Gnowee: A Hybrid Metaheuristic Optimization Algorithm for Constrained, Black Box, Combinatorial Mixed-Integer Design

James E. Bevins[a,*] R. N. Slaybaugh[b]

[a]*Air Force Institute of Technology, Department of Engineering Physics*
2950 Hobson Way, Wright-Patterson Air Force Base, OH 45433, USA

[b]*University of California, Berkeley, Nuclear Engineering Department*
4173 Etcheverry Hall, Berkeley, CA 94720, USA

[*]Email: james.bevins@afit.edu

Number of pages: 43

Number of tables: 7

Number of figures: 6

Abstract

This paper introduces Gnowee, a modular, Python-based, open-source hybrid metaheuristic optimization algorithm (available from `https://github.com/SlaybaughLab/Gnowee`). Gnowee is designed for rapid convergence to nearly globally optimum solutions for complex, constrained nuclear engineering problems with mixed-integer and combinatorial design vectors and high-cost, noisy, discontinuous, black box objective function evaluations. Gnowee's hybrid metaheuristic framework is a new combination of a set of diverse, robust heuristics that appropriately balance diversification and intensification strategies across a wide range of optimization problems.

There are many potential applications for this novel algorithm both within the nuclear community and beyond. Given that a set of well-known and studied nuclear benchmarks do not exist for the purpose of testing optimization algorithms, comparisons between Gnowee and several well-established metaheuristic algorithms are made for a set of eighteen established continuous, mixed-integer, and combinatorial benchmarks representing a wide range of types of engineering problems and solution space behaviors. These results demonstrate Gnoweee to have superior flexibility and convergence characteristics over this diverse set of design spaces. We anticipate this wide range of applicability will make this algorithm desirable for many complex engineering applications.

Keywords: Evolutionary computation, metaheuristic, optimization algorithms

2

I INTRODUCTION

Many engineering problems and processes can be described by multi-dimensional surfaces that form the "fitness" landscape of the solution [1, 2, 3, 4]. A single point on the fitness landscape is obtained through evaluation of one n-dimensional solution set against desired objectives such as performance (in the example case, degree of spectrum matching), size, cost, etc. All solution sets that meet the given design constraints of a problem form the complete fitness landscape, which can have many locally-optimum solution sets (multi-modal).

One example that motivated this research is the challenge of designing a material stack-up to efficiently modify a neutron energy spectrum from one produced by an available neutron source to one that is desired for a specific application subject to the physical constraints of the implementation of such a stack-up design at the facility. The ability to develop custom neutron spectra would potentially address capability gaps faced by nuclear forensics, medical isotope production, medical physics, and nuclear data communities, to name a few. However, the combination of a large n-dimensional search space, a noisy, multi-modal fitness landscape, and a complex and/or expensive objective function evaluation, often via a physics solver code like Monte Carlo simulations, makes the determination of the globally optimum solution virtually impossible via intuition or parametric studies. All or many of these characteristics are true for a variety of nuclear design challenges, and formal optimization techniques offer an opportunity to improve the solutions available for these types of design challenges while simultaneously increasing the rate of research and development.

At its most basic, an optimization problem can be stated as

$$
\begin{aligned}
\underset{\vec{x} \in \mathbb{R}^d}{\text{Minimize}} \qquad & f_i(\vec{x}), & (i = 1,\ 2,\ \ldots,\ I)^T, \\
\text{Subject to:} \qquad & g_j(\vec{x}) \leq 0, & (j = 1,\ 2,\ \ldots,\ J), \\
& h_k(\vec{x}) = 0, & (k = 1,\ 2,\ \ldots,\ K),
\end{aligned}
\tag{1}
$$

where I, J, and K are the number of objective functions, inequality, and equality constraints, respectively. $\vec{x}$ is the candidate design solution given by $\vec{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^d$, where $\mathbb{R}^d$ is the design space. Optimization problems can be classified by sub-setting the mathematical formulation in ways that are illustrative to assessing the requirements for a given optimization algorithm. Some of the more general classifications are considered here: single objective ($I = 1$)

versus multi-objective ($I > 1$); linear versus nonlinear objective function(s); differentiable versus derivative free objective function(s); unconstrained ($J = K = 0$) versus constrained ($J|K > 0$); continuous, discrete, combinatorial, or mixed-integer (MI) design space ($\mathbb{R}^d$); and uni-modal (convex) versus multi-modal (non-convex) fitness landscapes ($f(\vec{x})$) [5].

In this work, we develop an algorithm designed for nuclear engineering applications that fall into the subset of the single objective, nonlinear, derivative-free, constrained or unconstrained, combinatorial _and_ mixed-integer, multi-modal categorizations of optimization problems. These nuclear engineering applications typically have black box objective functions and/or constraint evaluations, may not be continuous, and can be very noisy due to limitations in the model or underlying nuclear data. While the categorizations addressed by this work cover most of the problem space for nuclear engineering applications and design problems of interest, there are many applications that could benefit from or require considering multiple objective functions. This is not yet explicitly considered for the algorithm presented here.

The literature is filled with examples of optimization algorithms designed to solve problems with either continuous, discrete, or combinatorial design vectors [1, 6, 7, 8, 9, 10, 11]. For many engineering problems of interest, this approach can be valid directly for limited cases with one design variable type, or made to work through mapping discrete variables onto continuous space or subsetting combinatorial variations. However, many practical engineering problems would benefit from a combined approach to fully sample the design space. For example, consider the design of a composite shield for mixed radiation [12]. The full design variable can be given by

$$\vec{x} = (x_1, x_2, x_3),\tag{2}$$

where x_1 is the $N \times M$ matrix of N materials with up to M elemental components, x_2 is the vector of ordering for the N composite materials, and x_3 is the vector of thicknesses for N composite materials.

Hu et al. [12] made the radiation shielding problem continuous and solved the optimization problem stepwise by first optimizing x_1 and then optimizing x_3 for a small subset of layered material combinations. However, a more robust approach that samples the full solution phase space would treat the optimization of continuous x_3, combinatorial x_2, and discrete x_1 concurrently. We assert that this concurrent treatment would increase design automation, provide better solutions, and

achieve them more quickly than current approaches.

The field of mixed-integer nonlinear programming (MINLP) most closely addresses this problem [13, 14]. Many techniques have been employed to solve MINLP through relaxation of the problem space and constraints, convexation of the problem, or application of problem-specific deterministic approaches, but these are not generally applicable or successful when faced with black box, noisy, discontinuous, or combinatorial problems [14, 15, 16]. Surrogates and response surface approaches have proven quite successful at addressing those limitations, but these approaches are often problem specific and/or still require smooth, continuous fitness landscapes [16, 17]. On the other hand, metaheuristic approaches are adaptable to a wide range of problems, and techniques have been developed to solve MI, continuous, discrete, and combinatorial problems [10, 18, 19, 20]. A metaheuristic approach is "a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms" [21].

The new hybrid metahueristic optimization algorithm, Gnowee, is a nearly globally convergent algorithm capable of handling noisy, black box, nonlinear, and discontinuous objectives and constraints informed by continuous, integer/binary, discrete, and/or combinatorial design spaces. Gnowee combines elements from many well established algorithms under the construct of a diverse, coherent, metaheuristic framework that rapidly converges to a minimum fitness threshold, but no global convergence is guaranteed or evaluated [8, 21, 22]. Emphasis is placed on nearly global convergence (as opposed to true global convergence) as Gnowee is developed for complex engineering design challenges with high-cost objective function evaluations. For such problems, the differences between the performance of the true global optimum versus a nearly global optimum is likely to be within the precision of the black box model and underlying data used. Further, it is not useful to improve design precision beyond the point practical for construction (e.g. consider machining tolerances and cost).

This paper is organized as follows. Section II provides useful background information. Section III describes the new Gnowee algorithm. The benchmark calculation settings and process are described in Section IV, and the benchmark results are presented in SectionV. Finally, Section VI summarizes the findings obtained using Gnowee and indicates possible directions for future research.

II BACKGROUND

In this section, a basic overview of metaheuristic optimization is covered. Next, classifications of metaheuristic search operators (heuristics) are discussed. Finally, Lévy flights are described for use in developing Markov chains.

II.A Metaheuristics Overview

The term metaheuristic was coined by Glover in 1986 by combining the Greek prefix meta- (metá, beyond in the sense of high-level) with heuristic (from the Greek heuriskein or euriskein, to search) [21, 23]. Sorensen and Glover define metaheuristic as "a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms" [21]. Originally intended to describe the framework of search strategies, the term *metaheuristic* is commonly used as being synonymous with nature-inspired stochastic algorithms, of which there are numerous and varied implementations in the literature [10, 24]. For the purposes of this paper, metaheuristics will be used to refer to the search technique framework developed, and heuristic will be used to refer to the specific search technique used.

While not holding to any specific metaphor of a nature-inspired optimization process, algorithmic implementations for many heuristics used by Gnowee were derived from those common to nature-inspired optimization algorithms. Traditional details of the processes mimicked by those algorithms are bypassed, and a description of the search strategies and algorithmic implementation are provided instead. By following this approach to algorithm development, i.e. developing a coherent, synergistic set of search strategies instead of adhering strictly to an optimization metaphor, a more robust optimization algorithm can be obtained while tailoring the resulting convergence characteristics. Finally, the No Free Lunch Theorem states that no algorithm will be universally superior, and this approach of identifying the heuristic search strategies allows for quick identification of the strengths and weaknesses of the metaheuristic framework constructed [25]. The identification of strengths and weaknesses through characterization of the heuristics employed can be used to assess the applicability of an algorithm to a specific problem.

II.B Metaheuristic Search Strategies

Lones [22] details a list of ten heuristic search operators that can be used to describe and classify metaheuristic-based algorithms. While not all inclusive nor describing the many variations of each approach in the literature, it provides a foundation upon which a metaheuristic framework can be built. It is beyond the scope of this paper to reproduce or expand upon Lones's results here, but the basic concepts and applicability of the heuristic search strategies are described for completeness. Additionally, the set of heuristics are included as a guide to understand the design decisions made through analyzing the inclusion, exclusion, and/or combination of different heuristics within the Gnowee metaheuristic framework.

The heuristic search strategies presented by Lones are [22]:

- **Neighborhood Search:** New solutions developed are a move, or step change, away from current solutions. A move can be defined as a change in a single variable or the entire solution. While the neighborhood can be defined and sampled multiple ways, the search carried out typically is local in nature.

- **Hill Climbing:** A sequence of moves is used to find a locally optimum solution where moves are only accepted if they lead to an improvement in fitness. The acceptance of only positive moves along a search path can lead to local optima trapping if not combined with another global search heuristic.

- **Accepting Negative Moves:** Allows moves to worse fitness solutions. Often used to prevent premature convergence at local optima.

- **Multi-Start:** Restart the search process with a new starting location once it converges at a local optimum. The use of a population can be considered a form of multi-start. The use of restarts and/or population-based techniques reduces local optima trapping in large search spaces.

- **Adaptive Memory Programming:** Uses past search experience to guide development of the next move. This can range from calculating random steps from the current design point to stored histories of recent moves. These strategies can help guide the search away from well explored regions, preventing cyclical patterns.

7

- **Population-Based Search:** Multiple search processes that are often cooperative and executed in parallel. In cooperative executions, information sharing can be tightly or loosely coupled. For large, multi-modal search spaces, this can accelerate global convergence, though it can be inefficient in relatively easy problems.

- **Intermediate Search:** Explores the space between two or more high fitness solutions. This can either lead to exploration of new regions between two local optima or faster local convergence if both solutions reside in the region of the same optimum. Particular implementations of this may not be universal as defining a logical "middle" between different types of design variables will differ.

- **Directional Search:** Carries out moves according to identified productive directions of search. Implementation can be guided by gradients, known locations of high fitness solutions, or estimates of the derivatives. This can lead to faster convergence, but, if not scaled properly for the size and complexity of the problem, can lead to premature convergence.

- **Variable Neighborhood Search:** Explores different neighborhoods around the current search point. The definition of different neighborhoods can vary from using moves of variable size to fixed definitions of search neighborhoods. This can help reduce local trapping and increase search efficiency in large, complex problems.

- **Search Space Mapping:** Constructs a map or partitions on the search space to guide the search process. This can aid in the efficiency of convergence but can require knowledge of the problem to devise the correct mapping or partitioning.

It is important to note that there is some overlap in the definitions of these strategies, and a particular algorithmic implementation may be categorized by more than one heuristic approach. This can be especially true when the algorithms employed were developed without explicit thought to the heuristics employed, which is a common feature of many of the existing metaheuristic algorithmic implementations in the literature. For clarity, since many specific algorithmic implementations were developed by modifying existing concepts within the literature, all possible heuristic classifications will be listed for each, with the primary listed first, when describing the algorithm's heuristics in Section III.
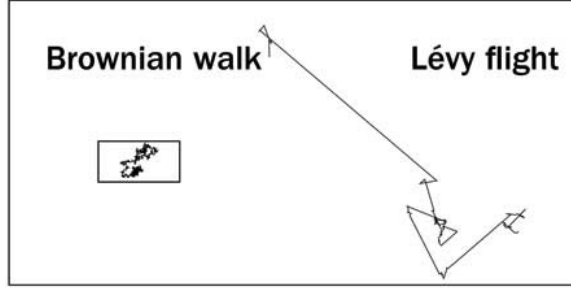
Fig. 1. Comparison of equal length paths for a Lévy flight and a normally distributed Brownian walk [29].

II.C Lévy Flights

Metaheuristic implementations are often stochastic, with a preponderance of implementations sampling from variations of uniform or normally distributed processes. As many algorithms are built on metaphors of physical, nature-inspired processes, this is a convenient choice as there are many naturally occurring phenomena that follow a normal distribution. An alternative is the Lévy distribution, which also has a basis in animal foraging behavior [26, 27, 28].

The Lévy distribution is given by

$$L_{\alpha,\gamma}(z) = \frac{1}{\pi} \int\limits_{0}^{+\infty} e^{-\gamma q^{\alpha}} \cos(qz) dq, \tag{3}$$

where α defines the index of the distribution and γ selects the scale of the process. Lévy distributions are heavy-tailed probability distributions where the wings are characterized by a power law behavior. This enables a more efficient global search of the phase space to be conducted through a higher probability of medium and large step sizes. Not only does the Lévy flight cover more of the phase space, it also avoids revisiting the same design point multiple times. This is shown graphically in Figure 1 for a Lévy flight and Brownian walk of the same path length. This feature is highly desirable in large, complex design spaces and allows for efficient exploration while limiting trapping in local optima, and Lévy Flights were first implemented as the basis for the random walk processes in the Cuckoo Search (CS) optimization algorithm and its variants [8].

Several algorithms have been developed to draw a stochastic sample with the correct probability density from a Lévy stable distribution. In this work, we chose the Mantegna algorithm for its speed and accuracy in sampling a Lévy distribution [26, 30]. We show the key steps for the

original, full algorithm to sample from a Lévy distribution, as some implementations vary in the literature [8, 26]. First the stochastic variable, ν, is calculated by

$$\nu = \frac{x}{y^{\frac{1}{\alpha}}}, \tag{4}$$

where x and y are normally distributed stochastic variables with standard deviations σ_x and σ_y. In order to ensure the stochastic process converges to the true distribution at both the origin and tails for arbitrary values of α, σ_x and σ_y cannot be chosen independently given there are two equations and three unknowns [31]. For this reason, $\sigma_y = 1$ is chosen and

$$\sigma_x(\alpha) = \left[\frac{\sigma_y \Gamma(1+\alpha)\sin(\pi\alpha/2)}{\Gamma\left(((1+\alpha)/2\right)\alpha 2^{(\alpha-1)/2}} \right]^{\frac{1}{\alpha}}. \tag{5}$$

A nonlinear transformation can be done to speed up the convergence to a Lévy stable process by

$$w = \nu \left[(K(\alpha) - 1)e^{(-\nu/C(\alpha))} + 1 \right], \tag{6}$$

where $K(\alpha)$ is given by

$$K(\alpha) = \frac{\alpha\Gamma((\alpha+1)/2\alpha)}{\Gamma(1/\alpha)} \left[\frac{\alpha\Gamma((\alpha+1)/2)}{\Gamma(1+\alpha)\sin(\pi\alpha/2)} \right]^{\frac{1}{\alpha}}, \tag{7}$$

and $C(\alpha)$ is obtained by numerically solving

$$\frac{1}{\pi\sigma_x} \int_0^\infty q^{1/\alpha} \exp\left[-\frac{q^2}{2} - \frac{q^{2/\alpha}C(\alpha)^2}{2\sigma_x^2(\alpha)} \right] dq$$
$$= \frac{1}{\pi} \int_0^\infty \cos\left[\left(\frac{K(\alpha)-1}{e} + 1 \right) C(\alpha) \right] e^{-q^\alpha} dq. \tag{8}$$

The stochastic Lévy sample is then given by

$$z = \gamma^{1/\alpha} \frac{1}{n^{1/\alpha}} \sum_{k=1}^n w_k, \tag{9}$$

where n is the number of independent stochastic variables.

III   PROPOSED ALGORITHM

In this section, the key features of the Gnowee algorithm are introduced and described. An overview of the Gnowee algorithm's framework is presented, followed by details of each of the initialization and update operators. Each operator describes the specific implementation in Gnowee, the heuristics used by the operator, and any Gnowee-specific variables that can be modified by the user.

III.A   Algorithm Framework

The general algorithm for Gnowee[a] is presented in Algorithm 1. Operators of the algorithm have been simplified in Algorithm 1 for conciseness, and each operator is described in detail in the following sections. Gnowee operators implement the neighborhood search, hill climbing, accepting negative moves, multi-start, adaptive memory programming, population based search, intermediate search, directional search, and variable neighborhood search heuristics. Search space mapping was not implemented as it generally required discretization of the phase space. A balance was struck between directional search, which can increase the rate of convergence, and accepting negative moves, which can increase the global search capabilities, to accomplish a fast, nearly global convergence within the design criterion described more fully in Section IV.

III.B   Operators

In this section, each of the major operators in the Gnowee algorithm shown in Algorithm 1 is described. For each operator, the basis in the literature, the specific heuristics employed, the Gnowee-specific implementation, and the associated Gnowee user-selected variables are described. Recommended values for these variables are shown in Section IV.B. For the combinatorial operators, a generic, generally applicable approach is described, and areas where slight modifications can be made based on specific optimization applications are noted.

---

[a]Open source and available from `https://github.com/SlaybaughLab/Gnowee`. Unless otherwise state, the URLs in this paper are current as of January 24th, 2018.

Algorithm 1: Gnowee Algorithm

    Input : User defined objective function, f; constraints, g and h; design space, $\vec{x}$; and algorithm settings (see Table I)

1 begin

2 $P.\vec{x} \leftarrow \text{initialization}(n)$ `// P is the parent population of size n`

3 $P.fit \leftarrow \text{population\_update}(P.\vec{x})$ `// fit is the assessed fitness`

4   **while** *convergence criterion is not met* **do**

5 $C.\vec{x} \leftarrow \text{three\_opt}(P.\vec{x_x})$

6 $P.fit \leftarrow \text{population\_update}(C.\vec{x})$

7

8 $C.\vec{x_c} \leftarrow \text{cont\_lévy\_flight}(P.\vec{x_c})$ `// C is the child population and` $\vec{x_c}$ `is the subset of the design vector containing continuous variables`

9 $C.\vec{x_d} \leftarrow \text{disc\_lévy\_flight}(P.\vec{x_d})$ `//` $\vec{x_d}$ `is the subset of the design vector containing discrete, integer, and binary variables`

10 $C.\vec{x_x} \leftarrow \text{comb\_lévy\_flight}(P.\vec{x_x})$ `//` $\vec{x_x}$ `is the subset of the design vector containing combinatorial variables`

11 $P.fit \leftarrow \text{population\_update}(C.\vec{x}, mh)$ `// mh indicates a Metropolis-Hastings algorithm is to be used`

12

13 $C.\vec{x} \leftarrow \text{crossover}(P.\vec{x_c}, P.\vec{x_d})$

14 $P.fit \leftarrow \text{population\_update}(C.\vec{x})$

15

16 $C.\vec{x} \leftarrow \text{scatter\_search}(P.\vec{x_c}, P.\vec{x_d})$

17 $P.fit \leftarrow \text{population\_update}(C.\vec{x})$

18

19 $C.\vec{x} \leftarrow \text{mutation}(P.\vec{x_c}, P.\vec{x_d})$

20 $P.fit \leftarrow \text{population\_update}(C.\vec{x})$

21

22 $C.\vec{x} \leftarrow \text{inversion\_crossover}(P.\vec{x_c}, P.\vec{x_d}, P.\vec{x_x})$

23 $P.fit \leftarrow \text{population\_update}(C.\vec{x})$

24

25 $C.\vec{x} \leftarrow \text{two\_opt}(P.\vec{x_x})$

26 $P.fit \leftarrow \text{population\_update}(C.\vec{x})$

III.B.1 Initialization

The Gnowee algorithm has flexible user selected initialization through random draws from a uniform distribution, nearly orthogonal Latin hypercube (NOLH), NOLH with random permutation, NOLH with Cioppa and Da Rainville permutations, and Latin hypercube (LHC) [32, 33, 34]. The use of LHC and NOLH sampling techniques increases the diversity of initial solution sets by better covering the design space. Increased diversity in the population improves global search while minimizing premature local convergence. This can dramatically improve early time (i.e. few function evaluations) convergence rates, but at late time, only minor differences were noted.

However, improvements in the early time convergence can be useful for the high cost objective function evaluation applications for which Gnowee was designed, and LHC was selected as the default initialization method.

The efficiency of the LHC algorithm to sample the design space is dictated by the number of population members. Larger populations provide a more robust sampling, but are cumbersome to carry throughout the optimization process due to the increased number of function evaluations performed on "dead weight" members (see Section IV.B for further discussion on the selection of the population size). Here, a compromise is struck to allow for robust LHC sampling by generating a number of LHC samples equal to the greater of twice the population size or three times the length of the design vector. After evaluating the fitness of the candidate designs, only the best p designs are carried forward into the optimization process.

III.B.2   3-opt

The 3-opt is in the family of heuristics introduced by Lin and Kernighan that has been applied to a wide variety of different algorithms [35]. For the Gnowee algorithm, the 3-opt heuristic is ideal as it is largely problem independent—requiring minimal, if any, knowledge of the problem—and can be implemented rather generically. Three-opt implements the neighborhood search, adaptive memory programming, hill climbing, and population-based search heuristics. To explain 3-opt, consider the following non-looping parent sequence:

$$P_1 = \boxed{\text{A}\,|\,\text{H}\,|\,\text{B}\,|\,\text{D}\,|\,\text{G}\,|\,\text{F}\,|\,\text{C}\,|\,\text{E}}$$

Three random, unique break points are selected from within the parent. For this example, the selected break points are H, G, and C. To simplify the diagram, consider the following sequence:

$$P_1 = \boxed{\text{S1}\,|\,\text{S2}\,|\,\text{S3}\,|\,\text{S4}}$$

where $S1$ is the sub-sequence given from A to the first break point $(A-H)$, $S2$ is the sub-sequence given from the point following the first break point, B, through the second break point $(B-D-G)$, $S3$ is the sub-sequence given from the point following the second break point, F, through the third break point $(F-C)$, and $S4$ is the renaming sub-sequence (E). Gnowee considers two re-orderings of the parent sequence around the three chosen break points. The first child, C_1, is defined as

$$C_1 = \boxed{\text{S1}\,|\,\text{S3}\,|\,\text{S2}\,|\,\text{S4}}$$

If the fitness of C_1 improves upon the fitness of P_1, then C_1 replaces P_1 in the parent population. The second permutation is then determined to be

$$C_2 = \boxed{\text{S1}\,|\,\text{S2}^{-1}\,|\,\text{S3}^{-1}\,|\,\text{S4}}$$

where $S2^{-1}$ and $S3^{-1}$ are the reverse of the respective original sub-segments; *e.g.*, $S2 = B - D - G$ and $S2^{-1} = G - D - B$. If C_2 improves the fitness, C_2 replaces P_1 in the parent population. This continues for each parent in the initial population.

III.B.3   Continuous Lévy Flight

Lévy flights are used to develop Markov chains that sample the design space. Lévy flights have primarily been employed in CS algorithms, but stochastic sampling of the search space is a commonly employed heuristic [10]. Lévy flights implement the variable neighborhood search, adaptive memory programming, hill climbing, directional search, accepting negative moves, multi-start, and population-based search heuristics described in Section II.B. Lévy flights on continuous variables are performed on a fraction of parents, f_l, as

$$\vec{x}_r^{\,g+1} = \vec{x}_r^{\,g} + \frac{1}{\beta}\vec{L}_{\alpha,\gamma}, \tag{10}$$

where g is the generation number, r is a unique random index, $r \in \{1, 2, \ldots, p\}$, β is a step size scaling factor, and the procedure to determine $\vec{L}_{\alpha,\gamma}$ was described in Equation 4 - Equation 9. Including β is typically necessary to avoid consistently taking steps that are large relative to the dimensions of the problem, which can result in oscillatory behavior. The recommended value for β reported in the literature varies from $\beta = 1$ to $\beta = L/100$ where L is the characteristic scale of the problem being considered [8, 10, 36]. This can be problematic when the scales vary significantly for different variables of the design vector, a fact often ignored.

Instead of scaling β to the characteristic scale of the problem, the Gnowee algorithm takes a slightly different approach. The value of the optimal β under this approach was determined using a hyper-optimization technique described in Section IV.B.

For each Lévy flight, the updated design vector is calculated using Equation 10. Next, a boundary rejection and re-sampling algorithm is implemented to check the resulting design vector's validity. If the step returns a valid result within the boundary of the problem for a given variable

of the design vector, the value is accepted and no further action is taken. However, if that step is outside the problem boundary for a given design variable, that solution is rejected and a new $L_{\alpha,\gamma}$ is generated. This method showed drastic improvement in decreasing the sensitivity of the optimization performance across the range of benchmarks considered with changes in β, allowing a constant value to be chosen without any impact on the algorithm's performance.

Finally, within the Lévy flights heuristic, a form of the Metropolis-Hastings rejection sampling algorithm is implemented [37]. All children that improve upon the fitness of the parent are automatically accepted. A fraction, $f_m h$, of the children that would normally be discarded due to lower fitness than their parent are instead compared against another random parent in the population. If the fitness of the child is an improvement over the random parent, the child replaces that parent. This technique works well when coupled with Lévy flights. For small steps, accepting a fraction of the solutions that would normally be discarded increases the local convergence rate through the directional search heuristic. For large steps, this can increase the global exploration through the acceptance of negative moves and multi-start heuristics. However, even with Lévy flights, care must be taken not to set $f_m h$ too large or the population diversity will decrease and result in premature convergence.

III.B.4 Discrete Lévy Flight

Lévy flights on discrete, integer, and binary variables are performed as described in Equation 10, but $L_{\alpha,\gamma}$ is calculated from

$$L_{\alpha,\gamma} = ROUND(TLF_{\alpha,\gamma} \times D(x)), \tag{11}$$

where $TLF_{\alpha,\gamma}$ is a truncated Lévy flight on the interval $[0,1]$ and D is a variable describing the scale of the discrete variable, x, being considered [31]. In this way, the TLF can be used to map a Lévy distribution onto a discrete variable by eliminating arbitrarily large steps. D can be kept general, such as the relative current location indexes, or problem-specific knowledge relevant to the objective function can be leveraged.

For the Gnowee algorithm, one child is generated per $f_l \times p$ parents. The Metropolis-Hasting algorithm is employed to accept a fraction of the children that do not improve upon their parent's fitness.

III.B.5   Combinatorial Lévy Flight

Combinatorial Lévy Flight combines Equation 11 from Section III.B.4 with an inversion operator. Inversion operators are common in Genetic Algorithm (GA) implementations, and they have been adopted by other algorithms such as Cuckoo Search (CS) [20, 36, 38]. Combinatorial Lévy Flight implements the neighborhood search, hill climbing, adaptive memory programming, directional search, accepting negative moves, multi-start, and population-based search heuristics described in Section II.B by creating new solutions through inverting a portion of the original solution set. For example, consider the parent sequence below:

$$P_1 = \boxed{\text{A} \mid \text{H} \mid \text{B} \mid \text{D} \mid \text{G} \mid \text{F} \mid \text{C} \mid \text{E}}$$

The inversion points can be selected randomly, based on current location indexes, or by using a problem-specific quantity related to the minimization of the objective function such as distance. For example, if H and G are chosen as the cut points, the child would be:

$$C_1 = \boxed{\text{A} \mid \text{H} \mid \text{G} \mid \text{D} \mid \text{B} \mid \text{F} \mid \text{C} \mid \text{E}}$$

For the Gnowee algorithm, one child is generated per $f_l \times p$ parents. The Metropolis-Hasting algorithm is employed to accept a fraction of the children that do not improve upon their parent's fitness.

III.B.6   Crossover

Crossover is a common feature of GA and Differential Evolution (DE) algorithms implemented in a variety of manners [6, 20]. Variations of the crossover concept have been applied in Particle Swarm Optimization (PSO) and CS algorithms as well [39, 40]. For Gnowee, a version of the implementation adopted by Walton is employed for continuous, integer, and discrete variables, where the child solution is calculated as [40]

$$\vec{x}_r^{\,g+1} = \vec{x}_0^{\,g} + \frac{(\vec{x}_0^{\,g} - \vec{x}_r^{\,g})}{\Phi}, \tag{12}$$

where r is a unique random index, $r \in \{1, 2, \ldots, f_e \times n\}$, f_e corresponds to the fraction of parents in the elite subset of high fitness designs, and Φ is the golden ration given by $\Phi = (1 + \sqrt{5})/2$. The golden ratio is used as it has been found to increase convergence over other choices in the range of

$[0, 2]$ for similar strategies [6, 40].

As implemented, this method executes the intermediate search, adaptive memory programming, hill climbing, directional search, and population-based search heuristics described in Section II.B. The addition of intermediate and directional search through elitism increases overall population fitness and allows for rapid search of promising localities, but can lead to premature convergence due to the loss of population diversity if f_e is not chosen properly.

III.B.7 Scatter Search

Gnowee uses an adaptation of scatter search pioneered by Egea et al. [41]. The scatter search approach leverages the information of the population to build variable search spaces based on the relative fitness of selected population members. In Gnowee, the scatter search heuristic is applied to continuous, integer, binary, and discrete variables. Scatter search implements intermediate search, variable neighborhood search, adaptive memory programming, hill climbing, and population-based search heuristics described in Section II.B. The scatter search heuristic updates the design vector according to

$$\vec{x}^{g+1} = \vec{c}_1 + (\vec{c}_2 - \vec{c}_1)\vec{r}, \tag{13}$$

where $\vec{x}^{g+1}$ is the updated design vector and $\vec{r}$ is a vector of uniformly distributed random variables. $\vec{c_1}$ and $\vec{c_2}$ are given by

$$\vec{c_1} = \vec{x_i^g} - \vec{d}(1 + \alpha\beta), \tag{14}$$

$$\vec{c_2} = \vec{x_i^g} - \vec{d}(1 - \alpha\beta), \tag{15}$$

where $\vec{x_i^g}$ is a current member of the elite sub-population. $\vec{d}$ is given by

$$\vec{d} = \frac{\vec{x_j^g} - \vec{x_i^g}}{2}, \tag{16}$$

where $\vec{x_j^g}$ is a randomly chosen member of the population. From Equation 14 and Equation 15, α and β are calculated as

$$\alpha = \begin{cases} 1, & \text{if } i < j \\ -1, & \text{otherwise.} \end{cases} \tag{17}$$

$$\beta = \frac{|j - i| - 1}{p - 2}, \tag{18}$$

where p is the size of the population.

This process is continued for $f_e \times p$ parents. If the generated children have a better assessed fitness than their parents, they are accepted into the population and replace their parents.

III.B.8   Mutation

Mutation is another foundational search heuristic of GA and DE algorithms that has been modified and adopted by a wide variety of other approaches [6, 10, 42]. The Gnowee algorithm implements mutations for continuous, integer, binary, and discrete variables. The mutation method implements intermediate search, variable neighborhood search, adaptive memory programming, hill climbing, and population-based search heuristics described in Section II.B. The mutation method employed is calculated as

$$\boldsymbol{X}^{g+1} = \boldsymbol{X}^g + r\boldsymbol{D}(\boldsymbol{P}_1 - \boldsymbol{P}_2), \tag{19}$$

where $\boldsymbol{X}$ is the population solution vectors, $\vec{x}$, r is a uniformly distributed random variable, $\boldsymbol{D}$ is a matrix of 0s and 1s where each value is determined through a random draw from a uniform distribution with 0 obtained if $< f_m$ and 1 otherwise, and $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are random permutations of the original $\boldsymbol{X}$.

III.B.9   Inversion and Crossover

Inversion and crossover are used in GA and DE algorithms to copy portions of one parent into another to create a unique child [6, 20, 42]. When combined with elitism, crossover can be used to copy traits of high fitness parents into the population. In Gnowee, crossover with elitism for continuous, integer, binary, discrete, and combinatorial variables implements the intermediate search, adaptive memory programming, hill climbing, directional search, and population-based

search heuristics described in Section II.B. Gnowee uses random research to determine the sub-segment to crossover in order to allow for general applicability. This can be kept general, such as the relative current location indexes, or problem-specific knowledge relevant to the objective function can be leveraged [20, 36].

Consider the following sequences: P_1, chosen randomly from elite sub-population, and P_2, a unique, randomly-chosen parent from the entire population:

$$P_1 = \boxed{\text{A} \mid \text{H} \mid \text{G} \mid \text{D} \mid \text{B} \mid \text{F} \mid \text{C} \mid \text{E}}$$

$$P_2 = \boxed{\text{E} \mid \text{A} \mid \text{G} \mid \text{C} \mid \text{H} \mid \text{B} \mid \text{D} \mid \text{F}}$$

Point H is chosen randomly as the first inversion point from P_1. The second inversion point is determined by the location that follows H in P_2, which is B in this example. After inversion, the child sequence is

$$C_1 = \boxed{\text{A} \mid \text{H} \mid \text{B} \mid \text{D} \mid \text{G} \mid \text{F} \mid \text{C} \mid \text{E}}$$

Only children that improve upon their parent's fitness are accepted. Next, the previous second inversion point, B, is taken as the starting inversion point in P_2, and the new second inversion point is determined from the location that follows B in P_1, which is F. The new child sequence generated is

$$C_2 = \boxed{\text{E} \mid \text{A} \mid \text{G} \mid \text{C} \mid \text{H} \mid \text{B} \mid \text{F} \mid \text{D}}$$

If C_2 improves upon the fitness of P_2, then the child replaces the parent in the population. This process is continued over each location in the elite parent for $f_e \times p$ elite parents.

III.B.10   2-opt

Two-opt is one of a family of heuristics introduced by Lin and Kernighan that has been applied to a wide variety of different algorithms [35]. The Gnowee algorithm adopts an implementation similar to that described by Zhou by adding Lévy flights and elitism to the selection process [36]. As implemented, 2-opt employs the neighborhood search, adaptive memory programming, hill climbing, directional search, and population-based search heuristics.

To describe 2-opt, consider the following parent sequence

$$P_1 = \boxed{\text{A} \mid \text{H} \mid \text{B} \mid \text{D} \mid \text{G} \mid \text{F} \mid \text{C} \mid \text{E}}$$

The first break point is chosen as A. The second break point, G for this example, is chosen from a TLF mapped onto the length of the sequence. The re-connection is made where the first break point connects to the second break point and inverts the sequence between the two points. The point originally following the first break point, H, then connects to the point originally following the second break point, F as shown in the sequence

$$C_1 = \boxed{\text{A} \mid \text{G} \mid \text{D} \mid \text{B} \mid \text{H} \mid \text{F} \mid \text{C} \mid \text{E}}$$

If the child improves upon the parent's fitness, then the solution is accepted. The starting break point then cycles through the list up to the nth item in the sequence (or $n-2$ for non-looping sequence) for each parent in the elite subset given by the top $f_e \times p$ parents.

III.B.11   Population Update

The descriptions of the individual operators describe the population update procedures used, but it is worth expounding upon this process. There are a few typical update strategies. One common approach is to select p new parents from the generated c children plus p existing parents. In this approach, the p best solutions are taken regardless of their origin, which can result in faster convergence at the expense of reduced diversity with a corresponding increased chance of premature local convergence. An alternative approach is to select the new p population members from the c children. This approach increases diversity and limits premature convergence, but may require more function evaluations as a large number of negative steps can be accepted.

The approach adopted by Gnowee attempts to strike a balance by *mostly* only allowing children to replace their parents. This keeps diversity high and avoids negative moves. As noted, however, elitism based heuristics, while serving to increase convergence, can also themselves decrease the diversity of the population. This risk is offset by employing the Metropolis-Hastings algorithm once per generation to allow for negative moves that, due to the nature of Lévy flights, can increase the diversity of the population.

IV   A STUDY OF CHARACTERISTICS OF GNOWEE

Gnowee was designed for rapid convergence of nearly globally optimum solutions for complex engineering problems with mixed-integer and combinatorial design vectors and high-cost, noisy, discontinuous, black box objective function evaluations. To demonstrate the performance of this

new algorithm, we compared Gnowee to many existing algorithms applied to two types of problems sets: (1) mathematical tests that have solution spaces that are challenging for optimization in different ways and (2) example engineering problems. Note that there are no completely comparable algorithms against which to test. Continuous, mixed-integer, and combinatorial implementations of several common metaheuristic algorithms, described in Section IV.A, were chosen to compare against as these algorithms best capture the behavior exhibited by Gnowee. The algorithm settings and convergence criteria used are described in Section IV.B. The common benchmark problems from each class of variables are described in Section IV.C, Section IV.D, and Section IV.E.

Another class of complex, constrained nuclear engineering optimization problems involves parameter optimization or fitting to obtain a model that best matches a measurement. These problems are slightly different from those used to benchmark Gnowee in this paper in that there is uncertainty in the cost function arising from measurement uncertainties. However, Gnowee can be applied to these types of problems as well if the uncertainty can be captured in the objective function (i.e. chi-square minimization), but Gnowee is not specifically tested against this class of problems.

IV.A   Benchmark Comparison Algorithms

Several well-established, openly-available algorithms were chosen to benchmark the performance of Gnowee. This section describes the algorithms chosen, the origin of the specific implementation chosen, and any modifications made to the original algorithm.

It is beyond the scope of this paper to describe each algorithm implemented for benchmarking in detail here; descriptions of each algorithm are available in the corresponding references provided below. All algorithms, with the exception of Discrete Cuckoo Search (DCS), a were adopted directly from their authors with modifications only to the convergence criteria, described in Section IV.B, and output information. References for the source location of each algorithm are given below. The algorithms chosen for benchmarking represent a diverse set of approaches with both unique and sometimes overlapping metaheuristic characteristics. We do not claim the implementations chosen represent the best possible implementation of a given algorithm. Instead, preference is given to well characterized algorithms available for use in engineering design problems.

For comparison against continuous vector algorithms, GA, simulated annealing (SA), PSO,

CS, MCS, and MEIGO were selected. GA, SA, and PSO were implemented using the Global Optimization Toolbox in Matlab® R2015b [43]. A Matlab® implementation of Yang's CS[b] and Walton's MCS[c] algorithms were obtained through the Mathworks® File and Link exchanges [8, 40]. The Matlab® implementation of MEIGO[d], openly available from the author's website, was used [18].

For comparison against mixed-integer vectors, GA and MEIGO were chosen. GA was implemented using the Global Optimization Toolbox in Matlab® R2015b [43]. A Matlab® implementation of MEIGO was obtained through the MEIGO website [18].

Finally, for combinatorial vector algorithms, Discrete GA and DCS were chosen. The Discrete GA algorithm for TSP was obtained through the Mathworks® File exchange[e]. No available implementation of the DCS algorithm was found, so the authors implemented their best interpretation of the algorithm by Zhou [36, 44]. Results from this implementation are consistent with those reported by Zhou [36].

IV.B   Discussion of Key Model Parameters

Each of the considered comparison algorithms has many possible user directed modifications that can be made through adjustment of algorithm parameters. For this study, all non-convergence related model parameters for the comparison algorithms are set to the defaults recommended by that software package and/or author [8, 18, 36, 40, 43].

The convergence criteria were set to ensure fair comparisons were made among the algorithms. The maximum number of function evaluations, F_{max}^{eval}, is capped at 200,000 for all algorithms. The stall convergence criteria is set at 1E-6 with the maximum number of stall evaluations, F_{stall}^{eval}, of 10,000. The fitness convergence is set at 1% of the known optimal fitness. Finally, because the solution process is stochastic, each problem was run 100 times to get the representative mean and standard deviation behavior. No attempt was made to control for or quantify run time

---

[b]Available from https://www.mathworks.com/matlabcentral/fileexchange/29809-cuckoo-search--cs--algorithm

[c]Available from https://www.mathworks.com/matlabcentral/linkexchange/links/2999-modified-cuckoo-search-mcs-open-source-gradient-free-optimiser

[d]http://www.iim.csic.es/~gingproc/meigo.html

[e]Available from https://www.mathworks.com/matlabcentral/fileexchange/13680-traveling-salesman-problem-genetic-algorithm

TABLE I
Gnowee algorithm settings.

Parameter	Description	Gnowee Value
P	population	25
S_i	population initialization	LHC
α	Lévy index	0.5
γ	Lévy scale	1
β	step size	10
f_1	Lévy fraction	1.0
f_m	mutation fraction	0.2
f_e	elite fraction	0.2

as the intended applications will have high objective function evaluation costs that dwarf the metaheuristic framework costs.

The Gnowee program-specific settings are shown in Table I. The parameters were determined from the set of benchmarks using iterative parametric studies. True hyper-optimization techniques were not used as most of the parameters had a large, stable minimum region making this approach sufficient.

For brevity, only the results from the selection of the population size, p, for Gnowee are presented in Figure 2 as it is the most sensitive and complex parameter. Full results for all parameters are available at `https://github.com/SlaybaughLab/Gnowee/tree/master/Benchmarks`. As shown in Figure 2, each class of problem, described further in Section IV.C, could have a different optimal population size, but 25 was chosen as it maximized benefits for the largest number of considered benchmark problems while minimizing the loss of performance for the remaining problems.

IV.C   Continuous Benchmark Problems

Two sets of continuous benchmark problems were chosen. The complete description of each problem is not presented, but the mathematical formulation of the objective function, constraints, and the optimal fitness is shown for an example problem from each set. The optimal fitness is the currently best known solution given the objective function, parameter search space, and problem constraints. For continuous unconstrained functions, this is often zero, but in practice it is often

non-zero as constraints play a role and the objective function is measuring a real quantity (such as cost). References to literature describing the benchmark problem, objective function, and defining parameters are provided for each of the remaining benchmarks.

The first set consists of four continuous, constrained engineering problems that are well described in the literature: optimization of a welded beam [45], pressure vessel [45, 46], speed reducer [45, 46], and spring [45, 10]. The pressure vessel design problem is stated as

Minimize:
$$f(\vec{x}) = 0.6224RLt_s + 1.7781R^2t_h + 3.1611Lt_s^2$$
$$+ 19.8621Rt_h^2$$

Subject to:
$$g_1(\vec{x}) = -t_s + 0.01932R \leq 0$$
$$g_2(\vec{x}) = -t_h + 0.00954R \leq 0$$
$$g_3(\vec{x}) = -\pi R^2 L - \frac{4}{3}\pi R^3 750 \times 1728 \leq 0$$
$$g_4(\vec{x}) = -240 + L$$

where $\vec{x}$, the upper, and the lower bounds are given by

$$\vec{x} = [R, L, t_s, t_h]$$
$$\vec{x}_{lb} = \left[10, 1 \times 10^{-8}, 0.0625, 0.0625\right]$$
$$\vec{x}_{ub} = [50, 200, 6.1875, 6.1875]$$

and R is the pressure vessel inner radius, L is the length of the pressure vessel, t_s is the thickness of the pressure vessel shell, and t_h is the head thickness. All variables are treated as continuous.

The second set consists of six continuous, unconstrained functions that are well described in the literature as benchmarks: Ackley, De Jong, Easom, Griewank, Rastrigin, and Rosenbrock [10, 43, 40]. Ackley's function is a multi-modal, n-dimensional function with a global minimum of $f_{opt} = 0$. It is shown in Figure 3 for $n = 2$ and given by

$$f(\vec{x}) = -20exp\left[-0.2\sqrt{\frac{1}{d}\Sigma_{i=1}^d x_i^2}\right] - exp\left[\frac{1}{d}\Sigma_{i=1}^d cos(2\pi x_i)\right]$$
$$+ (20 + e) \qquad\qquad x_i \in [-32.768, 32.768].$$

IV.D   Mixed-Integer Benchmark Problems

Mixed-integer optimization problems combine continuous, binary, integer, and/or discrete variables. Here, three engineering design benchmarks are considered: MI spring [47], MI pressure vessel [45], and a MI chemical process design [19]. For brevity, only the pressure vessel is described in detail. The remaining benchmarks are described in detail in the literature references.

The MI pressure vessel design is similar to the continuous pressure vessel design specified in Section IV.C with the exception that two variables, t_s and t_h, can only take discretized values. The $\vec{x}$, the upper, and the lower bounds are the same as specified in Section IV.C, but for the mixed-integer case, t_s and t_h can only take values that are discrete multiples of 0.0625.

IV.E   Combinatorial Benchmark Problems

The traveling salesman problem (TSP) was chosen for the combinatorial optimization benchmarks. TSP problems were drawn from TSPLIB[f] and were chosen to span a range of city lengths without being excessively large given the intended engineering design applications. For this study, the Eil51, St70, Pr107, Bier127, and Ch150 TSP problems were chosen.

V   COMPUTATIONAL RESULTS AND COMPARISONS

Benchmarks were carried out as described in Section IV for continuous constrained, continuous unconstrained, mixed-integer, and combinatorial (TSP) benchmark problems. The results presented here highlight the optimization performance of Gnowee against the eight algorithms described in Section IV.A. For brevity, only a subset of the results is included; the full results set is available from `https://github.com/SlaybaughLab/Gnowee/tree/master/Benchmarks/results`. Here, specific results are presented for the continuous constrained spring optimization to serve as an example, and summary results are presented for the remaining problems.

For Gnowee, success is measured by consistent, rapid convergence to nearly globally optimum solutions. To benchmark in a method consistent with the Gnowee design objectives, convergence criteria were set to a "limited" number of function evaluations (200,000), a threshold of 1% of the best known optimal fitness, or a "limited" function evaluation stall limit (10,000). A stall is

---

[f]Available from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

defined as when the algorithm failed to find an improved solution, and the stall limit is the number of function evaluations that can be performed without improvement before the algorithm exits.

To quantitatively capture the performance of a given algorithm to facilitate comparison of algorithm performance, a figure of merit (FOM) is defined as

$$FOM = \frac{f_{avg}(\vec{x}) - f_{opt}}{f_{opt}} \left(N_{f(\vec{x})}^{avg} + 3\sigma_N \right) . \tag{20}$$

where $f_{avg}(\vec{x})$ is the average fitness obtained, f_{opt} is the optimal fitness, $N_{f(\vec{x})}^{avg}$ is the average number of function evaluations performed, and σ_N is the standard deviation of the fitness solution distribution.

This formulation of the FOM emphasizes the convergence rate and width of the the functional evaluation distribution while also penalizing prematurely convergent algorithms for poor fitness solutions. Note that a smaller FOM value indicates better performance, so we are seeking the smallest FOM. The choice of using the 3σ number of function evaluations emphasizes the importance of minimizing the total number of function evaluations, an important factor for the high-cost objective function evaluation applications for which Gnowee was designed. In these applications, an optimization is often only performed a few times, perhaps even just once, making the edge cases more interesting than the average behavior of the algorithm.

In the results presented below, bolded values indicate premature convergence, where the fitness achieved is $> 1\%$ from the optimal solution. The italicized values indicate the best performing algorithm according to the FOM.

V.A Continuous Constrained Benchmark Results

The continuous benchmarks considered are subdivided into constrained and unconstrained classes for ease of comparison. First, the detailed results for the spring optimization problem are shown in Table II for the run conditions and convergence criteria specified in Section IV.B. Table II presents design variable values (x_*), fitness ($f(\vec{x})$), number of function evaluations ($N_{f(\vec{x})}$), and FOM for both the average and best performance obtained over the 100 optimization iterations performed by each algorithm. For the spring optimization, Gnowee outperformed the other algorithms on average by reducing the number and standard deviation of function evaluations required, while MEIGO and SA had the best single performance optimizations.
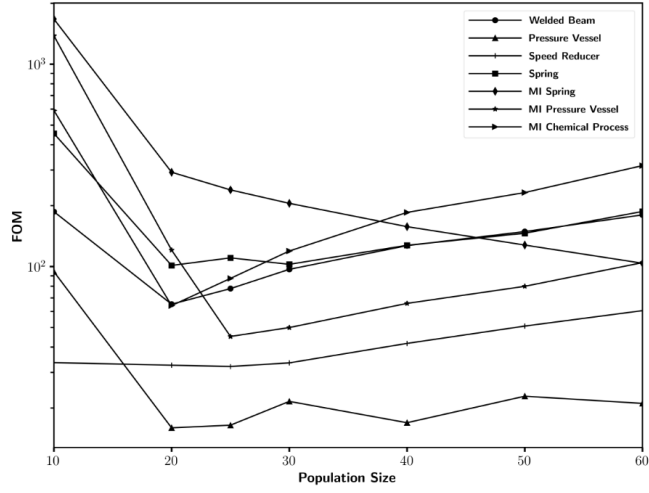
a) Mixed-Integer and Continuous Constrained
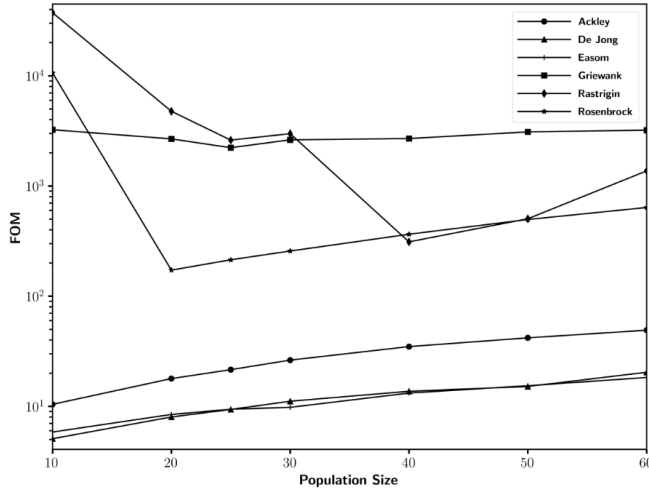
b) Continuous Unconstrained
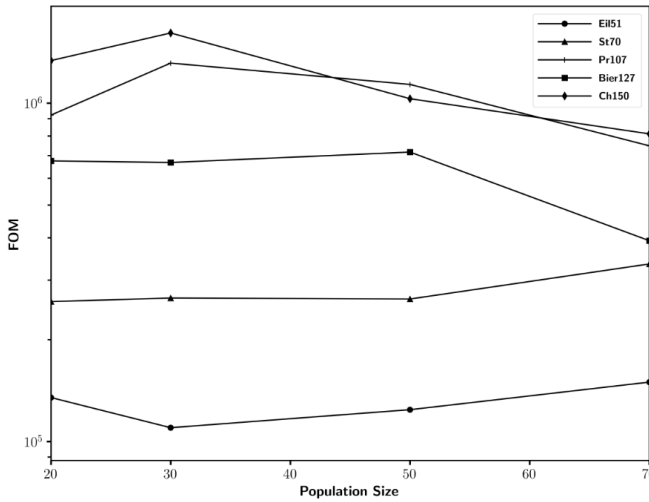
c) TSP

Fig. 2. Parametric hyper-optimization results for Gnowee population size. Each algorithm and benchmark were run for 100 iterations. The figure of merit (FOM), is defined in Equation 20.
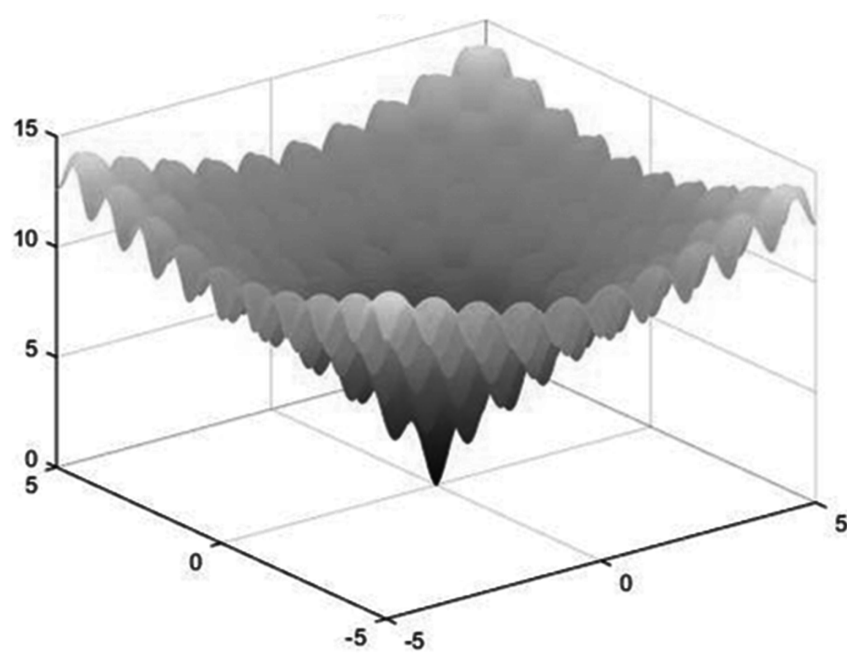
Fig. 3. Two-dimensional Ackley function.

TABLE II
Spring optimization results (Optimum fitness = 0.012665).

	GA[43]	SA[43]	PSO[43]	CS [10]	MCS[40]	MEIGO[18]	Gnowee
x_1^{avg}	**0.060063 $\pm$ 0.00558**	0.050810 $\pm$ 0.00090	**0.052298 $\pm$ 0.00314**	0.051703 $\pm$ 0.00127	**0.058462 $\pm$ 0.00592**	0.052817 $\pm$ 0.00133	*0.051620 $\pm$ 0.00119*
x_2^{avg}	**0.610416 $\pm$ 0.18392**	0.335680 $\pm$ 0.02133	**0.375903 $\pm$ 0.08398**	0.357026 $\pm$ 0.03064	**0.560899 $\pm$ 0.18946**	0.385194 $\pm$ 0.03386	*0.355059 $\pm$ 0.02874*
x_3^{avg}	**5.461663 $\pm$ 3.19803**	12.840958 $\pm$ 1.32526	**11.363353 $\pm$ 3.39514**	11.560822 $\pm$ 1.83294	**6.573904 $\pm$ 3.75091**	9.997362 $\pm$ 1.53760	*11.648626 $\pm$ 1.72111*
$f_{avg}(\vec{x})$	**0.014398 $\pm$ 0.00161**	0.012778 $\pm$ 0.00002	**0.012890 $\pm$ 0.00030**	0.012771 $\pm$ 0.00004	**0.014068 $\pm$ 0.00156**	0.012785 $\pm$ 0.00006	*0.012763 $\pm$ 0.00002*
$N_{f(\vec{x})}^{avg}$	**16947 $\pm$ 6379**	5820 $\pm$ 5085	**7959 $\pm$ 10523**	16948 $\pm$ 7174	**19013 $\pm$ 13684**	10279 $\pm$ 9835	*4738 $\pm$ 1836*
FOM_{avg}	**4938**	188.3	**702.9**	323.2	**6654**	376.7	*79.0*
x_1^{best}	0.051341	0.050318	0.052021	0.051230	0.051768	*0.052121*	0.051092
x_2^{best}	0.348240	0.324492	0.364483	0.345779	0.358488	*0.367169*	0.342205
x_3^{best}	11.878832	13.476806	10.886333	11.987921	11.224128	*10.703094*	12.210091
$f_{best}(\vec{x})$	0.012740	0.012715	0.012711	0.012694	0.012705	*0.012691*	0.012694
$N_{f(\vec{x})}^{best}$	7900	835	27900	5225	4328	*1090*	3813
FOM_{best}	47	3.3	100.3	12.1	13.5	*2.2*	8.7

TABLE III

Summary of FOM results for continuous constrained optimization benchmarks.

	Welded Beam [45]	Pressure Vessel [45]	Speed Reducer [45]	Spring [45]
GA[43]	**106667.6**	**762.0**	56.5	**4938**
SA[43]	**5125.8**	**8488.6**	**1437.1**	188.3
PSO[43]	329.4	**1209.5**	62.2	**702.9**
CS[10]	442.6	529.4	51.3	323.2
MCS[40]	**2933.8**	**1472.6**	26.0	**6654**
MEIGO[18]	*35.0*	**1567.4**	*11.7*	355.3
Gnowee	80.6	*108.4*	29.7	*79.0*

The summary of FOM results obtained for the continuous, constrained benchmark problems are shown in Table III. In Table II and Table III, bold results indicate fitness greater than 1% from the global optimum and italicized results indicate the best performance for the average and the overall best run. Across all of the benchmarks in this category, Gnowee had the best performance twice and never finished worse than third. Additionally, Gnowee did not suffer from premature convergence issues that affected all of the other algorithms except CS. Rapid yet continual convergence across a wide range of problems is a key feature of Gnowee enabled by the diverse set of hybrid heuristics.

One way to better visualize the results from Table III is to look at the histogram of converged solutions as shown in Figure 4 for SA and Gnowee. The histogram includes optimization results for 1000 iterations of the best average performer, Gnowee, and one of the top individual performers, SA. SA heavily relies on the directional search, neighborhood search, and hill climbing heuristics, resulting in rapid convergence to a *local* optimum. However, this comes at the expense of robust global search, often resulting in a solution that stalls short of the global, or in this case nearly global, optimum. Gnowee, on the other hand, does not achieve nearly as rapid initial convergence, but its convergence is more consistent due to the balancing of local and global search heuristics.

The effect on the balanced heuristics on the convergence can be shown throughout the complete optimization evolution as illustrated in Figure 5 for the spring benchmark. This figure shows the average fitness versus fixed function evaluation intervals for up to 10,000 function evaluations, illustrating some key points. First, while Gnowee is not the best algorithm at all fitness con-
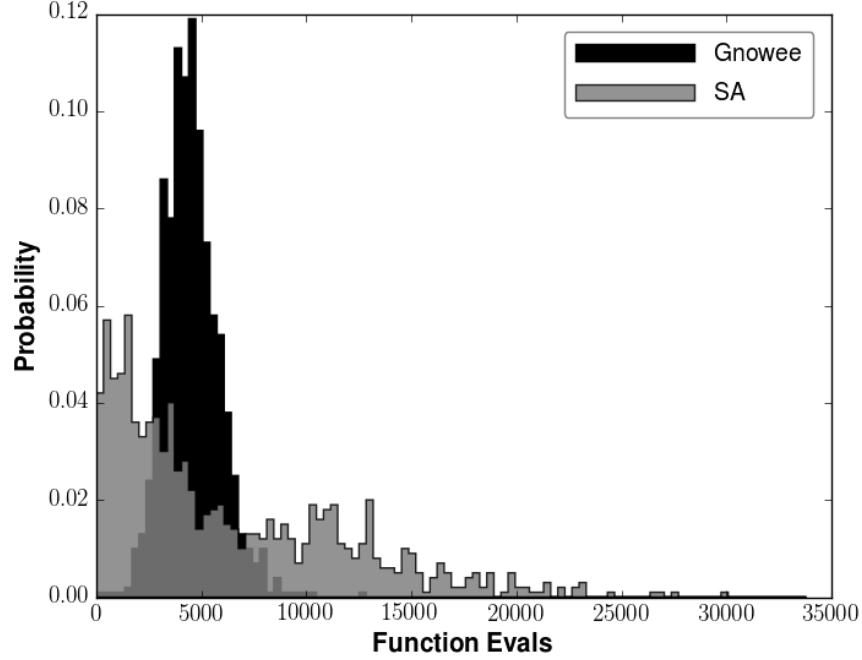
Fig. 4. Histogram of the number of function evaluations performed before meeting the convergence or stall criteria specified in Section IV.B.

vergence levels, it is competitive across the full range considered. Second, Gnowee maintains its convergence rate and continues to converge while other algorithms start to plateau. Finally, this highlights a potential area for future work. In this example, and across several others, Gnowee's initial solution is among the worst. Work to improve the initial starting solution could accelerate Gnowee's convergence and improve performance.

Finally, a look at the convergence history of the design parameters illustrates the importance of balancing local and global search. Convergence history is shown in Figure 6 for a single optimization of the spring benchmark using Gnowee. In Figure 6, each point represents an improvement in overall fitness (top subplot) and the corresponding values for each of the three design variables (bottom three subplots). When there are periods of gradual change in the design variables with improvements in fitness, the solution is improving via local search heuristics. When there are large changes in one or more design variables from one point to the next, the solution is improving via global search heuristics.

This combination of local and global improvement is the reason that Gnowee outperforms most other algorithms most of the time. Unlike GA, SA, PSO, or MCS, which tend to pre-
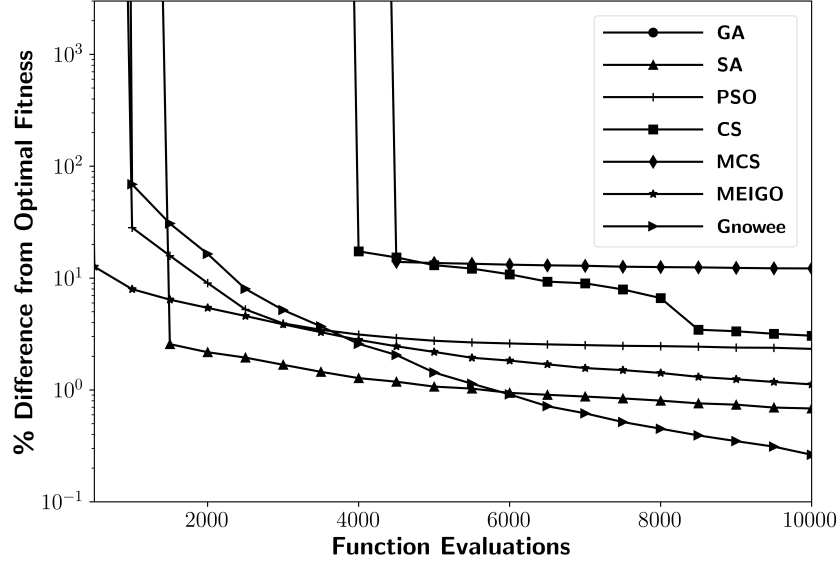
Fig. 5. Spring optimization fitness convergence. 1σ standard deviation in fitness is generally of comparable size or smaller than the plot markers.

maturely converge for some problems due to shifts towards local search with increasing genera-tion/evaluations, Gnowee maintains global search heuristics throughout the optimization as il-lustrated by the jumping between local optima throughout the optimization history shown in Figure 6.

V.B   Continuous Unconstrained Benchmark Results

The other subset of continuous benchmarks was the unconstrained functions described in Section IV.C. The summary of the FOM results obtained for these benchmark problems is shown in Table IV, where bold results indicate fitness greater than 1% from the global optimum and italicized results indicate the best performance for the average and the overall best run. Each column of the table indicates the benchmark function and the dimensionality considered. Here, Gnowee obtains the best performance for the Rastrigin and Rosenbrock functions and the second best performance for the remaining functions. Similar to the continuous constrained benchmark results shown in Section V.A, Gnowee maintains consistent continual convergence throughout the optimization process for all problems except for the Rastrigin function. For the easier problems, Gnowee's poor early time solutions slow the convergence process slightly, thereby hindering its performance when compared to the other algorithms.
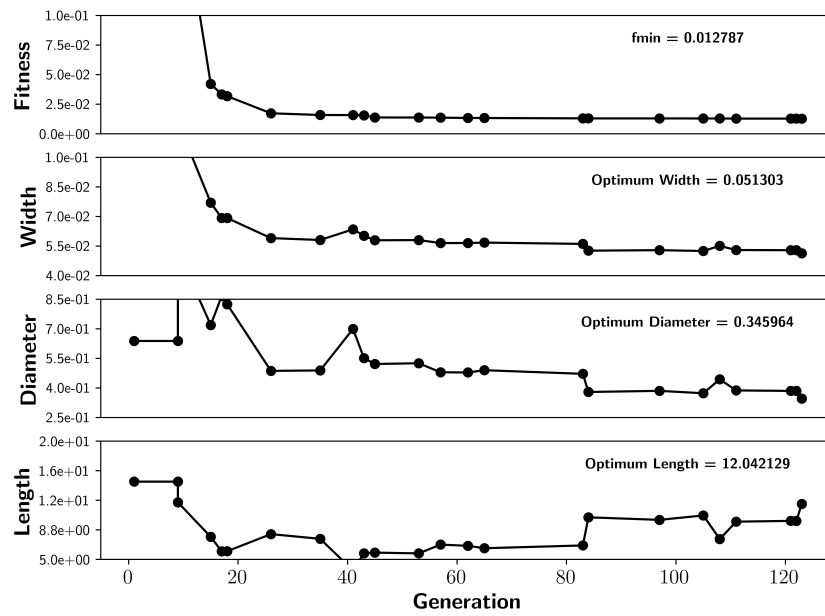
Fig. 6. Spring optimization history for a single Gnowee run showing the convergence of the fitness and each design parameter.

TABLE IV
Summary of FOM results for continuous unconstrained optimization benchmarks.

	Ackley (3-D)	De Jong (4-D)	Easom (2-D)	Griewank (6-D)	Rastrigin (5-D)	Rosenbrock (5-D)
GA[43]	**1666.8**	10.7	**6809.3**	*92.1*	**15843**	**181239**
SA[43]	109.7	53.7	**22415.2**	**27654.5**	**122362**	**12420**
PSO[43]	33.7	15.2	11.9	**3665.4**	**29436**	**25884**
CS[10]	85.9	30.5	58.8	**13956.8**	**274627**	**47472**
MCS[40]	**1955.5**	12.6	54.4	**16746.4**	**155804**	**143587**
MEIGO[18]	*12.5*	*3.5*	*3.2*	**2597.3**	**11514.8**	224.4
Gnowee	19.7	8.9	6.5	**1549.7**	***1753.0***	*152.2*

TABLE V

Summary of FOM results for constrained mixed-integer optimization benchmarks.

	Pressure Vessel [45]	Spring [47]	Chemical Process [19]
GA[43]	**897.9**	**1697.9**	**18890.1**
MEIGO[18]	**358.3**	330.1	95.2
Gnowee	*40.4*	*219.2*	*92.1*

The results in Table IV are useful for understanding the interplay of the well defined fitness landscape and the search heuristics used. For example, consider the Griewank function results. The Greiwank function is a high-frequency, multi-modal landscape superimposed on a convex design landscape. When solving this problem, an initial population is likely to be formed in the large convex region and funneled to the edges of the flatter, multi-modal region in the center.

GA's focus on intermediate search heuristics is highly effective in a problem like Griewank. Intermediate search allows for pairs of solutions at opposite edges to effectively traverse the complex multi-modal region in the center. Conversely, algorithms that use neighborhood or variable neighborhood search heuristics will have to traverse many local optima that can trap individual members. Algorithms incorporating intermediate or directional search heuristics such as PSO, MEIGO, and Gnowee do better on this problem. Algorithms that focus on neighborhood or variable neighborhood search like SA and CS do poorly. Finally, SA and MCS use "cooling" parameters that limit the range of the neighborhood and variable neighborhood search heuristics as the number of generations progresses. This can lead to local trapping and premature convergence on large highly multi-modal problems like Griewank and is another factor in their poor performance.

V.C   Mixed-Integer Benchmark Results

The mixed-integer benchmarks allow for the performance of Gnowee to be assessed in problems containing a combination of continuous, integer, binary, and discrete variables. The summary of the FOM results obtained for the mixed-integer benchmark problems are shown in Table V, where bold results indicate fitness greater than 1% from the global optimum and italicized results indicate the best performance for the average and the overall best run.

Here, Gnowee obtains the best performance across all benchmarks considered. Similar to the

TABLE VI
Summary of FOM results for TSP optimization benchmarks.

	GA[43]	DCS[36]	Gnowee
Eil51	**10940.4**	1509.7	*555.6*
St70	**27452.0**	3804.6	*1403.1*
Pr107	**36563.1**	6067.9	*3380.5*
Bier127	**49869.9**	8337.3	*3918.6*
Ch150	**115495.0**	14396.6	*5261.4*

previous benchmarks, Gnowee maintains rapid, consistent, continual convergence throughout the optimization process across the range of the benchmarks considered. Although MEIGO performs similarly well for the spring and chemical process problems, it pre-maturely converges more often than not for the pressure vessel design. Once again, Gnowee's diverse, robust set of heuristics limits premature convergence across all benchmarks considered.

V.D   Combinatorial Benchmark Results

The combinatorial aspects of Gnowee were benchmarked using the TSP problems described in Section IV.E. Unlike the algorithms used to solve the previous benchmarks, the GA and DCS algorithms used TSP specific information, i.e. the distance between pairs of cities, to guide the search process. To enable a coherent comparison between the algorithms, distance based searches were also incorporated into the Gnowee heuristics.

The summary of the FOM results obtained for the TSP benchmark problems are shown in Table VI, where bold results indicate fitness greater than 1% from the global optimum and italicized results indicate the best performance for the average and the overall best run. Each row of the table indicates the benchmark TSP route considered.

Due to the dimensionality of the TSP problems, indicated by the number in the benchmark problem name, the solution space is extremely large. This large space makes 1% convergence results with limited function evaluations difficult, as illustrated in Table VI by the fact that 13 of the 15 cases do not meet the 1% convergence criteria. While not meeting the stated convergence criteria, Gnowee does outperform DCS and GA across all of the TSP benchmarks.

The primary driver for Gnowee's increased performance is a reduction in the number of

TABLE VII

Summary of function evaluation results for TSP optimization benchmarks.

	GA[43]	DCS[36]	Gnowee
Eil51	**103415 ± 24218**	104341 ± 39398	*9294 ± 6033*
St70	**157455 ± 30426**	174362 ± 37953	*16238 ± 9823*
Pr107	**159692 ± 37900**	**198006 ± 15497**	*27447 ± 16157*
Bier127	**197340 ± 10306**	**228991 ± 6345**	*37483 ± 18077*
Ch150	**199603 ± 3276**	**231786 ± 5338**	*48757 ± 24725*

function evaluations needed, as shown in Table VII. Gnowee reduces the number of function evaluations by a factor of ∼4-10 over the DCS and GA algorithms. The reduction in function evaluations will enable nearly-global solutions with limited function evaluations in the smaller sets of combinatorial variables found in typical engineering design problems.

VI CONCLUSION

This paper presented the new Gnowee algorithm and detailed comparison of its performance to well-established metaheuristic algorithms from the literature. Gnowee was developed to provide a modular, open-source, general-purpose hybrid metaheuristic optimization algorithm that can achieve rapid, nearly-globally optimum solutions for complex nuclear engineering design challenges. Gnowee is capable of optimizing constrained, noisy, derivative-free, multi-modal analytic or black box objective functions informed by continuous, integer/binary, discrete, and/or combinatorial variables. Gnowee is unique in its formulation and broad application where the metaheurisitic framework enables optimization of a wide variety of engineering problems with widely varying design vectors and fitness landscapes. Additionally, the rapid, consistent convergence of Gnowee enables the algorithm to be employed with high-cost objective functions such as radiation transport calculations.

Gnowee achieves this rapid, nearly global convergence across a large range of design vectors and fitness landscapes through the implementation of a diverse, robust metaheuristic framework that balances diversification and intensification strategies. This was demonstrated with a set of eighteen benchmark problems spanning multiple variable types where Gnowee's convergence rate and consistency were measured against established metaheuristic algorithms. While Gnowee was

not always the top performing algorithm, it also only finished worse than second once (the speed reducer benchmark where it finished a close third). Such high performance across so many problem types is why this algorithm is a good choice for many types of nuclear engineering problems.

Future work might include implementing better strategies to develop the initial population, the addition of population and diversity control algorithms, and a "bulletin board" approach to parallelization. Currently, several initialization strategies are available in Gnowee, but the benchmark results showed Gnowee to consistently generate worse starting solutions than the other algorithms considered. The results obtained indicate that the metaheuristic framework employed generally maintains a diverse set of solutions, but the few cases that did not converge within the specified criteria often suffered from a loss of diversity. Actively managing the diversity could improve the algorithm's performance across a wider range of problems. Finally, Gnowee was implemented with the idea that parallelization would occur at the function evaluation stage following each heuristic. However, asynchronous, "bulletin board" approaches could be employed to accelerate the wall time of the solution in time sensitive applications.

Overall, the benchmark results show that Gnowee maintained consistent, continual convergence across a wide set of benchmarks while all of the other algorithms considered suffered from premature convergence issues on several of the benchmarks considered. Gnowee therefore shows strong promise as an effective and all purpose optimization algorithm for a wide range of nuclear engineering problems.

DISCLAIMER

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

REFERENCES

[1] D. S. Lee, L. F. Gonzalez, J. Periaux, and K. Srinivas, "Efficient hybrid-game strategies coupled to evolutionary algorithms for robust multidisciplinary design optimization in aerospace engineering," *IEEE Transactions on Evolutionary Computation*, **15**, *2*, 133 (2011); 10.1109/TEVC.2010.2043364.

[2] X.-S. Yang, *Engineering Optimization*, John Wiley & Sons, Hoboken (2010).

[3] P. Möller, D. G. Madland, A. J. Sierk, and A. Iwamoto, "Nuclear Fission Modes and Fragment Mass Asymmetries in a Five-dimensional Deformation Space." *Nature*, **409**, *6822*, 785 (2001); 10.1038/35057204.

[4] J.-M. Do, J.-J. Lautard, A.-M. Baudron, S. Douce, and G. Arnaud, "Use of Meta-Heuristics for Design of Fuel Loading Pattern in Light Water Reactors Comprising Some Radial and Axial Heterogeneities," *2011 IEEE International Parallel and Distributed Processing Symposium*, 374–380 (2011); 10.1109/IPDPS.2011.175.

[5] O. Guler, *Foundations of Optimization*, Springer, New York (2010).

[6] R. Storn and K. Price, "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, **11**, 341 (1997); 10.1023/A:1008202821328.

[7] M. K. Marichelvam, T. Prabaharan, and X. S. Yang, "A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems," *Evolutionary Computation, IEEE Transactions on*, **18**, *2*, 301 (2014); 10.1109/TEVC.2013.2240304.

[8] X.-S. Yang and S. Deb, "Cuckoo Search via Levy Flights," *Nature & Biologically Inspired Computing*, 210–214 (2009).

[9] A. Ouaarab, B. Ahiod, and X. S. Yang, "Discrete Cuckoo Search Algorithm for the Travelling Salesman Problem," *Neural Computing and Applications*, **24**, *7-8*, 1659 (2014); 10.1007/s00521-013-1402-2.

[10] X.-S. Yang, *Nature-Inspired Optimization Algorithms*, Elsevier, London (2014).

[11] J. Hou, S. Qvist, R. Kellogg, and E. Greenspan, "3D in-core fuel management optimization for breed-and-burn reactors," *Progress in Nuclear Energy*, **88**, *April*, 58 (2016); 10.1016/j.pnucene.2015.12.002.

[12] H. Hu and Q. Wang, "Study on Composite Material for Shielding Mixed Neutron and Gama Rays," *IEEE Transactions on Nuclear Science*, **55**, *4*, 2376 (2008).

[13] S. Burer and A. N. Letchford, "Non-Convex Mixed-Integer Nonlinear Pogramming: A Survey," *Surveys in Operations Research and Management Science*, **17**, *2*, 97 (2012); 10.1016/j.sorms.2012.08.001.

[14] C. A. Floudas and C. E. Gounaris, "A Review of Recent Advances in Global Optimization," *Journal of Global Optimization*, **45**, *1*, 3 (2009); 10.1007/s10898-008-9332-8.

[15] P. Bonami, M. Kilinç, and J. Linderoth, "Algorithms and Software for Convex Mixed Integer Nonlinear Programs," J. Lee and S. Leyffer (Editors), *Mixed Integer Nonlinear Programming*, 1–39, Springer, New York; 10.1007/978-1-4614-1927-3.

[16] T. Hemker, "Derivative Free Surrogate Optimization for Mixed-Integer Nonlinear Black Box Problems in Engineering," PhD Thesis, Informatik der Technischen Universitat Darmstadt (2008).

[17] R. G. Regis and C. A. Shoemaker, "Constrained global optimization of expensive black box functions using radial basis functions," *J. Global Opt*, 31(1), 153–171 (2005).

[18] J. A. Egea, D. Henriques, T. Cokelaer, A. F. Villaverde, A. MacNamara, D.-P. Danciu, J. R. Banga, and J. Saez-Rodriguez, "MEIGO: An Open-Source Software Suite Based on Metaheuristics for Global Optimization in Systems Biology and Bioinformatics," *BMC Bioinformatics*, **15**, 136 (2014); 10.1186/1471-2105-15-136.

[19] L. Yiqing, Y. Xigang, and L. Yongjian, "An Improved PSO Algorithm for Solving Non-convex NLP/MINLP Problems with Equality Constraints," *Computers and Chemical Engineering*, **31**, *3*, 153 (2007); 10.1016/j.compchemeng.2006.05.016.

[20] G. Tao and Z. Michalewicz, "Iver-over Operator for the TSP," *Parallel Problem Solving from Nature - PPSN V, 5th International Conference*, Amsterdam (1998); DOI: 10.1007/BFb0056922.

[21] K. Sorensen, M. Sevaux, and F. Glover, "A History of Metaheuristics," *Handbook of Heuristics*, Springer; 10.1093/brain/122.11.2197-a.

[22] M. A. Lones, "Metaheuristics in Nature-Inspired Algorithms," *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, 1419–1422 (2014); 10.1145/2598394.2609841.

[23] F. Glover, "Paths for Integer Programming," *Computers and Operations Research*, **13**, *5*, 533 (1986); http://dx.doi.org/10.1016/0305-0548(86)90048-1.

[24] I. J. Fister, D. Fister, and I. Fister, "A Comprehensive Review of CS," *International Journal of Mathermatical Modeling and Numerical Optimization*, **4**, *4*, 387 (2013).

[25] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, **1**, *1*, 67 (1997); 10.1109/4235.585893.

[26] R. N. Mantegna, "Fast, Accurate Algorithm for Numerical Simulation of Levy Stable Stochastic Processes," *Physical Review E*, **49**, *5* (1994).

[27] K. Zhao, R. Jurdak, J. Liu, D. Westcott, B. Kusy, H. Parry, P. Sommer, and A. Mckeown, "Optimal Levy-Flight Foraging in a Finite Landscape," *Journal of the Royal Society Interface*, **12** (2015).

[28] T. Tran, T. T. Nguyen, and H. L. Nguyen, "Global Optimization using Levy Flights," *Second National Symposium on Research, Development and Application of Information and Communication Technology*, Hanoi (2004).

[29] A. Witze, "Sharks Use Math to Hunt," (2010)URL https://www.sciencenews.org/article/sharks-use-math-hunt.

[30] E. Pantaleo, P. Facchi, and S. Pascazio, "Simulations of Lévy flights," *Physica Scripta*, **T135** (2009).

[31] R. N. Mantegna and H. E. Stanley, "Stochastic Process with Ultraslow Convergence to a Gaussian: The Truncated Levy Flight," *Physical Review Letters*, **73**, *22*, 2946 (1994); 10.1103/PhysRevLett.73.2946.

[32] F. A. C. Viana, "Things You Wanted to Know About the Latin Hypercube Design and Were Afraid to Ask," *10th World Congress on Structural and Multidisciplinary Optimization*, 1–9, Orlando (2013).

[33] T. M. Cioppa and T. W. Lucas, "Efficient Nearly Orthogonal and Space-Filling Latin Hypercubes," *Technometrics*, **49**, *1*, 45 (2007); 10.1198/004017006000000453.

[34] F.-M. D. Rainville, C. Gagné, O. Teytaud, and D. Laurendeau, "Evolutionary Optimization of Low-Discrepancy Sequences," *ACM Transactions on Modeling and Computer Simulation*, **22**, 1 (2012); 10.1145/2133390.2133393.

[35] S. Lin and W. Kernighan, "An Effective Heurisic Algorithm for the Traveling-Salesman Problem," *Operations Research*, **21**, *2*, 498 (1973).

[36] Y. Zhou, X. Ouyang, and J. Xie, "A Discrete Cuckoo Search Algorithm for Travelling Salesman Problem," *International Journal of Collaborative Intelligence*, **1**, *1*, 68 (2014).

[37] W. K. Hastings, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," *Biometrika*, **57**, *1*, 97 (1970).

[38] T. Starkweather, S. McDaniel, K. E. Mathias, L. D. Whitley, and C. Whitley, "A Comparison of Genetic Sequencing Operators," *Proceedings of the fourth International Conference on Genetic Algorithms*, 69–76 (1991).

[39] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *IEEE International Conference on Neural Networks*, vol. 4, 1942–1948 (1995); 10.1109/ICNN.1995.488968.

[40] S. Walton, O. Hassan, K. Morgan, and M. R. Brown, "Modified Cuckoo Search: A New Gradient Free Optimisation Algorithm," *Chaos, Solitons and Fractals*, **44**, *9*, 710 (2011); 10.1016/j.chaos.2011.06.004.

[41] J. A. EGEA, R. MARTI, and J. R. BANGA, "An Evolutionary Method for Complex Process Optimization," *Computers & Operations Research*, **37**, *1*, 315 (2010).

[42] T. BÄCK and H.-P. SCHWEFEL, "An Overview of Evolutionary Algorithms for Parameter Optimization," *Evolutionary Computation*, **1**, *1*, 1 (1993); 10.1162/evco.1993.1.1.1.

[43] MATHWORKS, "MatLab: Global Optimization Toolbox User's Guide R2015b," (2015).

[44] X. OUYANG, Y. ZHOU, Q. LUO, and H. CHEN, "A Novel Discrete Cuckoo Search Algorithm for Spherical Traveling Salesman Problem," *Applied Mathematics and Information Sciences*, **7**, *2*, 777 (2013); 10.12785/amis/070248.

[45] L. C. CAGNINA, S. C. ESQUIVEL, and C. A. COELLO COELLO, "Solving Engineering Optimization Problems with the Simple Constrained Particle Swarm Optimizer," *Informatica (Ljubljana)*, **32**, *3*, 319 (2008).

[46] S. AKHTAR, K. TAI, and T. RAY, "A Socio-Behavioural Simulation Model for Engineering Design Optimization," *Engineering Optimization*, **34**, *4*, 341 (2002); 10.1080/03052150212723.

[47] J. LAMPINEN and I. ZELINKA, "Mixed Integer-Discrete-Continuous Optimization by Differential Evolution," *MENDEL'99*, Brno (1999).