such as mutation, crossover and selection to evolve better solutions [65]. There have been a lot of inspiring GA that performed surprisingly well in various fields [11, 19, 32, 41].

We incorporate ideas from the genetic algorithm to enrich CSO and call the enhanced version of CSO as Competitive Swarm Optimizer with mutated agents or, in short, CSO-MA. After pairing up the swarm in groups of two at each iteration, we randomly choose a loser particle p as an agent, randomly pick a variable indexed as q and then randomly change the value of $\mathbf{x}_{pq}$ to either $\mathbf{xmax}_q$ or $\mathbf{xmin}_q$, where $\mathbf{xmax}_q$ and $\mathbf{xmin}_q$ represent, respectively, the upper bound and lower bound of the q-th variable, respectively. This change is similar to the "mutation" step in GA. A conservative mutation strategy is to randomly reassign each loser particle to a random position on the boundary. If the current optimal value is already close to the global optimum, this change will not hurt since we implement this experiment on a loser particle, which is not already leading the movement of the whole swarm; otherwise, this chosen agent restarts a journey from the boundary and has a chance to escape from a local optimum.

We apply CSO-MA to minimize functions that are not necessarily separable or convex and they may have multiple local optima. The computational complexity of CSO is $\mathcal{O}(nD)$, where n is the swarm size and D is the dimension of the problem. Since our modification only adds one coordinate mutation operation to each particle, its computational complexity is the same as that of CSO, see section 4. Algorithm 1 displays a pseudo code of CSO-MA.

Algorithm 1

The Pseudo Code for CSO-MA

A swarm of n particles.

x ← Randomly assign initial positions in space to particles.

v ← Randomly assign initial velocities to particles.

while not stopping criteria **do**

Randomly divide the swarm into $\left\lfloor \frac{n}{2} \right\rfloor$ pairs.

for each pair **do**

Compare their objective function values and set the one with smaller value as winner and the other as loser.

Update loser particles.

if $\mathbf{x}_{loser}$ out of searching space **then**

$\mathbf{x}_{loser}$ ← position at boundary.

end if

Randomly choose a loser $\mathbf{x}_p$ and a coordinate index q.

Randomly change q-th variable of $\mathbf{x}_p$ to either $\mathbf{xmax}_q$ or $\mathbf{xmin}_q$, where $\mathbf{xmax}_q$, $\mathbf{xmin}_q$ represent upper bound and lower bound of q-th variable.

end for

end while

3.1 Parameter Tuning

Tuning parameters is a perennial and critical issue for meta-heuristic algorithms because a poor choice for them can result in very poor performance. Most of these algorithms have at least two or three parameters, which makes a systematic understanding on how they interact to impact the algorithm's performance tricky. [8] provided a thorough experiment choosing parameters for CSO to solve problems of different scales whose results clearly exhibited parameters' various influence.

We experimented with tuning values for CSO-MA's parameters ϕ and swarm size n and found that the original default values for the tuning parameters for CSO for ϕ and n can be reliably transferred to CSO-MA; they are provided later on.In the next section, where we use several benchmark functions to ascertain performance of CSO-MA, we vary the size of n to determine whether the performance of the algorithms depend on the dimension of the optimization problem. Simulations show that, under such a setup, a change of ±0.05 in the value ϕ, as long as it is non-negative, does not affect CSO-MA's performance. Our experience is that our tuning parameters seem effective for the problems we tried and note that when a parallel-computing program or machine is available to run the algorithm, a large value of n should be used. In the next section, we also discuss whether it is helpful to have the number of agents that mutate at each iteration as an additional parameter in CSO-MA and whether it has an impact on the solution quality.

4 Simulation

We now use simulation to compare performance of CSO-MA with a few state-of-the-art swarm based competitors using benchmark functions commonly used in the literature [62, 73, 74]. We also include a non-swarm based algorithm, Cuckoo search, in our comparison.

We use eight benchmark functions with different mathematical properties and consider cases when they have dimensions $D = 100$, 500 and 1000. Functions f_1, f_2, f_3 are the Schwefel N.2.21 function (non-differentiable, non-separable), the Rosenbrock function (non-separable), and the Sphere function. They are defined on the space $[-100, 100]^D$, where D refers to the dimension of the function. Function f_4 is the Rastrigin function (multimodal) and it is defined on $[-5.12, 5.12]^D$. Function f_5 is the Schwefel function (non-convex, multimodal) and it is defined on $[-500, 500]^D$. Function f_6 is the Gramacy & Lee function (non-convex, multimodal) and it is defined on $[0.5, 2.5]^D$. Function f_7 is the Griewank function (non-convex, non-separable, multimodal), defined on $[-600, 600]^D$. The last function f_8 is the Ackley function (non-convex, multimodal), defined on $[-32, 32]^D$. Except for the Gramacy & Lee function which has a global minimum of $-0.869D$, all other functions have a global minimum of zero.

In addition to CSO, we compare CSO-MA algorithm with the following algorithms: (i) the Modified CSO (MCSO) algorithm, which replaces the CSO's pairwise competition strategy by a triplet competition and is recognized as an improved CSO, (ii) the Cooperatively Coevolving PSO 2 (CCPSO2) algorithm that uses a Cauchy and a Gaussian distribution for sampling next-generation particles, respectively, at *pbest* and *gbest* [33], (iii) the Multilevel Cooperative Coevolution (MLCC) designed to conduct a self-adaptive neighborhood search for promising particles [73], (iv) the Separable Covariance Matrix Adaptation Evolution Strategy (SEP-CMA-ES), which generates new candidate solutions by sampling around old particles and the sampling covariance matrix is constructed by incorporating information from the current solution [52], (v) the Efficient Population Utilization Strategy for PSO (EPUS-PSO), which adjusts the population size according to the search results and (vi) the Dynamic Multi-Swarm (DMS-PSO) that adopts a dynamically changing neighborhood structure for each particle [22, 34]. The last algorithm that we include for comparison is the Cuckoo search algorithm, which uses Levy flights and random walk to update new solutions [72].

4.1 Simulation Setup

We followed recommendations for these choices from [8], which were based on a series of tests. Specifically, when optimizing $100D$ problems, we set $n = 100$, $\phi = 0$. For higher-dimensional optimization problems, they recommended the choice for these tuning parameters depend whether the objective function is separable or not. Specifically, for $500D$ problems, they suggested $n = 250$, $\phi = 0.1$ for separable functions and $n = 250$ and $\phi = 0.05$ for non-separable functions; for $1000D$ problems, they suggested $n = 500$ and $\phi = 0.15$ for separable functions and $n = 500$ and $\phi = 0.10$ for non-separable functions. Since CSO-MA inherits the same particle updating strategy from CSO, we follow the tuning formula for CSO and show that under the same parameter setup, the optimization performance of CSO-MA is improved. For MCSO, the tuning values of the parameters come from Table 3 of [39].

For other algorithms, similar simulations have been carried out in [8, 39] and we adopt the same parameter tuning strategy there (we refer to the parameter setup for Cuckoo search algorithm suggested in [72]). For all algorithms, we stop running their runtime if it requires more than $5000D$ function evaluations, which was the guideline proposed in [60].

We implemented our algorithms using C++ on Xcode 9.0.1 and compiled them via GCC 7.2.0. The codes are available upon requests from the first author. All experiments were run on Hoffman2 shared cluster housed at the University of California at Los Angeles. For each function of a specific dimension, we ran each algorithm 25 independent times and recorded all outcomes. The hardware employed is a 2.2 GHz Intel Xeon E5–2650v4 CPU and 8 GB memory.

4.2 Simulation Results

Tables 1, 2 and 3 present means and standard deviations of the values of the objective functions from our simulation. The bold numbers in each column represent the best performing algorithm among the nine algoirthms for minimizing each of the eight benchmark functions. Wilcoxon rank test, which is a nonparametric method to test whether there is a significant difference between two sets of measurements, was then used to compare performance of CSO-MA with other algorithms. At the 0.05 significance level, all results from all tests were significant in all three tables, suggesting the algorithm with the bold value finds a smaller objective function value then each of the other algorithms. The last column in each table with the heading "**w**/**t**/**l**" displays the number of times CSO-MA wins, ties and loses to the corresponding algorithm.

There is the celebrated "No Free Lunch" rule that says no algorithm can outperform all other algorithms in all situations. An interesting interpretation of this theorem is recently available in [37]. Table 4 ranks the ability of CSO-MA to minimize these benchmark functions relative to other algorithms. For instance, when minimizing functions with $D = 100$, CSO-MA has 3 times defeating over all other eight algorithms, 3 times over other seven algorithms, 2 time over other six algorithms, etc., and we record such result as (3, 3, 2, …, 0) corresponding to the header "Rank", "Rank2", etc. A smaller rank indicates that the algorithm has a better minimization performance. We select algorithms that are best for optimizing these functions but also algorithms that are consistently highly ranked. The last column in the table displays the "Average Rank" and so indicates whether the algorithm can stably solve different optimization tasks.

From the summary tables, we observe that CSO-MA outperforms the other algorithms for minimizing function f_1, f_5 and f_6 regardless of the dimension of the problem. For function f_2 and f_4, CSO-MA provides competitive results among all the algorithms. Although CSO-MA did relatively poor minimizing functions f_3, f_7 and f_8, its results are acceptable because these solutions are within 10^{-3} units from the true optimum. One possible explanation is that for these functions, CSO-MA sacrifices its ability to exploit at the expense of having the mutated agents do more space exploration. A summary observation is that CSO-MA is the most consistent optimizer among these algorithms since on average, it has the best performances in terms of minimizing the objective functions regardless of the dimension of

the problem, which is also confirmed by its average rank for minimizing the functions by the various algorithms.

Other algorithms perform differently for each benchmark functions, which again confirms the "No Free Lunch" rule. CSO, MLCC, CCPSO2 can consistently provide intermediate results while MCSO's outputs are not stable. EPUS-PSO, DMS-PSO and Cuckoo have a relatively poor ability optimizing these commonly-used functions.

Throughout, we set 10^{-3} as the tolerance level and so solutions that are within $\pm 10^{-3}$ from the true "optimal" value are deemed optimal. Under this rounding setup, CSO-MA's overall performance relative to the other seven algorithms for optimizing the 24 benchmark functions becomes more impressive with 131 wins, 37 ties, 0 loses compared to the earlier more stringent criterion with 130 wins, 4 ties, 34 loses.

Additionally, we compare performance of CSO-MA with a recent state-of-the-art hybridized version of cuckoo search and particle swarm optimization proposed by [9]. When this hybrid algorithm is applied to to minimize $1000D$ f_1 to f_8, we obtain corresponding mean results of 5.47E+01, 3.26E+03, 2.50E-08, 2.33E+02, 2.15E+05, −3.79E+02, 7.54E-04 and 2.66E-01. This shows that CSO-MA still outperofrms this hybrid PSO algorithm, which also has a more complicated structure and more tuning parameters, making it is less friendly to use in practice.

4.3 More Mutated Agents?

The change we make in CSO-MA algorithm is to randomly select an agent from the loser list at every iteration and reassign it at random to a point on the boundary. Our results have shown that this is an effective strategy. A natural question to ask is whether having more mutated agents at every iteration will further enhance performance of CSO-MA.

To address this question, we keep the benchmark test configurations fixed and compare CSO-MA results when $m = 2, \ldots, 10$ versus the case when $m = 1$. The histogram in Figure 1 shows the number of times significantly improved results are obtained via the Wilcoxon test for the 24 test cases when a larger value of m is used versus $m = 1$. The top histogram (a) shows different values of m and the bottom histogram (b) shows corresponding results when m is expressed as a percentage of n. We observe from the two histograms results for the case when $m = 1$ generally outperforms other cases and further, a larger value of m tends to decrease the algorithm's effectiveness. One explanation is that when m increases, there is less balance between exploration and exploitation. In particular, a larger value of m encourages the swarm to explore a larger area since more particles are assigned to random positions on the boundary and so more likely to find a better solution. This follows from the fact that for some optimization problems, like finding D-optimal designs to be discussed later, support points tend to be at the boundary of the search space. However, with a larger value of m, more particles mutate and this may make the swarm more difficult to exploit the current promising area.

Our analysis of the differences between GA and CSO-MA is that the latter has in-built features that likely explain its out-performance when compared with its other competitors.

For example, GA requires that a part of offspring chromosomes to mutate and for CSO-MA, there is only one mutation per iteration. This means that CSO-MA requires fewer number of computational operations and so saves time. Further, if the size of the cohort/swarm or the number of chromosomes is fixed, GA replaces existing "bad" chromosomes with newly-mutated offspring chromosomes that may not identify more promising solutions, and also loses all the information provided by the previous "bad" chromosomes. In contrast, the amount of information lost by mutating particles, i.e. one particle and one coordinate per iteration, in CSO-MA is relatively trivial.The upshot is that in CSO-MA, loser particles do not lead the swarm movement, delay the movement speed, able to inform others of unpromising areas and allow CSO-MA to explore new areas more effectively.

4.4 Swarm Diversity

Earlier work suggests that swarm diversity may affect the quality of a swarmn based algorithm. Following [44], we use the swarm diameter as an index to measure the diversity of a swarm. This index is defined by

$$Dia = \max_{(i \neq j)} \sqrt{\sum_{r=1}^{D} (\mathbf{x}_{ir} - \mathbf{x}_{jr})^2},$$

where $\mathbf{x}_{ir}$ and $\mathbf{x}_{jr}$ are, respectively, the r^{th} component or dimension of the i^{th} and j^{th} particles.

When the swarm diversity index drops to zero, this implies that the search has ended and all particles have converged to a single point. The swarm cannot revive and find a better solution. On the contrary, as long as the swarm diversity is above a specific level, the swarm has a chance to explore other areas of the space, which equals to mean that the terms $\mathbf{x}_i^t - \mathbf{x}_j^t$ and $\bar{\mathbf{x}}^t - \mathbf{x}_j^t$ could be nonzero.

Table 5 shows the swarm diameter *Dia* of CSO and CSO-MA at the start and at the end of the search, with at a mid-way point as they iterate to minimize each of the benchmark functions f_4 to f_8. These functions were chosen because they have multiple local minima and their global minima are not located at or near the boundary of the search space. This means that merely adopting the common practice of searching for a global optimum at or near the boundary is not helpful, i.e., these functions are hard to optimize. Columns 4 and 6 display the averaged values of these functions found at the 1^{st}, $2500D^{th}$ and $5000D^{th}$ function evaluation numbers, where D is the dimension of the function, and D = 100, 500 and 1000. We note that CSO-MA has 10 wins, 3 ties and 2 loses compared with performance of CSO. It appears that CSO-MA's success in finding better solutions than CSO is due to its having a more diverse swarm during the search process. CSO also seems to run out of energy midway during its search whereas CSO-MA always keeps a dynamic and diversified swarm and enables it to jump out of local optima. An asterisk in the last column indicates that the at the end of the $5000D$ iterations, the optimized value by CSO-MA is significantly better than that by CSO at the 0.05 significant level.

We also measure the *Dia*'s for all other algorithms during the test as we implement on CSO and CSO-MA. In short, all other algorithms we compare CSO-MA with, they cannot achieve a similar swarm diversity level as CSO-MA has in the whole process of the test.

CSO-MA's enhanced performance is not necessarily limited to optimizing multimodal functions. We observe that results from CSO-MA for optimizing unimodal functions f_1 to f_3 are comparable to those from CSO. For space consideration, we do not display the *Dia* patterns for optimizing the other functions, but note that they share a very similar pattern. In real applications, the CPU time required to find the optimum is unknown and so it is common to employ longer runs. The implication is that under such a circumstance, CSO-MA has the potential of finding a better solution given a longer runtime compared to CSO.

[10] proposed a mutating-to-the-boundary strategy in an improved PSO algorithm called Elastic Boundary for Particle Swarm Optimization (EBPSO). At each iteration of EBPSO, it defines a elastic region given current global value. Then each particle is examined by a criterion to determine whether it needs to fly to a boundary area of the elastic region according to an updating function for space exploration. Compared to their algorithm design, CSO-MA has a dominant advantage that at each iteration, only one particle needs to be mutated (calculation complexity of the mutation step $\mathcal{O}(1)$), while for EBPSO, all particles have to be examined and some have to be mutated (calculation complexity of the mutation step at least $\mathcal{O}(nD)$). In the table 2 of [10], the mean results EBPSO obtained for minimizing four benchmark functions are 5.05E-31, 5.26E-03, 3.98E-01 and 2.85E+00. Under the same testing setup, CSO-MA's mean results are 7.03E-55, 4.92E-07, 2.17E-06 and 1.90E-03. These results show that CSO-MA's mutation strategy is more effective than EBPSO.

4.5 Algorithm Speed

CSO-MA only adds a mutation operation on one particle per iteration and so the algorithmic complexity does not change compared to the original CSO. Table 6 records average running time for CSO and CSO-MA to minimize each function. The table shows that there is no significant efficiency gap between them because, for the same function, both algorithms require very similar CPU time.

The next section demonstrates that CSO-MA can find hard to find optimal designs for a nonlinear regression model with multiple interacting factors. The purpose of the application is to show CSO-MA can find the optimal design and other commonly used metaheuristic algorithms cannot. We focus on an example but our experience is that CSO-MA can also find other types of optimal designs for other nonlinear models as well. To this end, we first provide a brief background on the fundamentals of constructing an optimal design.

5 Application: Locally *c*-Optimal Designs for High Dimensional Statistical Models

5.1 Background

Optimal experiment design is an increasingly important sub-field in statistics in part due to the rising experimental cost and the need to make statistical inference accurately and reliably at minimum cost [66]. There are several design monographs of varying levels of mathematical complexity and they include [3, 48]. [2] contains real optimal design problems and their solutions, including one that involves an engineering problem of optimally allocating wells in the Los Angeles basin.

We assume the statistical model has the form

$$\mathbf{E}y = f(\mathbf{x}, \theta),\ \mathbf{x} \in \boldsymbol{\Omega}$$

where y is the response variable, $f(\mathbf{x}, \boldsymbol{\theta})$ is the mean function with input vector $\mathbf{x}$ and $\theta \in \mathbb{R}^p$ is the p-dimensional vector of model parameters.

Design problems arise early in the study and the concerns are how to select levels or combination levels of the explanatory variables to observe the response y in some optimal way. If there is a pre-determined N number of observations and a given objective for the study, common design questions are the optimal number of design points k, their optimal locations $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k$ which are chosen from a user-selected design space $\boldsymbol{\Omega}$ and the optimal number of replications n_i at x_i, $i = 1, \ldots, k$ subject to $n_1 + \ldots + n_k = N$. These are optimal exact designs which are very difficult to find and study theoretically because they usually require some number theory.

An alternative is to find optimal approximate designs where the optimal number of replicates n_i at x_i in the exact designs are replaced by their optimal proportions w_i, $i = 1, \ldots, k$ and subject to $w_i + \ldots + w_k = 1$. Doing so turns the problem into a convex optimization problem for which there is a general framework to find and verify optimality of a design. We denote such an approximate design by $\boldsymbol{\eta} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k; w_1, w_2, \cdots, wk)$.

The worth of a design is usually measured by its Fisher information matrix $\mathbf{I}(\boldsymbol{\eta}, \boldsymbol{\theta})$, which is the negative of the expectation of the second derivatives of the total log-likelihood function with respect to the model parameters. For nonlinear models, the information matrix depends on the unknown parameters which we want to estimate. One way to overcome this problem is to assume nominal parameter values are available, either from similar studies or an expert's opinion. Optimal designs are then found by replacing the unknown parameters in the information matrix by their nominal values so that direct optimization becomes possible. The resulting designs are locally optimal and are implemented to generate data from which parameters are re-estimated. The procedure repeats with the hope that the estimates will stabilize after a couple of iterations.

Many design criteria are formulated as a scalar concave function of the information matrix. For example, a locally D-optimal design maximizes $\log\{\det(\mathbf{I}(\boldsymbol{\eta}, \boldsymbol{\theta}))\}$ for a given nominal

value $\boldsymbol{\theta}$. Another design criterion is c—optimality, which is used to estimate a function of the model parameters as accurately as possible. For a given nominal $\boldsymbol{\theta}$ value, a locally c—optimal design minimizes the asymptotic variance of the estimated quantity of interest; equivalently, it maximizes $-\mathbf{c}^T\mathbf{I}^{-1}(\boldsymbol{\eta}, \boldsymbol{\theta})\mathbf{c}$, where $\mathbf{c}$ is an user-specified column vector. In either case, the optimization is over all approximate designs on $\boldsymbol{\Omega}$.

A benefit of working with approximate designs and concave objective functions is that there are equivalence theorems for confirming optimality of an approximate design among all designs. For example, given a nominal value for the $p \times 1$ vector $\boldsymbol{\theta}$, arguments based on directional derivatives show that an approximate design $\boldsymbol{\eta}$ is locally D-optimal if and only if for $\mathbf{x} \in \boldsymbol{\Omega}$,

$$\mathbf{S_D}(\boldsymbol{\eta}, \mathbf{x}) = \frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^T \mathbf{I}^{-1}(\boldsymbol{\eta}, \boldsymbol{\theta}) \frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} - p \leq 0,$$

with equality at the support points of $\boldsymbol{\eta}$. Likewise, $\boldsymbol{\eta}$ is locally c-optimal if and only if for $\mathbf{x} \in \boldsymbol{\Omega}$,

$$\mathbf{S_c}(\boldsymbol{\eta}, \mathbf{x}) = [\frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^T \mathbf{I}^{-1}(\boldsymbol{\eta}, \boldsymbol{\theta})\mathbf{c}]^2 - \mathbf{c}^T\mathbf{I}^{-1}(\boldsymbol{\eta}, \boldsymbol{\theta})\mathbf{c} \leq 0,$$

with equality at the support points of $\boldsymbol{\eta}$. If an approximate design is not optimal, its proximity to the optimum can be measured via an efficiency lower bound [46]. For example, for D-optimality, a direct deduction from the equivalence theorem shows that if the maximum value of the sensitivity function $\mathbf{S_D}(\boldsymbol{\eta}, \mathbf{x})$ over the design space is a positive number a, the D-efficiency of the approximate design $\boldsymbol{\eta}$ is at least $e^{-a/p}$.

In practice, we evaluate the worth of an approximate design by finding the maximum value a of its sensitivity function over the design space. This is another sub-optimization task and typically easier to maximize compared to finding an optimal design because a lot fewer number of variables are involved. For instance, if the model has three additive factors, optimizing the sensitivity function is a three-dimensional optimization problem; in contrast, determining the optimal design is a $(4k - 1)$-dimensional optimization problem, where k is at least 4 and equal the number of support points of the optimal design.

To check whether the CSO-MA generated design is optimal, we examine whether the sensitivity function is bounded above by 0 and attains 0 at all design points. If the model has only one or two factors, a plot of the sensitivity function across the design space may suffice. For example, Figure 2 displays the sensitivity function of a CSO-generated design for estimating the slope in a negative binomial model with logarithm dose as the only factor and it confirms its optimality. However, when there are three or more factors, the plot is harder to appreciate visually.

We next tackle the task of finding c-optimal approximate designs for a negative binomial models with multiple interacting explanatory factors. This is a challenging design problem as it involves many variables to optimize. In the statistical literature, we were only able to

locate only one design paper that finds locally D-optimal design for a two-parameter negative binomial model [51]. In what is to follow, we apply CSO-MA to search for an optimal design using Matlab 2018a.

5.2 Negative Binomial Regression Model

Negative binomial regression models extend the commonly used Poisson regression models by accommodating for over-dispersed or under-dispersed data. They model the mean of a count outcome y and its relationship with a vector of independent variables $\mathbf{x}$ as follows:

$$\mathbf{E}(y) = e^{\mathbf{x}^T\theta} \quad \text{and} \quad \text{Var}(y) = e^{\mathbf{x}^T\theta}(1 + ae^{\mathbf{x}^T\theta}).$$

Here a is the dispersion parameter; if $a > 0$, the variance exceeds the mean and the data is over-dispersed and if $a < 0$, the data is under-dispersed.

There is little work on constructing optimal designs for the negative binomial model, especially where the model has several interacting factors. The most recent work is from [51], where they derived locally c-optimal approximate designs for two-factor negative binomial models given a series of specific conditions. As an illustrative example, suppose we want to find a locally c-optimal design for estimating the second parameter in a negative binomial model with the logarithm dose of a drug as the only factor. We assume $a = 3$, the log dose of the drug is between $[-3, 5]$ and the nominal values are $\boldsymbol{\theta} = (0.5, 1.7)^T$. We set $\mathbf{c} = (0, 1)^T$, applied CSO-MA. It converged in 0.05s to the design supported at points −0.637 and 5.000 with weights 0.560 and 0.440 respectively. Figure 2 displays the sensitivity function of this design and confirms its optimality. The criterion value of the c-optimal design is 0.483.

5.3 Locally Optimal Approximate Designs for a High-Dimensional Negative Binomial Model

The model of interest is a negative binomial model for a five-factor negative binomial model with all pairwise interactions and the outcome is a count variable. From the design perspective, this is a high-dimensional regression model with 16 parameters and the design problem has a total of 96 variables to optimize if the optimal design is supported at 16 points; otherwise, the number of variables in the optimization problem can increase substantially. We scale each factor values to between −1 and 1 and so the design space is $[-1, 1]^5$.

Table 7 displays four sets of randomly generated nominal values for the model parameters and each is drawn from $U(-2, 2)$. We implemented CSO, MCSO, PSO and DE for comparison purposes and initiated them using 200 candidate solutions. Parameter tuning strategies for these optimizers followed the suggestions in [8, 39, 53, 56] and CSO-MA shared the same parameter setup with CSO. The problem we need to solve has dimension $(5 + 1)k$ where k is theoretically larger than 16. We started the search with $k = 20$ and results show that the optimal designs can be found under this setup. Hence, our optimization problem has 120 variables to optimize. For CSO, we set parameter $\phi = 0.05$; $\phi_1 = 0.05$, $\phi_2 = 0.05$ for MCSO; $w = 0.8$, $\beta_1 = \beta_2 = 2$ for PSO and $CR = 0.3$, $F = 0.75$ for DE.

To fix ideas, suppose $a = 0.2$ and we are interested in estimating θ_2. We set $\mathbf{c} = (0, 0, 1, 0, \cdots, 0)^T$. To find an optimal design for estimating this coefficient, we ran each of the four algorithms 10 times and compare their values of the c-optimality criterion. The stopping criterion was still the "5000D" function evaluation rule described previously. Using CSO-MA's results as a reference, we calculated relative efficiencies of designs found by the other four algorithms. Table 8 summarizes the results and suggests that CSO-MA consistently finds designs with much higher efficiencies than other algorithms for the negative binomial models.

5.4 A Locally *D*-Optimal Exact Design for Logistic Model with a Single Factor

CSO-MA is a flexible algorithm and can also find optimal exact designs for nonlinear models. We recall there is no general theory to confirm optimality of an exact design and so we show an example where the CSO-MA generated design outperforms an optimal exact design found by a popular algorithm in the design literature.

The Fedorov exchange algorithm is a common method for finding optimal approximate designs in the statistical community; see Fedorov (1972). It requires that the objective function be differentiable and it finds the optimal design by adding one or more points at each iteration to the current design to form a new design for the next iteration. The common problem is that clusters of points are formed around the support points and they need to be collapsed into single points periodically. The algorithm can be mathematically proven to converge to the optimal design but it is frequently slow and becomes ineffective when the model is nonlinear and has several interacting factors.

Some have attempted to tweak the algorithm to find optimal exact designs. For example, [30] modified the algorithm to find a six-point locally D-optimal exact design on the design space $[-1, 1]^2$ for a logistic model containing an intercept, two factors, two second-order terms and one interaction term using the following nominal values for the parameters: $\theta_0 = -1$, $\theta_1 = 2$, $\theta_2 = 0.5$, $\theta_3 = 2$, $\theta_4 = 0.1$, $\theta_5 = 0.01$. They reported that the six-point D—optimal exact design $\boldsymbol{\eta}$ is supported at (−1.00, 1.00), (1.00, −1.00), (−1.00, −0.70), (0.06, 0.07), (1.00, −0.03) and (0.14, 1.00).

We applied CSO-MA and the generated 6-point D-optimal exact design $\boldsymbol{\eta}^*$ is supported at (−1.00, 1.00), (−1.00, −1.00), (−0.06, −1.00), (0.37, 1.00), (0.61, 1.00) and (0.71, 0.00). A direct calculation shows that the design $\boldsymbol{\eta}$ has a D-efficiency of only 72% relative to $\boldsymbol{\eta}^*$ and further, CSO-MA completed the search in less than 1 second. This shows that one should be careful to claim optimality of optimal exact designs because (i) they can be very difficult and time-consuming to determine and (ii) the claimed optimal design could be wrong as is in the case just mentioned.

6 Summary

Earlier results from the literature have shown that CSO (i) frequently experiences fewer premature convergence issues and avoids a common problem with PSO when the optimization problem is high-dimensional, and (ii) tends to outperform other state-of-the swarm algorithms. We incorporate mutated agents in the CSO algorithm and show that

CSO-MA enhances its swarm diversity in the search for the global optimum. This modification further improves CSO's exploration-exploitation balance. Numerical results showed that this enhancement increases CSO-MA's ability to tackle high-dimensional and complicated optimization problems successfully without requiring a lot of extra runtime.

Our applications of CSO-MA to find c-optimal approximate designs for the high dimensional negative binomial models and the D-optimal exact designs for the logistic models demonstrate the effectiveness of CSO-MA to solve difficult optimal design problems. Our experience is that CSO-MA can be used to find various types of optimal designs for other nonlinear regression models with several interacting factors. For space consideration, we omit details.

Metaheuristics and memetic computing are continuously developed to solve increasingly difficult optimization problems. For example, a most recently developed improved swarm-based algorithm called variable-size cooperative coevolutionary particle swarm optimization was proposed for feature selection on high-dimensional data [55]. Many disciplines have benefited from metaheuristics and memetic computing, but the extent varies considerably from field to field. For disciplines where they are under-used, it is likely because the disciplines are less informed about the power, flexibility and utility of metaheuristics and memetic computing to solve complicated large scale optimization problems.

Some of our future work is to apply memetic computing to construct more challenging different types of optimal experimental designs for more complicated statistical models, such as random effects models or hierarchical models with various correlation structures and the models have more interacting explanatory variables. The design criterion may be non-differentiable, such as for finding a standardized maximin design, that requires solving a multi-level nested optimization problem. We are also interested to find Bayesian optimal designs that incorporate prior information of the model parameters at the design stage or find multiple-objective optimal designs, where some criteria are more important than others. We believe CSO-MA, singly or when appropriately hybridized with another metaheuristic algorithm, has potential for solving challenging and high-dimensional optimization problems in public health and medical studies.

Acknowledgements

Both Wong and Zhang were partially supported by a grant from the National Institute of General Medical Sciences of the National Institutes of Health under Award Number R01GM107639. The contents are solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

References

1. Bansal Jagdish Chand, Singh PK, Saraswat Mukesh, Verma Abhishek, Jadon Shimpi Singh, and Abraham Ajith. Inertia weight strategies in particle swarm optimization. In Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on, pages 633–640. IEEE, 2011.
2. Berger MPF and Wong WK. Applied optimal designs. Jonh Wiley & Sons, Chichester, West Sussex, UK, 2005.
3. Berger MPF and Wong WK. An introduction to optimal designs for social and biomedical research. Jonh Wiley & Sons, Chichester, West Sussex, UK, 2009.

4. Campos Mauro, Krohling Renato A, and Enriquez Ivan. Bare bones particle swarm optimization with scale matrix adaptation. IEEE Transactions on Cybernetics, 44(9):1567–1578, 2014. [PubMed: 25137686]
5. Carlisle Anthony and Dozier Gerry. Adapting particle swarm optimization to dynamic environments. In International Conference on Artificial Intelligence, volume 1, pages 429–434, 2000.
6. Chen Wei-Neng, Zhang Jun, Lin Ying, Chen Ni, Zhan Zhi-Hui, Chung Henry Shu-Hung, Li Yun, and Shi Yu-Hui. Particle swarm optimization with an aging leader and challengers. IEEE Transactions on Evolutionary Computation, 17(2):241–258, 2013.
7. Chen Xianshun, Ong Yew-Soon, Lim Meng-Hiot, and Tan Kay Chen. A multi-facet survey on memetic computation. IEEE Transactions on Evolutionary Computation, 15(5):591–607, 2011.
8. Cheng Ran and Jin Yaochu. A competitive swarm optimizer for large scale optimization. IEEE Transactions on Cybernetics, 45(2):191–204, 2015. [PubMed: 24860047]
9. Chi Rui, Su Yi-xin, Zhang Dan-hong, Chi Xue-xin, and Zhang Hua-jun. A hybridization of cuckoo search and particle swarm optimization for solving optimization problems. Neural Computing and Applications, 31(1):653–670, 2019.
10. Chi Yuhong, Sun Fuchun, Jiang Langfan, Yu Chunming, and Zhang Ping. Elastic boundary for particle swarm optimization. In International Conference in Swarm Intelligence, pages 125–132. Springer, 2012.
11. Deb Kalyanmoy, Pratap Amrit, Agarwal Sameer, and Meyarivan TAMT. A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation, 6(2):182–197, 2002.
12. Eberhart Russ C and Shi Yuhui. Comparing inertia weights and constriction factors in particle swarm optimization. In Evolutionary Computation, 2000. Proceedings of the 2000 Congress on, volume 1, pages 84–88. IEEE, 2000.
13. Eberhart Russel C and Kennedy James. Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks, volume 4, pages 1942–1948, 1995.
14. Eberhart Russell C and Shi Yuhui. Comparison between genetic algorithms and particle swarm optimization. In International Conference on Evolutionary Programming, pages 611–616. Springer, 1998.
15. Gang Ma, Wei Zhou, and Xiaolin Chang. A novel particle swarm optimization algorithm based on particle migration. Applied Mathematics and Computation, 218(11):6620–6626, 2012.
16. Gao Liang, Qian Weirong, Li Xinyu, and Wang Junfeng. Application of memetic algorithm in assembly sequence planning. The International Journal of Advanced Manufacturing Technology, 49(9–12):1175–1184, 2010.
17. Ghamisi Pedram and Benediktsson Jon Atli. Feature selection based on hybridization of genetic algorithm and particle swarm optimization. IEEE Geoscience and Remote Sensing Letters, 12(2):309–313, 2015.
18. Gu Shenkai, Cheng Ran, and Jin Yaochu. Feature selection for high-dimensional classification using a competitive swarm optimizer. Soft Computing, 22(3):811–822, 2018.
19. Haupt Randy L. and Haupt Sue Ellen. Practical genetic algorithms. John Wiley & Sons, 2004.
20. Higashi Natsuki and Iba Hitoshi. Particle swarm optimization with gaussian mutation. In Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE, pages 72–79. IEEE, 2003.
21. Luo Wenguang Ye Hongtao and Li Zhenqiang. Convergence analysis of particle swarm optimizer and its improved algorithm based on velocity differential evolution. Computational Intelligence and Neuroscience, 2013:7, 2013.
22. Hsieh Sheng-Ta, Sun Tsung-Ying, Liu Chan-Cheng, and Tsai Shang-Jeng. Solving large scale global optimization using improved particle swarm optimizer. In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on, pages 1777–1784. IEEE, 2008.
23. Ishaque Kashif and Salam Zainal. A deterministic particle swarm optimization maximum power point tracker for photovoltaic system under partial shading condition. IEEE Transactions on Industrial Electronics, 60(8):3195–3206, 2013.

24. Ishaque Kashif, Salam Zainal, Amjad Muhammad, and Mekhilef Saad. An improved particle swarm optimization (pso)-based mppt for pv with reduced steady-state oscillation. IEEE Transactions on Power Electronics, 27(8):3627–3638, 2012.
25. Kalantzis Georgios, Apte Aditya, Radke Richard, and Jackson Andrew. A reduced order memetic algorithm for constraint optimization in radiation therapy treatment planning. In 2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pages 225–230. IEEE, 2013.
26. Kennedy James. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, volume 3, pages 1931–1938. IEEE, 1999.
27. Kennedy James and Mendes Rui. Population structure and particle swarm performance. In Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on, volume 2, pages 1671–1676. IEEE, 2002.
28. Krasnogor Natalio and Smith James. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. IEEE Transactions on Evolutionary Computation, 9(5):474–488, 2005.
29. Kumarappan N and Arulraj R. Optimal installation of multiple dg units using competitive swarm optimizer (cso) algorithm. In Evolutionary Computation (CEC), 2016 IEEE Congress on, pages 3955–3960. IEEE, 2016.
30. Lall Shwetank, Jaggi Seema, Varghese Eldho, Varghese Cini, and Bhowmik Arpan. An algorithmic approach to construct d-optimal saturated designs for logistic model. Journal of Statistical Computation and Simulation, 88(6):1191–1199, 2018.
31. Larsen RB, Jouffroy J, and Lassen B. On the premature convergence of particle swarm optimization. In 2016 European Control Conference (ECC), pages 1922–1927, 6 2016.
32. Leung Yiu-Wing and Wang Yuping. An orthogonal genetic algorithm with quantization for global numerical optimization. IEEE Transactions on Evolutionary Computation, 5(1):41–53, 2001.
33. Li Xiaodong and Yao Xin. Cooperatively coevolving particle swarms for large scale optimization. IEEE Transactions on Evolutionary Computation, 16(2):210–224, 2012.
34. Liang Jane-Jing and Suganthan Ponnuthurai N. Dynamic multi-swarm particle swarm optimizer with local search. In Evolutionary Computation, 2005. The 2005 IEEE Congress on, volume 1, pages 522–528. Ieee, 2005.
35. Liu Bo, Wang Ling, Jin Yi-Hui, Tang Fang, and Huang De-Xian. Improved particle swarm optimization combined with chaos. Chaos, Solitons & Fractals, 25(5):1261–1271, 2005.
36. Liu Chen, Du Wen-Bo, and Wang Wen-Xu. Particle swarm optimization with scale-free interactions. PloS One, 9(5):e97822, 2014. [PubMed: 24859007]
37. McDermott J. When and why metaheuristics researchers can ignore "no free lunch" theorems. SN Computer Science, page In press, 2020.
38. Meng Ke, Wang Hong Gang, Dong ZhaoYang, and Wong Kit Po. Quantum-inspired particle swarm optimization for valve-point economic load dispatch. IEEE Transactions on Power Systems, 25(1):215–222, 2010.
39. Mohapatra Prabhujit, Das Kedar Nath, and Roy Santanu. A modified competitive swarm optimizer for large scale optimization problems. Applied Soft Computing, 59:340–362, 2017.
40. Moore Jacqueline and Chapman Richard. Application of particle swarm to multiobjective optimization. Department of Computer Science and Software Engineering, Auburn University, 32, 1999.
41. Morris Garrett M, Goodsell David S, Halliday Robert S, Huey Ruth, Hart William E, Belew Richard K, Olson Arthur J, et al. Automated docking using a lamarckian genetic algorithm and an empirical binding free energy function. Journal of Computational Chemistry, 19(14):1639–1662, 1998.
42. Nakisa Bahareh, Rastgoo Mohammad Naim, Norodin Md Jan, et al. Balancing exploration and exploitation in particle swarm optimization on search tasking. Research Journal of Applied Sciences, Engineering and Technology, 8(12):1429–1434, 2014.

43. Nezami Omid Mohamad, Bahrampour Anvar, and Jamshidlou Paria. Dynamic diversity enhancement in particle swarm optimization (ddepso) algorithm for preventing from premature convergence. Procedia Computer Science, 24:54–65, 2013.
44. Olorunda Olusegun and Engelbrecht Andries P. Measuring exploration/exploitation in particle swarms using swarm diversity. In 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), pages 1128–1134. IEEE, 2008.
45. Ong Yew-Soon, Lim Meng-Hiot, Zhu Ning, and Wong Kok-Wai. Classification of adaptive memetic algorithms: a comparative study. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 36(1):141–152, 2006.
46. Pázman Andrej. Foundations of optimum experimental design, volume 14. Springer, 1986.
47. Pehlivanoglu Y Volkan. A new particle swarm optimization method enhanced with a periodic mutation strategy and neural networks. IEEE Transactions on Evolutionary Computation, 17(3):436–452, 2013.
48. Pukelsheim Friedrich. Optimal design of experiments. SIAM, 2006.
49. Ratnaweera Asanga, Halgamuge Saman K, and Watson Harry C. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Transactions on Evolutionary Computation, 8(3):240–255, 2004.
50. Robinson Jacob, Sinton Seelig, and Rahmat-Samii Yahya. Particle swarm, genetic algorithm, and their hybrids: optimization of a profiled corrugated horn antenna. In Antennas and Propagation Society International Symposium, 2002. IEEE, volume 1, pages 314–317. IEEE, 2002.
51. Rodríguez-Torreblanca C and Rodríguez-Díaz JM. Locally d-and c-optimal designs for poisson and negative binomial regression models. Metrika, 66(2):161–172, 2007.
52. Ros Raymond and Hansen Nikolaus. A simple modification in cma-es achieving linear time and space complexity. In International Conference on Parallel Problem Solving from Nature, pages 296–305. Springer, 2008.
53. Shi Yuhui and Eberhart Russell C. Parameter selection in particle swarm optimization. In International Conference on Evolutionary Programming, pages 591–600. Springer, 1998.
54. Shi Yuhui and Eberhart Russell C. Fuzzy adaptive particle swarm optimization. In Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, volume 1, pages 101–106. IEEE, 2001.
55. Song Xian-fang, Zhang Yong, Guo Yi-nan, Sun Xiaoyan, and Wang Yong-li. Variable-size cooperative coevolutionary particle swarm optimization for feature selection on high-dimensional data. IEEE Transactions on Evolutionary Computation, 14(8):1–14, 2019.
56. Storn Rainer and Price Kenneth. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 11(4):341–359, 1997.
57. Suganthan Ponnuthurai N. Particle swarm optimiser with neighbourhood operator. In Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, volume 3, pages 1958–1962. IEEE, 1999.
58. Sun Chaoli, Ding Jinliang, Zeng Jianchao, and Jin Yaochu. A fitness approximation assisted competitive swarm optimizer for large scale expensive optimization problems. Memetic Computing, pages 1–12, 2016.
59. Sun Jun, Palade Vasile, Wu Xiao-Jun, Fang Wei, and Wang Zhenyu. Solving the power economic dispatch problem with generator constraints by random drift particle swarm optimization. IEEE Transactions on Industrial Informatics, 10(1):222–232, 2014.
60. Tang Ke, Yáo Xin, Suganthan Ponnuthurai Nagaratnam, MacNish Cara, Chen Ying-Ping, Chen Chih-Ming, and Yang Zhenyu. Benchmark functions for the cecâĂŹ2008 special session and competition on large scale global optimization. Nature Inspired Computation and Applications Laboratory, USTC, China, 24, 2007.
61. Taormina Riccardo and Chau Kwok-Wing. Datadriven input variable selection for rainfall-runoff modeling using binary-coded particle swarm optimization and extreme learning machines. Journal of Hydrology, 529:1617–1632, 2015.
62. Tian Jing, Yu Weiyu, and Xie Shengli. An ant colony optimization algorithm for image edge detection. In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on, pages 751–756. IEEE, 2008.

63. Trelea Ioan Cristian. The particle swarm optimization algorithm: convergence analysis and parameter selection. Information Processing Letters, 85(6):317–325, 2003.
64. Valenzuela Jorge and Smith Alice E. A seeded memetic algorithm for large unit commitment problems. Journal of Heuristics, 8(2):173–195, 2002.
65. Whitley Darrell. A genetic algorithm tutorial. Statistics and computing, 4(2):65–85, 1994.
66. Willan Andrew R and Pinto Eleanor M. The value of information and optimal clinical trial design. Statistics in Medicine, 24(12):1791–1806, 2005. [PubMed: 15806619]
67. Worasucheep C. A particle swarm optimization for high-dimensional function optimization. In ECTI-CON2010: The 2010 ECTI International Confernce on Electrical Engineering/Electronics, Computem, Telecommunications and Information Technology, pages 1045–1049, May 2010.
68. Xinchao Zhao. A perturbed particle swarm algorithm for numerical optimization. Applied Soft Computing, 10(1):119–124, 2010.
69. Xiong Guojiang and Shi Dongyuan. Orthogonal learning competitive swarm optimizer for economic dispatch problems. Applied Soft Computing, 2018.
70. Xu Gang, Wu Zhi-Hua, and Jiang Mei-Zhen. Premature convergence of standard particle swarm optimisation algorithm based on markov chain analysis. International Journal of Wireless and Mobile Computing, 9(4):377–382, 2015.
71. Yang Qiang, Chen Wei-Neng, Gu Tianlong, Zhang Huaxiang, Yuan Huaqiang, Kwong Sam, and Zhang Jun. A distributed swarm optimizer with adaptive communication for large-scale optimization. IEEE Transactions on Cybernetics, 2019.
72. Yang Xin-She and Deb Suash. Cuckoo search via lévy flights. In 2009 World congress on nature & biologically inspired computing (NaBIC), pages 210–214. IEEE, 2009.
73. Yang Zhenyu, Tang Ke, and Yao Xin. Multilevel cooperative coevolution for large scale optimization. In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on, pages 1663–1670. IEEE, 2008.
74. Yang Zhenyu, Tang Ke, and Yao Xin. Self-adaptive differential evolution with neighborhood search. In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on, pages 1110–1116. IEEE, 2008.
75. Zhang Qiang, Cheng Hui, Ye Zhencheng, and Wang Zhenlei. A competitive swarm optimizer integrated with cauchy and gaussian mutation for large scale optimization. In Control Conference (CCC), 2017 36th Chinese, pages 9829–9834. IEEE, 2017.
76. Zhang Wen-Xiao, Chen Wei-Neng, and Zhang Jun. A dynamic competitive swarm optimizer based-on entropy for large scale optimization. In Advanced Computational Intelligence (ICACI), 2016 Eighth International Conference on, pages 365–371. IEEE, 2016.
77. Zhou Jianhong, Fang Wei, Wu Xiaojun, Sun Jun, and Cheng Shi. An opposition-based learning competitive particle swarm optimizer. In Evolutionary Computation (CEC), 2016 IEEE Congress on, pages 515–521. IEEE, 2016.
78. Zibakhsh A and Saniee Abadeh M. Gene selection for cancer tumor detection using a novel memetic algorithm with a multi-view fitness function. Engineering Applications of Artificial Intelligence, 26(4):1274–1281, 2013.
79. Zitzler Eckart, Deb Kalyanmoy, and Thiele Lothar. Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation, 8(2):173–195, 2000. [PubMed: 10843520]

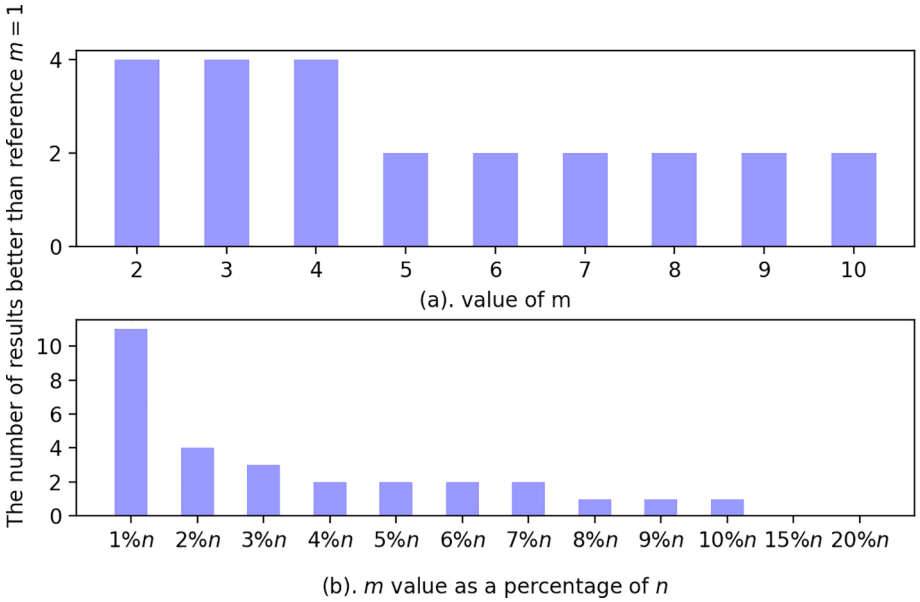

Fig. 1.

The number of significantly better results found by the algorithm using different m values compared to using $m = 1$ for optimizing the 24 benchmark functions. Parameter n denotes the swarm size.

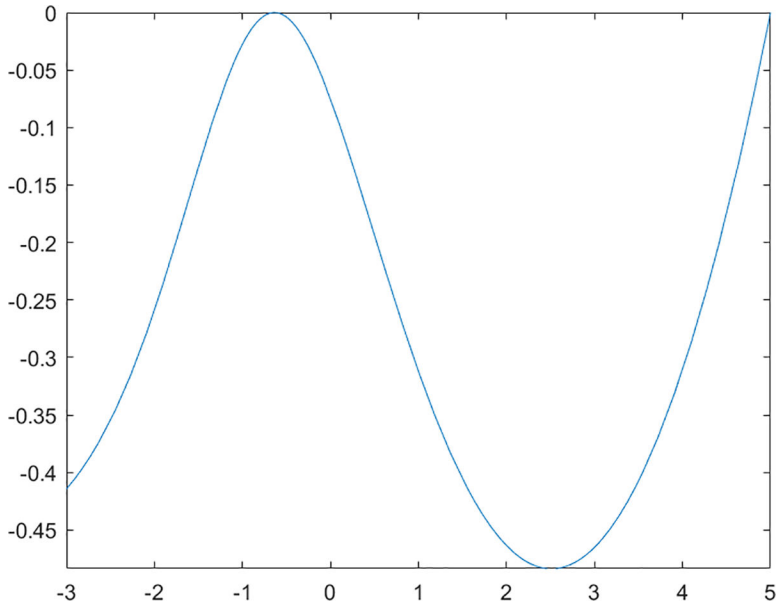

Fig. 2.

The sensitivity function of the *c*-optimal design for a one-factor negative binomial model on the design space $x \in [-3, 5]$ with $a = 3$, $\boldsymbol{\theta} = (0.5, 1.7)^T$ and $\mathbf{c} = (0, 1)^T$. The plot confirms optimality of the CSO-generated design supported at −0.637 and 5.000.

Table 1

Performances of the nine algorithms for minimizing the eight 100D benchmark functions. The values in the last column "**w/t/l**" shows the number of times CSO-MA wins (significantly better), ties (insignificant difference) and loses (significantly worse) to other algorithms using the Wilcoxon rank test at the 0.05 significance level. The bold numbers indicate results for the best performing algorithm among the nine for minimizing each of the benchmark functions.

		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	w/t/l
CSO	Mean	1.37E+01	1.18E+02	2.05E−72	5.29E+01	6.30E+03	−7.15E+01	**2.22E−16**	**4.44E−15**	5/2/1
	Std dev	2.25E+00	1.03E+01	2.42E−73	1.78E+00	1.31E+02	1.07E+00	**0.00E+00**	**0.00E+00**	
CSO-MA	Mean	**8.67E−03**	9.05E+01	1.88E−33	5.33E−06	**8.15E+02**	**−8.69E+01**	**2.22E−16**	**4.44E−15**	–
	Std dev	**2.08E−04**	3.60E+00	3.57E−35	1.52E−06	**1.59E+01**	**2.27E−01**	**0.00E+00**	**0.00E+00**	
MCSO	Mean	5.22E+00	8.97E+01	**7.11E−78**	9.28E+01	7.14E+03	−7.03E+01	**2.22E−16**	**4.44E−15**	5/2/1
	Std dev	9.07E−01	2.33E+00	**3.56E−78**	8.01E−01	2.05E+02	1.33E+00	**0.00E+00**	**0.00E+00**	
CCPSO2	Mean	7.11E+00	4.21E+02	7.56E−14	3.88E−02	3.62E+03	−6.25E+01	3.41E−03	1.61E−13	8/0/0
	Std dev	7.68E+00	8.72E+01	3.41E−14	1.98E−01	4.19E+02	6.68E+00	1.42E−02	5.20E−12	
MLCC	Mean	3.44E+01	1.52E+02	5.29E−14	**4.65E−13**	1.12E+03	−8.07E+01	1.59E−12	1.06E−12	7/0/1
	Std dev	8.70E+00	5.34E+01	2.35E−14	**9.15E−14**	8.36E+01	3.72E−01	7.77E−13	9.24E−15	
SEP-CMA-ES	Mean	5.15E+01	**4.88E+00**	7.44E−14	2.93E+02	2.65E+03	−7.88E+01	3.50E−03	2.06E+01	6/0/2
	Std dev	1.91E+01	**1.53E+00**	9.06E−15	4.76E+01	2.49E+02	3.02E+00	1.71E−02	8.53E−03	
EPUS-PSO	Mean	2.24E+01	4.75E+03	9.02E−01	4.55E+02	5.79E+03	−6.74E+01	2.99E−01	2.05E+00	8/0/0
	Std dev	1.11E+00	3.80E+02	8.29E−02	1.04E+01	9.53E+01	3.21E−01	2.30E−02	2.20E−01	
DMS-PSO	Mean	6.24E+00	2.86E+02	1.05E−20	1.73E+02	2.66E+03	−7.21E+01	6.52E−10	5.49E−13	8/0/0
	Std dev	5.22E−01	3.18E+01	6.61E−22	3.52E+01	1.66E+02	1.98E+00	2.21E−11	9.86E−14	
Cuckoo	Mean	3.45E+01	6.57E+02	8.54E−01	4.22E+02	4.13E+03	−4.74E+01	3.62E−01	7.62E+00	8/0/0
	Std dev	1.17E+00	4.95E+01	1.06E−02	1.42E+01	1.53E+03	4.33E+00	7.72E−04	3.29E−01	

Table 2

Performances of the nine algorithms for minimizing the eight 500D benchmark functions. The values in the last column "**w/t/l**" shows the number of times CSO-MA wins (significantly better), ties (insignificant difference) and loses (significantly worse) to other algorithms using the Wilcoxon rank test at the 0.05 significance level. The bold numbers indicate results for the best performing algorithm among the nine for minimizing each of the benchmark functions.

		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	w/t/l
CSO	Mean	4.58E+01	4.80E+02	1.92E−66	1.58E+02	4.94E+04	−2.89E+02	**4.44E−16**	**8.88E−15**	4/0/4
	Std dev	9.47E+00	5.29E+00	3.99E−67	8.31E+00	7.53E+02	6.33E+00	**0.00E+00**	**0.00E+00**	
CSO-MA	Mean	**6.88E+00**	6.99E+02	5.34E−03	3.84E+01	**2.10E+04**	**−3.50E+02**	6.88E−05	4.40E−04	-
	Std dev	**9.41E−01**	4.52E+00	7.11E−04	2.07E+00	**4.25E+02**	**1.62E+00**	8.73E−07	2.12E−05	
MCSO	Mean	8.34E+01	9.35E+02	**1.42E−83**	8.57E+02	5.67E+04	−2.43E+02	1.23E−02	1.53E−14	6/0/2
	Std dev	4.42E+00	1.30E+01	**7.62E−85**	1.19E+01	4.07E+03	8.11E+00	3.24E−03	5.69E−15	
CCPSO2	Mean	6.32E+01	7.55E+02	6.19E−11	4.04E+00	4.65E+04	−3.15E+02	1.06E−03	4.32E−13	5/0/3
	Std dev	5.22E+00	4.57E+00	3.69E−12	5.29E−01	7.66E+02	6.03E+00	2.16E−03	5.57E−14	
MLCC	Mean	7.05E+01	9.14E+02	3.64E−13	**2.02E−11**	4.77E+04	−2.69E+02	2.15E−13	4.21E−13	4/0/4
	Std dev	5.82E+00	7.61E+01	6.28E−14	**3.05E−11**	2.78E+03	1.01E+01	2.45E−13	3.94E−13	
SEP-CMA-ES	Mean	6.05E+01	**2.87E+02**	2.33E−14	2.22E+03	3.54E+04	−2.73E+02	8.06E−04	3.00E+01	6/0/2
	Std dev	1.00E+00	**2.75E+01**	3.28E−15	1.57E+02	4.29E+02	2.10E+00	2.90E−03	4.31E−01	
EPUS-PSO	Mean	4.40E+01	5.63E+04	8.22E+00	4.03E+03	7.62E+04	−2.85E+02	5.95E−02	5.56E−01	8/0/0
	Std dev	5.51E−01	4.14E+03	2.01E+00	1.12E+02	1.62E+03	3.07E+00	3.99E−03	2.04E−02	
DMS-PSO	Mean	7.35E+01	2.85E+04	5.27E−06	4.29E+03	4.30E+04	−2.88E+02	1.57E−05	8.59E+00	6/0/2
	Std dev	4.00E+00	9.14E+02	8.86E−08	7.02E+01	9.35E+02	7.00E+00	2.46E−06	4.33E−01	
Cuckoo	Mean	6.03E+01	4.27E+04	3.27E+00	6.67E+02	6.09E+04	−1.66E+02	7.02E+00	2.54E+01	8/0/0
	Std dev	2.47E+00	1.03E+03	1.02E+00	2.11E+01	1.44E+02	3.57E+00	4.19E−01	1.16E+00	

Table 3

Performances of the nine algorithms for minimizing the eight 1000D benchmark functions. The values in the last column "**w/t/l**" shows the number of times CSO-MA wins (significantly better), ties (insignificant difference) and loses (significantly worse) to other algorithms using the Wilcoxon rank test at the 0.05 significance level. The bold numbers indicate results for the best performing algorithm among the nine for minimizing each of the benchmark functions.

		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	w/t/l
CSO	Mean	2.80E+01	1.45E+03	7.50E−02	1.60E+02	2.38E+05	−3.69E+02	1.30E−02	4.90E−01	8/0/0
	Std dev	3.62E+00	1.94E+02	2.09E−05	4.16E+01	3.81E+03	3.54E+00	2.64E−04	6.68E−03	
CSO-MA	Mean	**2.01E+01**	**1.19E+03**	4.70E−03	1.53E+02	**9.95E+04**	**−4.98E+02**	1.24E−02	3.01E−03	-
	Std dev	**8.27E−01**	**9.12E+01**	5.90E−04	6.31E+00	**1.88E+03**	**9.84E−01**	2.62E−04	9.40E−05	
MCSO	Mean	8.17E+01	2.03E+03	2.97E−66	2.34E+03	1.35E+05	−4.07E+02	**2.85E−15**	1.24E+00	6/0/2
	Std dev	3.59E−01	3.02E+01	9.94E−68	2.64E+01	1.11E+04	6.00E+00	**1.61E−05**	3.72E−01	
CCPSO2	Mean	7.45E+01	1.33E+03	5.29E−13	3.05E−01	2.45E+05	−3.57E+02	3.00E+00	**1.06E−12**	5/0/3
	Std dev	3.98E+00	1.17E+02	9.54E−14	2.60E−01	9.33E+02	1.07E+00	8.22E−01	**3.77E−13**	
MLCC	Mean	8.99E+01	1.82E+03	8.45E−13	**3.66E−10**	1.88E+05	−3.46E+02	4.18E−07	1.06E−12	4/0/4
	Std dev	2.65E+00	1.53E+02	4.67E−14	**4.92E−11**	4.87E+03	9.06E+00	2.47E−13	4.82E−13	
SEP-CMA-ES	Mean	4.22E+01	2.12E+03	5.92E−11	5.60E+03	2.25E+05	−3.11E+02	3.66E−04	3.42E+01	6/0/2
	Std dev	5.07E+00	7.93E+01	4.41E−13	2.17E+02	9.45E+03	3.77E+00	1.08E−05	2.26E+00	
EPUS-PSO	Mean	5.13E+01	9.66E+04	3.98E+02	4.57E+03	6.60E+05	−2.56E+02	7.44E+00	1.56E+01	8/0/0
	Std dev	1.07E+00	1.08E+03	1.77E+01	1.49E+02	1.15E+04	3.03E+00	9.62E−01	1.07E+00	
DMS-PSO	Mean	9.15E+01	5.74E+04	3.29E−03	3.83E+03	7.75E+05	−3.03E+02	4.11E+00	1.10E+01	7/0/1
	Std dev	3.44E−01	1.55E+03	3.12E−05	9.54E+01	9.29E+03	8.07E+00	5.50E−01	4.82E−01	
Cuckoo	Mean	8.26E+01	9.02E+04	2.34E+01	2.54E+03	8.66E+05	−1.47E+02	1.62E+01	4.60E+01	8/0/0
	Std dev	2.06E+00	2.63E+02	5.52E−01	3.06E+01	1.11E+04	2.98E+00	5.10E−01	1.66E+00	

Table 4

The number of times on algorithms' ranks when minimizing f_1 to f_8 ($D = 100, 500, 1000$). A smaller rank value indicates a better minimization performance of the algorithm optimizing the specific function.

	Number of times	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8	Rank 9	Average Rank
$D = 100$	CSO	0	3	0	2	2	0	0	1	0	3
	CSO-MA	3	3	2	0	0	0	0	0	0	**1**
	MCSO	1	4	0	0	1	1	0	0	1	2
	CCPSO2	0	0	1	2	1	1	2	1	0	6
	MLCC	1	2	0	1	2	1	1	0	0	3
	SEP-CMA-ES	1	0	2	0	0	1	2	0	2	7
	EPUS-PSO	0	0	0	0	0	1	3	1	3	8
	DMS-PSO	0	0	1	3	2	2	0	0	0	5
	Cuckoo	0	0	0	0	0	1	0	5	2	9
$D = 500$	CSO	2	2	2	1	0	1	0	0	0	**1**
	CSO-MA	3	0	2	1	1	0	1	0	0	**1**
	MCSO	1	1	0	0	0	2	2	1	1	6
	CCPSO2	0	2	0	3	1	2	0	0	0	3
	MLCC	1	1	1	1	2	0	2	0	0	4
	SEP-CMA-ES	1	1	1	0	2	1	1	0	1	5
	EPUS-PSO	0	1	0	0	1	1	0	2	3	8
	DMS-PSO	0	0	2	1	0	1	2	1	1	7
	Cuckoo	0	0	0	1	1	0	0	4	2	9
$D = 1000$	CSO	0	1	2	2	2	0	1	0	0	5
	CSO-MA	4	0	2	1	0	1	0	0	0	**1**
	MCSO	2	2	0	0	3	1	0	0	0	2
	CCPSO2	1	3	0	1	1	2	0	0	0	4
	MLCC	2	1	2	1	1	0	0	1	0	2
	SEP-CMA-ES	0	0	2	2	0	2	0	1	1	6
	EPUS-PSO	0	0	0	1	0	0	2	3	2	8
	DMS-PSO	0	0	0	0	1	1	4	1	1	7

	Number of times	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8	Rank 9	Average Rank
	Cuckoo	0	0	0	0	0	1	1	2	4	9

Table 5

Average swarm diameter measured at the (1^{st}, $2500D^{th}$, $5000D^{th}$) function evaluations when CSO and CSO-MA are applied to minimize the benchmark functions f_4 to f_8, which all have many local minima. The "CSO's Results" and "CSO-MA's results" columns display the optimal values of the functions found by CSO and CSO-MA when it terminates at the $5000D^{th}$ evaluation. An asterisk indicates the algorithm has performed significantly better the other at the $\alpha = 0.05$ significance level.

Function	Dimension	CSO's *Dia*	CSO's Results	CSO-MA's *Dia*	CSO-MA's Results
f_4	$D = 100$	(50.40, 0.00, 0.00)	5.29E+01	(52.61, 6.85, 5.30)	5.33E−06*
	$D = 500$	(103.73, 0.00, 0.00)	1.58E+02	(103.76, 5.21, 5.26)	3.84E+01*
	$D = 1000$	(143.18, 0.00, 0.00)	1.60E+02	(143.92, 3.48, 6.52)	4.16E+01*
f_5	$D = 100$	(4878.05, 0.00, 0.00)	6.30E+03	(4871.30, 914.08, 965.59)	1.31E+02*
	$D = 500$	(10069.29, 0.00, 0.00)	4.94E+04	(10057.30, 905.36, 1061.222)	7.53E+02*
	$D = 1000$	(14047.72, 0.00, 0.00)	2.38E+05	(14051.95, 604.02, 798.37)	3.81E+03*
f_6	$D = 100$	(9.87, 0.00, 0.00)	−7.15E+01	(9.75, 0.63, 1.97)	−8.69E+01*
	$D = 500$	(19.98, 0.00, 0.00)	−2.89E+02	(19.62, 1.37, 0.67)	−3.50E+02*
	$D = 1000$	(27.84, 0.00, 0.00)	−3.69E+02	(28.11, 1.12, 1.09)	−4.98E+02*
f_7	$D = 100$	(5829.34, 0.00, 0.00)	2.22E−16	(5816.03, 745.05, 611.29)	2.22E−16
	$D = 500$	(12196.04, 0.00, 0.00)	4.44E−16*	(12230.41, 708.34, 715.51)	6.88E−05
	$D = 1000$	(16750.82, 0.00, 0.00)	1.30E−02	(16779.22, 681.06, 720.33)	1.24E−02
f_8	$D = 100$	(315.39, 0.00, 0.00)	4.44E−15	(318.09, 44.81, 44.29)	4.44E−15
	$D = 500$	(655.34, 0.00, 0.00)	8.88E−15*	(639.74, 39.53, 41.36)	4.40E−04
	$D = 1000$	(893.27, 0.00, 0.00)	4.90E−01	(885.87, 43.27, 43.41)	3.01E−03*

Table 6

Runtime for CSO and CSO-MA completing 5000D evaluations on benchmark functions. Average results are given based on 25 independent runs for each function.

	100D		**500D**		**1000D**	
	CSO	**CSO-MA**	**CSO**	**CSO-MA**	**CSO**	**CSO-MA**
f_1	3.0s	3.1s	94.5s	92.6s	412.3s	417.1s
f_2	6.6s	6.6s	180.4s	181.9s	746.2s	758.6s
f_3	3.0s	2.9s	92.9s	86.3s	411.7s	420.2s
f_4	9.2s	9.1s	247.8s	249.0s	998.0s	1025.5s
f_5	6.2s	6.2s	165.6s	165.0s	723.5s	734.6s
f_6	9.7s	9.7s	245.2s	260.1s	1040.5s	1052.1s
f_7	10.1s	10.0s	265.0s	271.4s	1144.2s	1150.3s
f_8	9.5s	9.9s	257.7s	255.2s	1053.1s	1064.7s

Table 7

Four simulated sets of values for θ_0, θ_1, ⋯, θ_{15}, each drawn from $U(-2, 2)$ randomly for the negative binomial models with 5 factors and all pairwise interactions.

$\boldsymbol{\theta}$	$\boldsymbol{\theta_0}$, $\boldsymbol{\theta_1}$, $\boldsymbol{\theta_2}$, ⋯, $\boldsymbol{\theta_{14}}$, $\boldsymbol{\theta_{15}}$
$\boldsymbol{\theta_1}$	$(1.03, 0.50, 0.75, 1.25, 0.80, 0.50, 1.80, -0.40, -1.00, 1.65, 0.65, 1.10, 1.30, 0.40, -0.20, 0.55)^T$
$\boldsymbol{\theta_2}$	$(0.09, -0.42, 1.13, 0.29, -0.20, -0.11, 1.40, 0.60, 0.58, -0.25, -0.35, 1.52, 1.01, 0.60, -0.07, 0.09)^T$
$\boldsymbol{\theta_3}$	$(-1.12, -0.66, 1.50, 0.14, -0.08, -1.01, 1.65, -0.03, 1.74, 0.62, -0.12, -0.68, -1.71, 0.40, 0.95, -1.89)^T$
$\boldsymbol{\theta_4}$	$(0.29, -0.20, -0.33, 1.54, 1.22, 0.44, 1.03, 0.71, -0.55, -0.29, 1.55, -0.47, 0.56, -1.30, 0.17, 0.66)^T$

Table 8

Mean *c*-optimality criterion values of the designs generated by the five algorithms after 5000*D* function evaluations for the negative binomial regression models with five factors and all pairwise interactions. The *c*-efficiency of each design relative to the CSO-MA generated design is in parentheses. The last row records average runtime for each algorithm.

	PSO	CSO	DE	MCSO	CSO-MA
θ_1	−0.56 (63%)	−0.38 (93%)	−0.52 (65%)	−0.45 (85%)	−0.35
θ_2	−0.64 (74%)	−0.41 (94%)	−0.57 (71%)	−0.51 (90%)	−0.36
θ_3	−0.90 (80%)	−0.74 (97%)	−0.80 (78%)	−0.79 (92%)	−0.68
θ_4	−0.49 (72%)	−0.33 (87%)	−0.44 (69%)	−0.47 (80%)	−0.29
Time	73.9 s	67.0s	82.1s	77.4s	68.5s