Territorial Differential Meta-Evolution: An Algorithm for Seeking All the Desirable Optima of a Multivariable Function

Richard Wehr rwehr@aerodyne.com
Department of Ecology and Evolutionary Biology, University of Arizona, Tucson, AZ, 85721, U.S.A.
Current: Aerodyne Research, Inc., Billerica, MA, 01821, U.S.A.

Scott R. Saleska saleska@arizona.edu
Department of Ecology and Evolutionary Biology, University of Arizona, Tucson, AZ, 85721, U.S.A.

https://doi.org/10.1162/evco_a_00337

Abstract

Territorial Differential Meta-Evolution (TDME) is an efficient, versatile, and reliable algorithm for seeking all the global or desirable local optima of a multivariable function. It employs a progressive niching mechanism to optimize even challenging, high-dimensional functions with multiple global optima and misleading local optima. This paper introduces TDME and uses standard and novel benchmark problems to quantify its advantages over HillVallEA, which is the best-performing algorithm on the standard benchmark suite that has been used by all major multimodal optimization competitions since 2013. TDME matches HillVallEA on that benchmark suite and categorically outperforms it on a more comprehensive suite that better reflects the potential diversity of optimization problems. TDME achieves that performance without any problem-specific parameter tuning.

Keywords

Function optimization, niching, differential evolution.

1 Introduction

The optima of multivariable functions are commonly sought but can be difficult or impossible to find. Analytical solutions are rarely obtainable and exhaustive sampling is usually impractical because even a narrowly bounded function domain quickly becomes unfathomable as its dimensionality increases. Consider that a 50-variable domain with just five distinguishable values for each variable contains more distinguishable locations than there are femtoseconds in the age of the universe. The search (i.e., optimization) must therefore proceed selectively and iteratively, guided by limited information gleaned about the function's "topography." Considering the optimization as minimization (without loss of generality), an intuitive approach would be to follow the local function gradient "downhill," but doing so can lead to a local rather than a global optimum if the function is multimodal, that is, if it has more than one basin.

A plethora of heuristic optimization algorithms have been developed over the last few decades in order to more reliably locate the global optima of multimodal functions (Sreedhar et al., 2020; Del Ser et al., 2019; Molina et al., 2018). Two powerful families of

Manuscript received: 6 April 2021; revised: 9 March 2022, 13 October 2022, 9 May 2023, and 6 June 2023; accepted: 9 June 2023.

heuristic optimizers are the swarm algorithms and the evolutionary algorithms (Sreedhar et al., 2020; Del Ser et al., 2019; Molina et al., 2018). The latter work somewhat analogously to biological evolution, treating variables as genes and points in the domain as individuals in a population that evolves by mutation, recombination, and selection of the fittest (Slowik and Kwasnicka, 2020). One particularly notable evolutionary algorithm that figures prominently in this work is Differential Evolution, or DE (Storn and Price, 1995; Price et al., 2005), which has spawned many variants in the 28 years since its publication (Das et al., 2016; Neri and Tirronen, 2009).

When there might be more than one global or desirable local optimum, and if the goal is not merely to find *any* of them but rather to find *all* of them, then a multimodal optimizer (MMO) is necessary (Das et al., 2011). An MMO can also be helpful if the function's topography consistently guides searches in the wrong direction. MMOs use various "niching" techniques to define subpopulations that can converge on multiple optima in a single coordinated search. Note the distinction between a multimodal function, which has multiple (local and/or global) optima, and a multimodal optimizer, which tries to find multiple (local and/or global) optima. Most swarm and evolutionary optimizers are unimodal (they can find only a single optimum) but all of the functions that require them are multimodal (as unimodal functions are easily optimized by gradient-based search methods).

In order to encourage and monitor the development of MMOs, quasi-annual competitions run by the IEEE Congress on Evolutionary Computation (CEC) and the Genetic and Evolutionary Computation Conference (GECCO) have pitted state-of-the-art MMOs against one another on the same benchmark suite since 2013 (Li et al., 2013; Molina et al., 2018; Epitropakis, 2019). That suite, which we will refer to as the CEC benchmark, consists of 12 test functions in various dimensionalities between 1D and 20D, for a total of 20 test problems. It is the only MMO benchmark suite in widespread use. Of the more than 20 optimizers that have competed on the CEC benchmark, the champion is the Hill-Valley Evolutionary Algorithm, or HillVallEA (Maree et al., 2018a, 2018b, 2019); the preceding champion was Covariance Matrix Self-Adaptation with Repelling Subpopulations, or RS-CMSA (Ahrari et al., 2017). There are also many MMOs that have been published over the last decade but not tested on the CEC benchmark, which is a problem lamented by Del Ser et al. (2019) because there are no objective, systematic data for comparing the performance of those algorithms. Even a dedicated, laborious comparison experiment might be impractical at this point, as there are too many algorithms to implement and test, and not enough comparative data to select a competitive subset.

The CEC benchmark is a major accomplishment in itself, and has been tremendously useful, but it is limited in three respects. First, it is weighted towards low dimensionality (D): 15 of its 20 problems have $D \leq 3$. As MMOs have become increasingly capable, perfect scores have become common on most of those problems, which therefore do not aid in ranking algorithms. More importantly, the weighting towards low D means that a CEC benchmark score does not give a good indication of performance on the sorts of challenging, high-D problems for which a clever heuristic is absolutely necessary due to the vastness of the search space—as are common, for example, in real-world model optimization.

Second, the CEC benchmark's stipulated search bounds are all unrealistically prescient: just large enough to include all the desired optima without much wasted space in any dimension. In real-world problems, the search bounds must be set without knowing where the desired optima lie, and tend to be generous for fear of excluding any. As a result, the desired optima commonly end up clustered in one region of the domain.

Third, the entire CEC benchmark favors algorithms that explore almost exclusively by niching—in other words, algorithms that allocate their computational resources to a large number of small (and therefore exploitative) subpopulations rather than a small number of large (and therefore explorative) ones. This bias in the benchmark was discovered in the course of our experiments and is perhaps not surprising, given that the benchmark was designed to test niching algorithms. As we will show, there are many standard benchmark problems in the literature that favor instead a single large population. For all three of the above reasons, the CEC benchmark may reward algorithms that do poorly in some important real-world situations.

Here we present a new multimodal optimizer, Territorial Differential Meta-Evolution (TDME), in which subpopulations evolve by DE within a succession of territories that are themselves defined partly by evolution. One advantage of TDME is that it weights effort towards the more fruitful regions of the domain, partitioning those regions more finely than the rest. Another advantage is that it can choose autonomously between the two resource allocation strategies mentioned above (many small subpopulations versus few large ones) in order to achieve exceptional versatility. We follow the recommendations of Del Ser et al. (2019) by evaluating TDME on the CEC benchmark using the benchmark champion as a baseline, and then we go on to evaluate both algorithms on a new, more comprehensive MMO benchmark of 86 problems that incorporates higher dimensionalities, poorly chosen search bounds, and functions that do not favor niching. We present the TDME algorithm in Section 2, describe our benchmark and evaluation criteria in Section 3, and report and discuss the results of the evaluations in Section 4.

2 The Territorial Differential Meta-Evolution Algorithm

2.1 Overview

TDME is a versatile multimodal optimization algorithm that does not require any tuning by the user to achieve the competitive performance demonstrated in this article across a wide variety of problem types. It can find any number of global optima, as well as local optima that might be of interest. The heart of TDME is Random-Control Differential Evolution (rcDE), a simple but versatile new variant of the unimodal optimizer DE (see Section 2.2). TDME runs a separate rcDE search in each of multiple domain-space territories whose borders are determined by an overarching, DE-like evolutionary process (see Section 2.3). The population of a territory is called a species because its individuals breed with one another but not with individuals from other territories. In this biological analogy, the territories represent genetic similarity rather than spatial proximity.

Pseudocode for TDME is provided in Algorithms 1–6. Actual code for TDME and its evaluation, in the R programming language, is included in the supplementary material.

2.2 Core Unimodal Optimizer: Random-Control Differential Evolution

Random-Control DE is simply the original DE/rand/1/bin version of Differential Evolution (Storn and Price, 1995) but with randomized control parameters. So let us first describe DE/rand/1/bin.

DE/rand/1/bin works on a population of NP individuals (i.e., NP points in the domain). In each generation, those NP individuals generate NP mutants with which they then breed to produce NP offspring. Each mutant is generated by multiplying the difference vector between two random individuals by a scale factor F and then adding

Algorithm 1: The high-level structure of TDME

```
potential number of live species = 1;
NP = 24;
initialize the first species by uniform random sampling on the whole domain;
while budget remaining do
    for sp = 1, ..., potential number of live species do
        evolve species sp by one generation (Algorithm 2);
        terminate species sp if it satisfies any termination criterion
         (Algorithm 5);
    end
    if all species have been terminated then
        if 5 cohorts in a row have ended entirely in stagnation then
            warn user to increase the number of generations used to check for
             stagnation;
        end
        if this will be the 2^nd cohort then
            NP = 8;
        else
            if this will be the 5^th cohort then
                if best fitness with NP = 24 was better than best fitness with NP = 8
                 then
                    NP = 24;
                end
            end
        end
        seed new species (Algorithm 3)
    end
end
```

the result to a third random individual. Use of a random difference vector causes the magnitude of mutation to scale with the diversity of the population as it adapts to the function's topography; it is the key innovation for which DE is named. In the subsequent breeding (or "crossover" or "recombination") step, each mutant breeds with a random unmutated individual that was not involved in its creation. Their offspring is generated by drawing each of its genes from one or the other of its parents, according to a crossover probability *CR*. The offspring then competes against its unmutated parent in terms of which has the better function value (or "fitness"). If the offspring has the better function value, it is selected to replace its unmutated parent in the population; otherwise, the parent is retained and the offspring is discarded. Mutants are used only for breeding and never enter the population. The three control parameters *NP*, *F*, and *CR* strongly impact the performance of DE, but in different ways on different problems. There is no one configuration that is effective in general, and no way to know how to select a suitable configuration for an unsolved problem except by costly trial and error. This lack of universality is the chief weakness of DE.

Several DE variants have approached this challenge by incorporating dynamic adaptation schemes for *F* and/or *CR* into the DE process (Das et al., 2016; Qin et al., 2009; Zhang and Sanderson, 2009). In such schemes, information about the search history is used to automatically adjust the control parameters as the search progresses.

Algorithm 2: Evolution of one species by one generation in TDME

```
   // X is an NP × D matrix whose element X_{i,d} is the
      location of individual i in dimension d
1  P = a vector formed by randomly shuffling the set {1, ..., NP};
2  CR = a vector of NP elements randomly generated on the interval (0,1];
3  randomDimension = a vector of NP elements randomly sampled with
     replacement from the set {1, ..., D};
4  for d = 1, ..., D do
5      F = a vector of NP elements randomly generated on the interval [0.5,1];
6      randomNumber = a vector of NP elements randomly generated on the
         interval [0,1];
7      for i = 1, ..., NP do // mutation
8          p = P_i; q = P_{(i+1) modulo NP}; r = P_{(i+2) modulo NP}; s = P_{(i+3) modulo NP};
9          M_{p,d} = X_{q,d} + F_i(X_{r,d}–X_{s,d}) // location of mutant p in
               dimension d
10     end
11     for i = 1, ..., NP do // crossover
12         if randomNumber_i ≤ CR_i or d = randomDimension_i then
13             O_{i,d} = M_{i,d} // location of offspring i in dimension d
14         else
15             O_{i,d} = X_{i,d} // location of offspring i in dimension d
16         end
17         reflect O_{i,d} off the search bounds as necessary to keep it in the domain;
18     end
19 end
20 discard and replace (by mutation and crossover) any offspring that is beyond
     the territory margins for species sp;
21 for i = 1, ..., NP do // selection
22     if parent i is fitter than offspring i then
23         for d = 1, ..., D do
24             X_{i,d} = O_{i,d};
25         end
26     end
27 end
```

In our experiments with precursors to TDME (which we do not present), dynamic parameter adaptation tended to encourage convergence on local optima and therefore reduced performance on challenging multimodal problems. One possible explanation is that the control parameters that gave the fastest short-term fitness improvement were those that led to convergence on the first optimum that presented itself. That behavior, however, might not be characteristic of dynamic parameter adaptation in general.

We found that a safer and simpler way to configure *F* and *CR* for success across the diverse problem suite in Section 3.2 was to randomize them within broad ranges. Specifically, rcDE draws a new *F* from a uniform distribution in the range [0.5, 1] for each gene of each individual in each generation, and draws a new *CR* from a uniform distribution in the range (0, 1] for each individual in each generation (but the same *CR* is used for all of the individual's genes); see lines 2 and 5 of Algorithm 2. This

randomization scheme was chosen as optimal based on experiments with various bounds for *F* and *CR* and with both uniform and Gaussian sampling distributions. The randomization of *F* is an idea that goes back at least to the DERSF algorithm (Das et al., 2005), while limited randomization of both *F* and *CR* was used to support their adaptation in the SDE (Salman et al., 2007), BBDE (Omran et al., 2009), SaDE (Qin et al., 2009), and DE/cluster (Li and Zhang, 2011) algorithms, and appeared most recently in the DSM-DE algorithm (Deng et al., 2019). rcDE differs from those DE variants in that it relies entirely on broad randomization of *F* and *CR* for versatility, without any parameter adaptation scheme. That versatility will be seen in Table 6, where rcDE's performance across a variety of functions compares well with JADE (Zhang and Sanderson, 2009), SaDE, and DE/cluster.

In addition to *F* and *CR*, it is necessary to choose a value for the population size *NP*. In the literature, *NP* has usually been either held constant at a value like 50 or else scaled according to the dimensionality *D*, although there has been no consensus on such a scaling relationship. Some algorithms have also used an adaptive *NP* (see Section 3.3 of Das et al., 2016). Our tests on the functions in Section 3.2 revealed that the optimal value of *NP* does not depend on the dimensionality of the problem per se but does depend on the character of the function's topography; for example, the optimal *NP* for rcDE is about 12 on the Ackley function, 20 on the Rosenbrock function, and 30 on the Schwefel function regardless of whether *D* is 10, 50, or 100 (see Section 3.2 for a description of these functions). The topography matters because a larger *NP* enhances exploration and thereby decreases the chance of converging before finding the global optimum—a well-known phenomenon (Yaman et al., 2019; Del Ser et al., 2019) that we confirmed in our testing of TDME variants and other algorithms. We found that, typically, if the local optima are few or uncompetitive (i.e., narrow or shallow compared to the global optimum), then a low *NP* (e.g., 6) is most efficient in rcDE. On the other hand, if the local optima are numerous and competitive, then a high *NP* (e.g., 30) is most efficient. In extreme situations, the topography might be so misleading that no feasible value of *NP* would be sufficient for a unimodal optimizer to find the global optimum. We consider a misleading topography to be one that tends to lead the search away from the global optimum; for example, because most of the domain is a large basin with a local optimum at the bottom, while the global optimum lies in a very narrow shaft at the large basin's rim. Given that the number of global optima and the number of competitive local optima increase differently with *D* for different functions, it makes sense that past studies have disagreed about how *NP* and *D* should relate. Unfortunately, *D* and the search bounds are the only pieces of information we have about a function before we attempt to solve it, and neither they nor the early performance of the optimizer gives any reliable indication of whether *NP* is too low or too high for ultimate efficiency or success.

TDME alleviates this *NP* dilemma in two ways. First, TDME adds a second mechanism for exploration: niching. The exploratory power provided by niching alleviates the need for a high *NP* even on challenging functions. For example, exploratory power would be reduced by dividing a single large population into several small, uncoordinated (and therefore partially redundant) subpopulations, but some new exploratory power would be gained by directing each of those subpopulations to a different region of the search space. For that reason, the introduction of niching reduces the optimal *NP*. Niching does not completely resolve the *NP* dilemma, though, because niching and evolution explore in fundamentally different ways, such that the optimal balance between them still depends on the function topography in an unpredictable way. For instance, we found that minimizing *NP* and relying almost entirely on niching for exploration is

the best strategy on all the functions in the CEC benchmark, but that it is a ruinously inefficient strategy on the Rosenbrock and Schwefel functions. Fortunately, TDME also helps in a second, crucial way: it runs multiple rcDE searches and thereby provides an opportunity to compare *NP* values and choose the best one. TDME's precise mechanism for choosing *NP* is described in Section 2.3.

Algorithm 2 describes the evolution of one species by one generation in TDME. Note that line 8 of Algorithm 2, which defines p, q, r, and s, is a convenient way to generate four random permutations of the set $\{1, \ldots, NP\}$ such that no element of the set occupies the same position in more than one permutation. This technique is used to ensure that the four roles involved in the generation of an offspring (i.e., the unmutated parent, the two individuals that define the difference vector, and the individual to which the difference vector is added) are always played by four different individuals, and that each individual plays each role for one and only one offspring.

2.3 Territorial Meta-Evolution

TDME's approach to niching is to partition the domain into a progressively larger number of territories and run a separate rcDE search in each. Territories that exist simultaneously are said to be in the same cohort. Once all the searches in a cohort are complete, all territory borders are forgotten and the domain is repartitioned into a new set of territories for the next cohort. This subsection describes the details of the territorial partitioning mechanism.

Before beginning territorial niching, TDME runs one rcDE search with $NP = 24$ on the whole domain, to rapidly solve many problems with a single global optimum (note lines 1–3 of Algorithm 1). The only test function we found that benefitted from a higher population size than that was the Schwefel function, for which the optimal value was closer to 30. But even on Schwefel, $NP = 24$ rarely failed on its first try.

After that initial high-*NP* rcDE search, TDME runs rcDE searches with $NP = 8$ in a succession of cohorts, with one territory in the first cohort, two in the second, three in the third, and so on (nominally; see Section 2.5 for exceptions). $NP = 8$ is close to the smallest viable population size for rcDE, and to the optimal population size for all the problems in the CEC benchmark, according to our testing. After three cohorts with $NP = 8$, TDME compares the best fitness obtained during those three cohorts (containing a total of up to six species) to the best fitness obtained during the original $NP = 24$ cohort (containing only one species). If the best fitness found using $NP = 8$ was as good as or better than the best fitness found using $NP = 24$, then *NP* is fixed at 8 for all future species and the nominal number of territories per cohort continues to increment by one each cohort; otherwise, *NP* is fixed at 24 and the nominal number of territories per cohort reverts to 2, incrementing by one each cohort thereafter (see lines 13–21 of Algorithm 1). The idea here is to do a brief comparison between two opposing strategies that require roughly equal computational effort: using a single population with $NP = 24$ versus niching into subpopulations with $NP = 8$. This method of assessing *NP* is not guaranteed to choose the best value for every function, but it was the most efficient among the numerous candidate methods we devised, and led to excellent results on every function we tested. The cost of the assessment is minimal and the benefit is great.

Territorial partitioning occurs by the following seeding mechanism (Algorithm 3). For each territory in a cohort, TDME places a seed on the whole domain. There are two mechanisms for generating live seeds (as opposed to ghost seeds, which are discussed in Section 2.5). All the live seeds in a cohort use the same mechanism, but the

Algorithm 3: Seeding of new species in TDME

```
1  seeding successful = false;
2  while seeding successful = false do // a successful seeding is one in
     which at least one seed remains live (see lines 22–26)
3      if at least two cohorts have completed then
4          if this will be the 5^th cohort and NP = 24 then
5              potential number of live species = 1;
6          end
7          increment potential number of live species by 1;
8      end
9      clear all ghost seeds;
10     for t = 1, ..., number of terminal points do
11         if terminal point t is eligible and its age < its ghost seed lifetime then
12             make terminal point t a ghost seed;
13         end
14     end
15     increment terminal point ages by 1;
16     if potential number of live species + number of ghost seeds > 1 then
17         if number of terminal points ≥ 5 and previous cohort did not use
             meta-evolution then
18             generate live seeds by meta-evolution (Algorithm 4);
19         else
20             generate live seeds by uniform random sampling on the whole
               domain;
21         end
22         for sp = 1, ..., potential number of live species do
23             if live seed sp is in the same basin as a candidate global optimum then
24                 turn live seed sp into a ghost seed;
25             end
26         end
27         if there is still at least one live seed then
28             seeding successful = true;
29         end
30     else
31         seeding successful = true;
32     end
33 end
34 if number of live seeds + number of ghost seeds > 1 then
35     generate a species population of size NP from each live seed (see
         Section 2.3);
36 else
37     initialize individuals by random uniform sampling on the whole domain;
38 end
```

chosen mechanism alternates from one cohort to the next. The first mechanism is simply uniform random sampling on the whole domain (line 20 of Algorithm 3). The second mechanism is a meta-evolutionary process similar to a generation of DE (Algorithm 4,

Algorithm 4: Territorial meta-evolution in TDME

1 $\mathbf{Y}$ = a $5 \times D$ matrix whose element $\mathbf{Y}_{i,d}$ is the location of the i^{th}-fittest point from the terminal point log in dimension d // selection
2 J = ceiling(*potential number of live species*/5);
3 clear $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{S}$;
4 **for** $j = 1, \ldots, J$ **do**
5 $\quad \mathbf{Q}_0$ = a vector formed by randomly shuffling the set {1,..., 5};
6 $\quad \mathbf{R}_0 = \mathbf{Q}_0$ offset by one element, with the last element of $\mathbf{Q}_0$ becoming the first element of $\mathbf{R}_0$;
7 $\quad \mathbf{S}_0 = \mathbf{R}_0$ offset by one element, with the last element of $\mathbf{R}_0$ becoming the first element of $\mathbf{S}_0$;
8 $\quad$ append $\mathbf{Q}_0$ to $\mathbf{Q}$, $\mathbf{R}_0$ to $\mathbf{R}$, and $\mathbf{S}_0$ to $\mathbf{S}$;
9 **end**
10 crop $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{S}$ to *potential number of live species* elements;
11 $\mathbf{CR}$ = a vector of *potential number of live species* elements randomly generated on the interval (0,1];
12 $\mathbf{randomDimension}$ = a vector of *potential number of live species* elements randomly sampled with replacement from the set $\{1, \ldots, D\}$;
13 **for** $d = 1, \ldots, D$ **do**
14 $\quad \mathbf{F}$ = a vector of *potential number of live species* elements randomly generated on the interval (0,1];
15 $\quad \mathbf{randomNumber}$ = a vector of *potential number of live species* elements randomly generated on the interval [0,1];
16 $\quad$ **for** $i = 1, \ldots,$ *potential number of live species* **do** // mutation
17 $\quad\quad q = \mathbf{Q}_i; r = \mathbf{R}_i; s = \mathbf{S}_i;$
18 $\quad\quad \mathbf{M}_{i,d} = \mathbf{Y}_{q,d} + \mathbf{F}_i(\mathbf{Y}_{r,d} - \mathbf{Y}_{s,d})$ // location of mutant i in dimension d
19 $\quad$ **end**
20 $\quad$ **for** $i = 1, \ldots,$ *potential number of live species* **do** // crossover
21 $\quad\quad$ **if** $\mathbf{randomNumber}_i \leq \mathbf{CR}_i$ or $d = \mathbf{randomDimension}_i$ **then**
22 $\quad\quad\quad \mathbf{Z}_{i,d} = \mathbf{M}_{i,d}$ // location of seed i in dimension d
23 $\quad\quad$ **else**
24 $\quad\quad\quad \mathbf{Z}_{i,d} = \mathbf{Y}_{i,d}$ // location of seed i in dimension d
25 $\quad\quad$ **end**
26 $\quad\quad$ reflect $\mathbf{Z}_{i,d}$ off the search bounds as necessary to keep it in the domain;
27 $\quad$ **end**
28 **end**

called from line 18 of Algorithm 3). This process always uses five parents, regardless of the number of seeds required. The parents are the fittest five points in the terminal point log, which consists of the fittest point found by each completed rcDE search so far. Any ties for fitness in the log are broken randomly. If there are fewer than five points in the log (as is always the case for the first cohorts), then the random seeding mechanism is used instead. Once the five parents have been chosen (line 1 of Algorithm 4), their offspring are generated by the same mutation and recombination process that occurs in an rcDE generation, except that parents are now allowed to play the same role (such as the unmutated parent, or an endpoint of the difference vector) for more than one

offspring, because there may be more offspring (i.e., seeds) than parents (compare lines 2–10, 17, and 18 of Algorithm 4 to lines 1, 8, and 9 of Algorithm 2). Also unlike rcDE, there is no competition between offspring and parents; all the offspring become seeds (note the absence of any selection loop after the mutation and crossover loops in Algorithm 4). In this meta-evolutionary process, selection of the fittest occurs instead when the five parents are chosen from the terminal point log, so that the process is guided not by how fortuitously the rcDE searches start, but by what they find. The alternation between evolutionary and random seeding encourages TDME to gradually concentrate on the most fruitful regions of the domain while preventing it from being permanently misled.

The territory borders are defined in terms of the seed locations. The potential border between two territories is the hyperplane equidistant from their seeds, but not every pair of territories will end up sharing a border (because another territory might lie between them). The actual border of a territory in a given direction is whichever of its potential borders is closest to its seed in that direction. Thus, a point is within a particular territory if and only if it is closer to that territory's seed than to any other seed. Additionally, a point is said to be within the margin of a territory's border if the point is no more than 20% farther from the territory's seed than it is from the other seed that defines the border. The rcDE offspring of a species are allowed to be placed within its territory or the margins of its territory. Potential offspring that are beyond the margins are rejected and replaced. The margins are used to identify when a species is being attracted by a basin in another territory. If the whole species migrates into the same margin, the species is immediately terminated to minimize waste of computational resources.

Each live species is initialized within its territory (not the margins) by uniform random sampling adapted to the shape of the territory, as follows. The first member of the species is the seed. Then, for each additional member, a template point is generated by uniform random sampling in a hypercube. The coordinates of that template point relative to the origin are divided by the distance from the origin to the surface of the hypercube in the direction of the template point. The coordinates are then multiplied by the distance from the seed to the territory border in the same direction. Finally, the corresponding member is placed in the territory at those rescaled coordinates relative to the seed.

2.4 Species Termination Criteria

The criteria by which subpopulations are deemed to have completed their searches are sometimes mentioned only briefly but have a great impact on the performance of an MMO (Zielinski et al., 2006). There are three reasons that TDME might terminate an rcDE search: convergence, stagnation, and marginal migration (Algorithm 5).

Convergence is deemed to have occurred when a fitness criterion and a proximity criterion are both met. The fitness criterion is that the difference between the maximum and minimum fitness in the species in the current generation (the "fitness diversity") is 100 times less than the greater of: (a) the predefined fitness tolerance for success (set to 10^{-5} in this work) and (b) the difference between the best fitness in the species and the best fitness found by any species so far (line 3 of Algorithm 5). The idea is that evolution requires diversity, and an rcDE search is extremely unlikely to ever improve its best fitness by more than 100 times its fitness diversity. If the fitness diversity is 100 times less than the fitness tolerance for success, then either the species has already succeeded, or it never will. And if the fitness diversity is 100 times less than the difference between the best fitness in the species and the best fitness found by any species so far, then the species

Algorithm 5: Termination of a species in TDME

```
B = the fittest individual from species sp;
converged, stagnated, and migrated = false;
if fitness diversity of species sp < max(tolerance, fitness of B – best fitness so far)/100
 then
    converged = true;
    for d = 1, ..., D do
        if diversity of species sp in dimension d > search range/1000 then
            converged = false;
        end
    end
end
if converged = false then
    if all of species sp is in one of its margins then
        migration = true;
    else
        if negligible improvement in mean fitness of species sp over last 100 gens then
            stagnated = true;
        end
    end
end
if converged, stagnated, or migrated = true then
    if B is close to and in the same basin as an entry in the terminal point log then
        if B and the entry have nearly the same fitness then
            if the entry is not already flagged as eligible to be a ghost species then
                flag the entry as eligible to be a ghost species;
                set the ghost seed lifetime for the entry to 4;
            else
                double the ghost seed lifetime for the entry;
            end
        else
            if B is fitter than the entry then
                flag the entry as ineligible to be a ghost species;
            else
                double the ghost seed lifetime for the entry;
            end
        end
        if B is even slightly fitter than the entry then
            replace the entry's fitness and location with B;
        end
        reset the age for the entry to 1;
    else
        add B to the terminal point log;
        set the age for this new terminal point log entry to 1;
        flag this new entry as ineligible to be a ghost species;
    end
    turn species sp into a ghost species;
end
```

will never match that fitness and must be converging on a local optimum. The proximity criterion is that the domain-space diversity in each dimension is 1,000 times less than the original search range in that dimension (line 6 of Algorithm 5). This criterion is more arbitrary and less essential but avoids a mistaken declaration of convergence if the species enters a broad, flat region of the domain.

Stagnation is deemed to have occurred when there has been negligible improvement to the mean fitness in the species over the last 100 generations (line 15 of Algorithm 5). "Negligible" is judged by projecting the recent rate of fitness improvement out well beyond the user-specified computational budget (by a factor of five) to see whether the species has any hope of reaching the best fitness found by any previous species before the budget is exceeded. Stagnation looks similar to convergence in terms of fitness, but a stagnated species has not converged on a point (for example, because it is split between two similar basins). The choice of 100 generations is arbitrary; it was effective for all the functions tested in this work, but a larger number of generations (e.g., 1,000) could be required to prevent premature declarations of stagnation if the dimensionality is high (e.g., $D > 50$) and the landscape is particularly tortuous. On the other hand, using 1,000 generations to check for stagnation would slow the overall search unnecessarily on some functions, as truly stagnated species would persist longer before being terminated. If five cohorts in a row end entirely in stagnation, TDME issues a warning to the user to increase the number of generations used to check for stagnation by a factor of five or ten.

Marginal migration was mentioned in Section 2.3. It is deemed to have occurred when the entire species has entered the same territory margin, indicating that it is being attracted by an optimum in another territory, which it may be forbidden from actually reaching (line 12 of Algorithm 5).

Once convergence, stagnation, or marginal migration has been declared, a proximity test and a hill-valley test (Section 2.6) are used to check whether the best point found by the species is already represented in the terminal point log (line 21 of Algorithm 5). If not, then the species' best point is added to that log (line 41 of Algorithm 5). If so, and if the new point is fitter than the preexisting log entry, then the entry is updated with the new point (line 37 of Algorithm 5). Moreover, if the fitness of the new point agrees with that of the preexisting entry to within a tolerance (equal to ten times the convergence fitness criterion mentioned above), then the entry is considered to have been found twice, in which case it is declared eligible to become the seed of a ghost species (line 24 of Algorithm 5) and (if it is also a candidate global optimum) to forbid live species from seeding in its basin, as explained in Section 2.5.

2.5 Ghost Species

Like territorial meta-evolution, ghost species are a new idea invented for TDME. Ghost species serve to discourage live (i.e., ordinary) species from revisiting old parts of the domain. A ghost species has a seed and a territory just like a live species, but no population. Thus live species are not allowed to expand into ghost territories. Entries in the terminal point log become ghost seeds once they have been found twice (see Section 2.4 and Algorithms 3 and 5). Being found twice is a prerequisite so that future searches will not be blocked from discovering a global optimum merely because an inadequate search randomly converged somewhere nearby. A new ghost seed persists through four cohorts and then expires. If the same point is found again after that, then it again becomes a ghost seed but now with a lifetime of eight cohorts—and then 16 the next time, and so on (lines 27 and 33 of Algorithm 5). This doubling of the ghost lifetime creates

an opportunity to reexplore old parts of the domain at intervals, in case an optimum was missed, but discourages such exploration more strongly the more redundant it is revealed to be. Importantly, exploration can still happen close to a ghost seed if a live seed happens to be placed nearby. TDME's ghost species were inspired by the taboo points in RS-CMSA, but the mechanism is altogether different.

There is a second way for a ghost species to be created, inspired by HillVallEA. If a live species is seeded in the same basin as a candidate global optimum that has been found at least twice, then the species is not populated but becomes a ghost instead, for that one cohort only (line 24 of Algorithm 3). A candidate global optimum is a point in the terminal point log whose fitness differs from that of the best point in the log by less than the fitness tolerance for success (here 10^{-5}). Whether or not a seed is in the same basin as a candidate global optimum is decided by a hill-valley test (Section 2.6).

Because of the presence of ghosts, the number of territories does not necessarily increment by one each cohort. Rather, it is the number of potential live species that increments by one ("potential" because some of those may turn out to be ghost species if they are seeded in the same basin as a candidate global optimum). The number of territories is then equal to the number of live species plus the number of ghost species. Thus the number of territories may even decrease from one cohort to the next, if several ghost seeds expire at the same time. If all potential live seeds become ghosts, then TDME skips the cohort, increments the number of potential live seeds if appropriate, and proceeds with seeding the next cohort (see the while-loop that starts on line 2 of Algorithm 3).

A snapshot of a TDME generation with both live and ghost species is shown in Figure 1.

2.6 Hill-Valley Testing

TDME uses a hill-valley test (Algorithm 6) whenever it needs to estimate whether two points lie in the same basin, specifically to check: (a) whether a termination point is already present in the termination point log (Section 2.4), or (b) whether a potential live seed has fallen in the same basin as a candidate global optimum that has been found at least twice (Section 2.5). TDME's hill-valley test is similar to HillVallEA's (Algorithm 1 of Maree et al., 2018b). In both tests, the function is evaluated at a number of regularly spaced locations along the straight line segment that joins the two points in question. HillVallEA then considers the two points to be in the same basin if the function value is higher at either of those two points than it is everywhere between them. Thus HillVallEA may still consider the two points to be in the same basin even if there are multiple ups and downs of the function value (suggesting multiple basins) between them. In contrast, TDME considers the two points to be in the same basin only if the function value increases monotonically on both sides of its minimum location along the line segment (or on one side of it, if it is one of the two end points). Thus, TDME is stricter about what is considered the same basin. TDME evaluates the function at $D + 1$ locations along the line segment, plus the two end points.

3 Evaluation Methods

3.1 Performance Metrics

As the goal of multimodal optimization is to find as many global (or otherwise desirable) optima as possible, it is standard practice to quantify the performance of multimodal optimizers in terms of the peak ratio (PR), which is the number of global optima

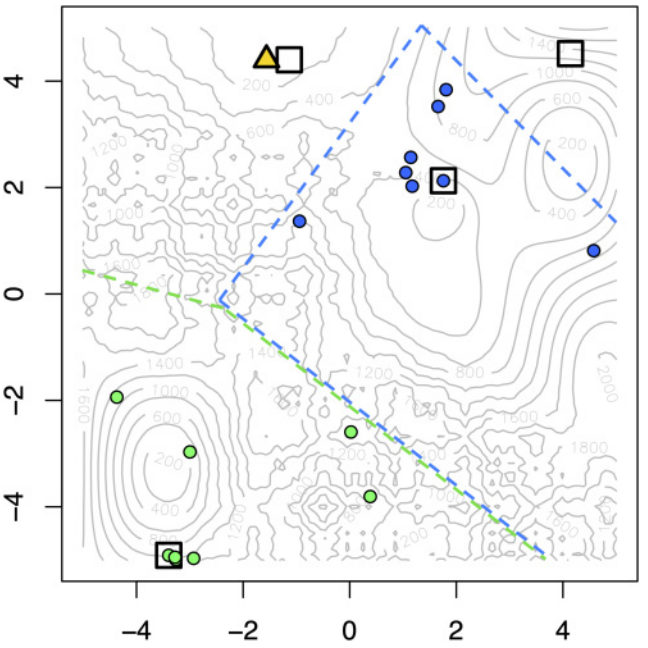

Figure 1: A snapshot of two live TDME species immediately after seeding, on a contour plot of CF1 in 2D. Individuals are represented as circles color-coded by species and seeds are represented as hollow black squares. One seed produced the blue species, one produced the green species, and two became ghost seeds for this cohort because they were in the same valley as a candidate global optimum that had already been converged on twice, which is represented by the yellow triangle. Territory borders for the live species are represented by color-coded dashed lines.

found as a fraction of the test function's true number of global optima. The focus on global optima obviates the need for a threshold that would identify optima as desirable. To enable direct comparison to previous work, we follow the CEC and GECCO MMO competitions in reporting peak ratios achieved within predetermined computational budgets, which are specified in terms of function evaluations (FEs). FEs are the currency of computational effort because the computational cost of the optimization code is negligible compared to that of the function code in most real-world applications (but not usually for test functions). The computational budgets specified by the CEC and GECCO competitions and used for all problems in this work are: 200,000 FEs when $D \leq 2$ and 400,000 FEs otherwise. We also use the competitions' most demanding fitness tolerance: a search was considered to have found a global optimum when it found a function value within 10^{-5} of the function's true global optimum function value. To avoid repeat counting of global optima, we used HillVallEA's hill-valley test (Algorithm 1 of Maree et al., 2018b) for both HillVallEA and TDME (HillVallEA's test is sufficient in this case because the function value can never drop below the global minimum; see Section 2.6). Because evolutionary algorithms are inherently stochastic, we ran each algorithm on each test problem 50 times and report the mean PR, as required by the MMO competitions.

Alongside the mean PRs, we report the multiplicative factor by which TDME's mean PR was larger than HillVallEA's; that is, the average number of global optima

Algorithm 6: Hill-valley testing in TDME

```
1  N_t = D + 1;
2  a = a vector of D elements giving the location of the first point in question;
3  b = a vector of D elements giving the location of the second point in question;
4  L = an (N_t + 2) × D matrix of which row L_k is the location of the k^th
     evaluation point along the straight line segment that joins points a and b;
5  for k = 1, ..., N_t + 2 do
6  |  L_k = a + ((k−1)/(N_t+1))(b − a);
7  end
8  F = f(L) where f is the function to evaluate;
9  G = a vector whose i^th element is the index of the i^th lowest value in F;
10 low block leftmost position = low block rightmost position = G_1;
11 same basin = TRUE;
12 i = 2;
13 while i ≤ N_t + 2 and same basin = TRUE do
14 |  next position to the left = low block leftmost position - 1;
15 |  next position to the right = low block rightmost position + 1;
16 |  if G_i = next position to the left then
17 |  |  low block leftmost position = next position to the left;
18 |  else
19 |  |  if G_i = next position to the right then
20 |  |  |  low block rightmost position = next position to the right;
21 |  |  else
22 |  |  |  same basin = FALSE;
23 |  |  end
24 |  end
25 |  i = i + 1;
26 end
```

found by TDME per global optimum found by HillVallEA. This PR factor is what matters most when comparing two MMOs on a benchmark whose FE budget is arbitrary. A factor of 1.5, for example, indicates that TDME was 50% more efficient than HillVallEA: it found 50% more global optima using the same amount of computation.

To elaborate the performance comparison on problems where both MMOs achieved the same mean PR, we also report the percent of runs that found all the global optima within the budget constraint, and the median number of function evaluations required for those runs to complete (Section 4.4).

When comparing TDME to unimodal DE optimizers on problems with only one global optimum, we quantify performance by the number of FEs required to get within 10^{-5} of the global optimum function value, averaged across 50 runs, because that is the measure reported by the original studies of those DE optimizers. A more comprehensive comparison of TDME with a wide range of unimodal optimizers is beyond the scope of this article but could be a focus for future work; for example, using the COCO black-box benchmarking suite (Hansen et al., 2021). Note that TDME is not intended to outcompete the best unimodal optimizers on problems with only one global optimum; it is intended to compare reasonably well against unimodal optimizers on such problems, but chiefly to optimize the kinds of problem that unimodal optimizers cannot.

Table 1: Test problems used in this work.

Function	D	Domain Bounds[1]	Number of Optima: Global	Number of Optima: Local	Reference
A. Functions that favor multiple species of minimal size					
Five-Uneven-Peak Trap	1	[0, 30]	2	3	Li et al. (2013)
Equal Maxima	1	[0, 1]	5	0	
Uneven Decreasing Maxima	1	[0, 1]	1	4	
Himmelblau	2	[−6, 6]	4	0	
Six-Hump Camel Back	2	[−2, 2]	2	2	
Shubert	2, 3, 4, 5, 6	[−10, 10]	$D \times 3^D$	Many	
Vincent	2, 3, 4, 5, 6	[0.25, 10]	6^D	0	
Modified Rastrigin	2	[0, 1]	12	0	
Composition Function 1 (CF1)	2, 3, 5, 10, 20	[−5, 5], [−10, 20]	6	Many	
Composition Function 2 (CF2)	2, 3, 5, 10, 20	[−5, 5], [−5, 25]	8	Many	
Composition Function 3 (CF3)	2, 3, 5, 10, 20	[−5, 5], [−25, 5]	6	Many	
Composition Function 4 (CF4)	2, 3, 5, 10, 20	[−5, 5], [−25, 5]	8	Many	
B. Functions that favor a single, larger species					
Griewank	2, 5, 10, 20, 50	[−600, 600]	1	Many	Das et al. (2009);
Rosenbrock	2, 5, 10, 20, 50	[−30, 30]	1	0 or 1	Qin et al. (2009);
Schwefel	2, 5, 10, 20, 30, 50	[−500, 500]	1	Many	Li and Zhang
RRS[2] Ackley	10, 30	[−32, 32]	1	Many	(2011)
RRS[2] Rastrigin	10	[−5.12, 5.12]	1	Many	
Schwefel's Problem 2.21	30	[−100, 100]	1	0	
Kowalik	4	[−5, 5]	1	Few	
Shekel 4D-5	4	[0, 10]	1	4	
Shekel 4D-7	4	[0, 10]	1	6	
Shekel 4D-10	4	[0, 10]	1	9	
C. Functions that favor multiple, larger species					
Biased Reflected Griewank	2, 5, 10, 20, 50	[−600, 600]	2	Many	This Work
Biased Reflected Rosenbrock	2, 5, 10, 20, 50	[−30, 30]	2	0 or 2	
Biased Reflected Schwefel	2, 5, 10, 20, 50	[−500, 500]	2	Many	

[1]The format is [lower bound, upper bound]; the same bounds are used in every dimension.
[2]Randomly rotated and shifted.

In our tables, results that were superior to another optimizer by more than the precision of the reported values are highlighted in bold. The standard error in the PR factor for 50 trials on an individual problem was typically about 1%—much smaller than the performance discrepancies we are concerned with in this article. For sets of 15 or more problems (e.g., Category A, B, or C), the standard error was never more than about 0.1%. In terms of establishing confidence in the relative performance of two optimizers across a variety of problems, the selection of particular benchmark problems is a greater source of uncertainty than is random variation among repeated trials. Focusing on the statistics of repeated trials would therefore give a formal but misleading characterization of the robustness of the performance comparison.

3.2 Benchmark Functions

The benchmark functions used in this work are described in Table 1, where they are separated into three categories according to which resource allocation strategy they

favor: (A) "niching-oriented" functions, which favor multiple species of minimal size; (B) "evolution-oriented" functions, which favor a single, larger species; and (C) "dual-oriented" functions, which favor multiple, larger species. This categorization is based on the results of our testing of numerous variations of TDME.

The niching-oriented functions (Category A) are those that were explored most effectively by separating the population into many small, reproductively isolated subpopulations. The effectiveness of exploration thus depended chiefly on the niching mechanism. The 12 Category A functions in Table 1 are the same 12 functions that are used in the 20 problems of the CEC benchmark. The composition functions CF1–CF4 each incorporate several elementary functions as described in Li et al. (2013). For those composition functions, we tested both the original problems and modified versions designed to increase their real-world relevance as discussed in Section 1. In particular, we: (1) tested all the composition functions in 2D, 3D, 5D, 10D, and 20D, rather than limiting some to lower D and (2) expanded their domains by a factor of three in each dimension. The domains were expanded without incorporating new global optima by changing the bounds from $[-5, +5]$ to $[-10, +20]$ on CF1, to $[-5, +25]$ on CF2, and to $[-25, +5]$ on CF3 and CF4. Expanding the domains of the other challenging functions with multiple global optima (Shubert, Vincent) would have added more global optima and defeated the purpose of this test, which was to concentrate the global optima in one region of the domain.

The evolution-oriented functions (Category B) are those that were explored most effectively by consolidating the population into one large breeding group. The effectiveness of exploration thus depended entirely on the evolutionary mechanism. These functions tended to be challenging functions with one global optimum (whereas easy functions with one global optimum can be solved by a single species of minimal size, which is the trivial scenario at the intersection of Categories A and B). The ten Category B functions in Table 1 are all standard test functions from the set used to compare unimodal DE optimizers in Li and Zhang (2011), which was similar to the set used for the same purpose in Das et al. (2009) and Qin et al. (2009). We chose these particular functions so that we could compare TDME against three advanced unimodal DE variants from the literature, and thereby investigate whether TDME's ability to navigate misleading landscapes and to find multiple global optima comes at an acceptable cost in terms of its performance on simpler problems. As noted in Section 3.1, a more comprehensive evaluation of TDME's performance on problems with only one global optimum is beyond the scope of this paper.

Our 86-problem MMO benchmark for comparison to HillVallEA includes three of the ten functions in part B of Table 1: the Griewank, Rosenbrock, and Schwefel functions, which are among those most commonly used to evaluate unimodal optimizers, and which each present a completely different topographical challenge. Griewank consists of many small basins embedded into the slopes of a single large basin, such that it appears unimodal at large scale but multimodal at small scale. Rosenbrock consists of a single curved trench that is easy to find but hard to navigate, with the global optimum somewhere along the bottom. Rosenbrock's trench is unique among the functions we assembled but ought to represent a common real-world scenario: the optimization of an under-constrained model, in which two or more parameters can trade off against one another without much changing the fitness. Schwefel consists of many competitive local optima in an irregular quasi-array, with the global optimum near one corner of the domain. The remaining Category B functions were less challenging and were used only to compare TDME and rcDE to unimodal optimizers.

The dual-oriented functions (Category C) are those that were explored most effectively by separating the population into a moderate number of moderately sized, reproductively isolated subpopulations. The effectiveness of exploration thus depended on both the niching mechanism and the evolutionary mechanism. The three Category C functions in Table 1 are of our own design because we did not come across any standard test functions in this category—although the real-world model optimization problem that inspired the development of TDME is in this category, and many other real-world problems could be as well. The Biased Reflected Griewank is formed by translating the optima of two identical standard Griewank functions from (0, 0) to (300, 300), reflecting one of them through the origin, multiplying them together, and then doubling the function value in the all-positive sector of the domain (i.e., the region of the domain in which every coordinate is greater than zero). The Biased Reflected Rosenbrock is formed in the same way, by translating the optima of two identical standard Rosenbrock functions from (1, 1) to (11, 11), reflecting one of them through the origin, multiplying them together, and then doubling the function value in the all-positive sector of the domain. The Biased Reflected Schwefel is formed in a slightly different way, by offsetting the fitness of two identical standard Schwefel functions so that the minimum fitness is zero for both (not translating them), reflecting one of them through the origin, multiplying them together, and then doubling the function value in the all-positive sector of the domain. These reflected functions each have two global optima (one in the all-positive sector and one in the all-negative sector) but largely retain the challenging and contrasting topographical qualities of the standard Griewank, Schwefel, and Rosenbrock functions. The doubling of the function value in the all-positive sector thwarts unimodal optimizers without changing the general character of the landscape: it biases searches against that sector, so that even with a large number of restarts, unimodal optimizers consistently fail to discover the global optimum there.

3.3 Comparison to Previous Algorithms

The creators of HillVallEA tested it on the CEC benchmark (Maree et al., 2019) and archived the C++ code for doing so on GitHub (https://github.com/scmaree/HillVallEA). We downloaded and ran that code in order to produce the results shown for HillVallEA on the CEC benchmark in Section 4.1 (thereby confirming the results reported in Maree et al., 2019). We then modified the code to incorporate our additional test problems and to output the median number of FEs used by runs that found all global optima. Given the modular nature of the code, those modifications were straightforward and did not affect the HillVallEA algorithm itself. The only functions that needed to be added were Schwefel and the Biased Reflected functions, each of which amounted to one or two equations and roughly half a dozen lines of code. The dimensionality, lower search bound, upper search bound, number of global optima to find, and true global optimum fitness were all easily altered input parameters in the original code.

However, when making the above modifications, we discovered a problem with the original code that needed to be fixed. Ostensibly, the code was supposed to monitor the number of distinct global optima found so far while the search progressed so that it could stop the search if they had all been found, to avoid wasting computation time. In reality, the code only checked the number of distinct global optima that had been found after the search had already ended. During the search, instead, the code simply assumed that any optima that were tied for best fitness were distinct global optima. That assumption was valid for HillVallEA on the original CEC benchmark, but we did not know whether it would hold for our additional test problems. We therefore modified

Table 2: Mean peak ratios (%) across 50 runs on the problems of the CEC benchmark.

Problem Number	Function	D	HillVallEA	TDME	Factor
1	Five-Uneven-Peak Trap	1	100	100	1.00
2	Equal Maxima	1	100	100	1.00
3	Uneven Decreasing Maxima	1	100	100	1.00
4	Himmelblau	2	100	100	1.00
5	Six-Hump Camel Back	2	100	100	1.00
6	Shubert	2	100	100	1.00
7	Vincent	2	100	100	1.00
8	Shubert	3	**97**	88	0.91
9	Vincent	3	97	96	0.99
10	Modified Rastrigin	2	100	100	1.00
11	CF1	2	100	100	1.00
12	CF2	2	100	100	1.00
13	CF3	2	100	100	1.00
14	CF3	3	92	**94**	1.02
15	CF4	3	75	75	1.00
16	CF3	5	67	**77**	1.15
17	CF4	5	75	74	0.99
18	CF3	10	67	67	1.00
19	CF4	10	60	59	0.98
20	CF4	20	48	48	1.00
		Mean	88.9	88.9	1.00

the code to monitor the number of distinct global optima found so far while the search progressed, and to record the number of FEs performed so far at the moment when the final global optimum was found.

We did not alter or tune the HillVallEA algorithm itself in any way. While it is possible that parameter tuning could improve HillVallEA's performance on the Category B and C problems in our extended test suite, any such improvement would almost certainly come at the cost of degraded performance on the Category A problems that HillVallEA was tuned for by its creators.

The results for unimodal optimizers in Section 4.2 are quoted directly from their original publications.

4 Results and Discussion

4.1 Niching-Oriented Functions

Recall that Category A functions favor exploration by niching and therefore small NP, and are exemplified by the CEC benchmark. TDME exactly matched HillVallEA's score on that benchmark (see Table 2), achieving a mean peak ratio of 88.9%, which is 4% better than the next best MMO, RS-CMSA (Ahrari et al., 2017; Maree et al., 2019). HillVallEA outperformed TDME by more than 1% on only one of the 20 CEC benchmark problems, and TDME outperformed HillVallEA by more than 1% on only two. The CEC benchmark is not the whole story of Category A, however, because many real-world optimization problems are high-dimensional with poorly chosen search bounds, whereas the CEC benchmark consists mostly of low-dimensional problems with well-chosen search bounds.

Table 3: Mean peak ratios (%) across 50 runs on CF1–CF4 with original and expanded domains and additional dimensionalities.

		Original Domain			Expanded Domain		
Function	D	HillVallEA	TDME	Factor	HillVallEA	TDME	Factor
CF1	2	100	100	1.00	100	100	1.00
	3	100	100	1.00	99	100	1.01
	5	100	100	1.00	42	**44**	1.05
	10	73	72	0.99	**33**	29	0.88
	20	42	43	1.02	16	15	0.94
CF2	2	100	100	1.00	99	100	1.01
	3	89	**98**	1.10	81	**97**	1.20
	5	72	**88**	1.22	53	**74**	1.40
	10	**76**	72	0.95	29	**44**	1.52
	20	**75**	58	0.77	14	**23**	1.64
CF3	2	100	100	1.00	**87**	83	0.95
	3	92	**94**	1.02	**67**	64	0.96
	5	67	**77**	1.15	42	**56**	1.33
	10	67	67	1.00	1	1	1.00
	20	61	61	1.00	0	0	—
CF4	2	76	77	1.01	75	76	1.01
	3	75	75	1.00	75	75	1.00
	5	75	74	0.99	49	48	0.98
	10	60	59	0.98	14	**25**	1.79
	20	48	48	1.00	11	**19**	1.73
	Mean	77.4	**78.2**	1.01	49.4	**53.7**	1.09

We explored the impact of the search bounds by expanding the domains of the composition functions in a way that did not introduce any additional global optima (see Section 3.2). With the original bounds, TDME and HillVallEA performed slightly differently on some of the individual problems but achieved nearly identical peak ratios on most of the individual problems and on the problem set as a whole (see Table 3). Domain expansion favored TDME by a factor of 1.09 overall but impacted each function differently, presumably due to differences in their topographical details. On CF4, for example, the extended portion of the domain was essentially a giant basin with the original domain at its center, where TDME's territorial meta-evolution soon concentrated its species. On CF1, in contrast, the extended portion of the domain was a regular array of local optima, well suited to hill-valley clustering.

Higher dimensionality did not confer a consistent advantage on either optimizer across the problems in Category A. On the Shubert and Vincent functions in particular, though, there was a clear pattern: the higher the dimensionality (and therefore the greater the number of global optima for these two functions), the better TDME performed relative to HillVallEA (see Table 4). 3D Shubert was the problem in the CEC benchmark that most favored HillVallEA, but when $D > 3$, TDME outperformed HillVallEA on Shubert by a large factor that increased with D. A similar but muted pattern

Table 4: Mean peak ratios (%) across 50 runs on Shubert and Vincent with increasing dimensionality.

Function	D	Number of Global Optima	HillVallEA	TDME	Factor
Shubert	4	324	31	**35**	1.12
	5	1215	4.8	**7.1**	1.48
	6	4374	0.3	**1.4**	4.56
Vincent	4	1296	**28**	21	0.76
	5	7776	**3.62**	3.30	0.91
	6	46656	**0.493**	0.475	0.96
		Mean	11.5	11.4	1.00

Table 5: Mean peak ratios (%) across 50 runs on three test functions from Category B.

Function	D	HillVallEA	TDME	Factor
Griewank	2	100	100	1.00
	5	0	**100**	∞
	10	10	**100**	10.0
	20	92	**100**	1.09
	50	96	**100**	1.04
Rosenbrock	2	100	100	1.00
	5	60	**100**	1.67
	10	88	**100**	1.14
	20	0	**100**	∞
	50	0	0	—
Schwefel	2	100	100	1.00
	5	86	**100**	1.16
	10	0	**100**	∞
	20	0	**98**	∞
	50	0	**92**	∞
	Mean	48.8	**92.7**	1.90

occurred for Vincent when $D > 3$: TDME gradually caught up to HillVallEA and the two became closely matched at $D = 6$.

4.2 Evolution-Oriented Functions

A critical performance contrast between TDME and HillVallEA was evident on the Category B functions, which favor exploration by within-species evolution over exploration by niching. TDME was always the better performer on these functions, achieving a mean peak ratio of 92.7% while HillVallEA achieved only 48.8% (see Table 5). Moreover, Category B contained several problems on which TDME never failed but HillVallEA never succeeded, given the fixed FE budget. As one cannot know how many

Table 6: The mean number (across 50 runs) of function evaluations required for success on functions with only one global optimum.

		$NP = 50$			$NP = 24$	
Function	D	JADE	SaDE	DE/cluster	rcDE	TDME
Rosenbrock	10	—	**42446**	—	51468	50547
RRS Ackley	10	10939	12224	**7700**	13016	12482
	30	31286	33014	**22664**	71752	69030
RRS Rastrigin	10	—	—	—	**21349**	23699
Schwefel	10	17460	16663	11973	11192	**10877**
	30	108000	**44283**	46303	73188**	93171*
Schwefel's Prob. 2.21	30	165000	**88934**	146000	382085*	363192*
Kowalik	4	—	6426	**5092**	7030	6544
Shekel 4D-5	4	—	4947	2617	2538	**2416**
Shekel 4D-7	4	—	4173	**2040**	2139	2072
Shekel 4D-10	4	—	4267	**1738**	2138	2060

*Some runs exceeded the function evaluation budget of past studies (300,000 FEs).
**4% of runs failed by converging on a local optimum.

global optima a function will have in advance, or what its landscape will be like, it is crucial for an optimizer to be able to handle functions like these three as well as niching-oriented functions like CF1–CF4.

Unimodal optimizers specialize in evolution-oriented functions, but TDME's Category B performance was competitive with three advanced DE-based unimodal optimizers: SaDE (Qin et al., 2009), JADE (Zhang and Sanderson, 2009), and DE/cluster (Li and Zhang, 2011) (see Table 6). Table 6 includes results for rcDE as well, to show that simply by randomizing F and CR and choosing a moderate fixed NP, the original DE/rand/1/bin algorithm can be made to perform roughly as well as more elaborate DE-based algorithms. There are some empty cells in Table 6 not because some algorithms were not tested on all functions, but because Qin et al. (2009) and Li and Zhang (2011) did not report the mean number of FEs required for success if some runs converged on local optima or exceeded their arbitrary function evaluation budget (100,000 function evaluations when $D < 30$ and 300,000 function evaluations when $D = 30$). Table 6 reports values for rcDE and TDME even in those cases but flags them with asterisks. Some functions from the benchmark suite in Qin et al. (2009) and Li and Zhang (2011) were not included in Table 6, either because they were effectively redundant (e.g. the rotated and shifted Ackley function was included but the original Ackley function was not), or because they were not challenging, or because none of the algorithms reliably succeeded within the function evaluation budget.

Note that the counterintuitive increase of peak ratio with D for $D \geq 5$ on the Griewank function in Table 5 was triple checked and is real, although we do not have an explanation for it. A similar pattern is found for the Biased Reflected Griewank in Table 7. The Griewank function seems to become easier to solve, in some sense, as D increases.

4.3 Dual-Oriented Functions

TDME's success on the niching-oriented problems (Section 4.1) came entirely from its niched searches with $NP = 8$. Its initial search with $NP = 24$ slowed TDME down

Table 7: Mean peak ratios (%) across 50 runs on three test functions from Category C.

Function	D	HillVallEA	TDME	Factor
Biased Reflected Griewank	2	100	100	1.00
	5	16	**95**	5.94
	10	29	**85**	2.93
	20	40	**91**	2.28
	50	49	**94**	1.92
Biased Reflected Rosenbrock	2	100	100	1.00
	5	73	**100**	1.37
	10	0	**75**	∞
	20	0	**49**	∞
	50	0	0	—
Biased Reflected Schwefel	2	100	100	1.00
	5	**100**	95	0.95
	10	10	**74**	7.4
	20	0	**43**	∞
	50	0	**3**	∞
	Mean	41.1	**73.6**	1.79

only on those problems. However, that initial search with $NP = 24$ was responsible for TDME's rapid success on the evolution-oriented problems, with which HillVallEA struggled (Section 4.2). TDME's ability to return to $NP = 24$ after the fourth cohort did not help it on niching- or evolution-oriented problems, and would seem to be superfluous judging by the standard test functions that we found in the literature. That is because all of the standard test functions that benefitted from $NP > 8$ had only one global optimum and no systematically misleading topography, so that a single $NP = 24$ search was sufficient. But it is not hard to imagine a landscape with the character of a Category B function and more than one global optimum, so that multiple $NP = 24$ searches would be required. We created the Biased Reflected Griewank, Rosenbrock, and Schwefel functions to represent such "Category C" problems. To efficiently solve problems in all three categories, an optimizer needs to be able to test and choose the best exploration strategy.

TDME accomplishes that goal well, as can be seen by its high, superior peak ratios on the Category C problems (Table 7). TDME was able to find both global optima on all these functions without difficulty, although the FE budget would have to be increased by a factor of two or three to obtain 100% peak ratios in the highest dimensionalities (20D and 50D). HillVallEA, on the other hand, was far from successful on these problems except in the lowest dimensionalities, when it could effectively saturate the search space with sampling points.

4.4 Low-Dimensional Problems

Of the 86 problems in our MMO benchmark suite, 22 were one- or two-dimensional. Such low-dimensional problems are typically quick to solve, and both TDME and HillVallEA easily achieved 100% peak ratios on 18 of them, given the CEC/GECCO computational budget. To better compare the relative performance of the two algorithms when dimensionality was low (or when their peak ratios were tied more generally), Table 8

Table 8: Percent of runs that found all the global optima and median number of function evaluations required for those runs to complete.[1]

		HillVallEA		TDME	
Function[2]	*D*	% of Runs	Median FEs	% of Runs	Median FEs
Five-Uneven-Peak Trap	1	**100**	**148**	100	1976
Equal Maxima	1	**100**	**568**	100	4081
Uneven Decreasing Maxima	1	**100**	**182**	100	408
Himmelblau	2	**100**	**1377**	100	5591
Six-Hump Camel Back	2	**100**	**678**	100	2256
Shubert	2	**100**	**20500**	100	57700
	3	**4**	**331094**	0	—
Vincent	2	**100**	**23696**	100	45721
Modified Rastrigin	2	**100**	**4324**	100	14780
CF1	2	**100**	**6194**	100	10806
	3	**100**	**10534**	100	18531
	5	**100**	**24700**	100	72640
	10	4	284404	**6**	**377469**
CF2	2	**100**	**17310**	100	39271
	3	34	143702	**84**	**84160**
	5	0	—	**24**	**200536**
CF3	2	98	22611	**100**	**23948**
	3	58	220328	**62**	**193008**
CF1 Expanded	2	**100**	**21769**	100	24468
	3	94	123624	**100**	**99345**
CF2 Expanded	2	90	41576	**100**	**27392**
	3	4	161704	**82**	**156476**
	5	0	—	**2**	**288334**
CF3 Expanded	2	**34**	**161371**	22	174531
Griewank	2	100	12190	**100**	**3648**
	5	0	—	**100**	**17808**
	10	10	5649	**100**	**36660**
	20	92	12398	**100**	**39600**
	50	96	33374	**100**	**144816**
Rosenbrock	2	100	8012	**100**	**3240**
	5	60	248773	**100**	**16836**
	10	88	399984	**100**	**49512**
	20	0	—	**100**	**169080**
Schwefel	2	**100**	**837**	100	1272
	5	100	44562	**100**	**4092**
	10	0	—	**100**	**10716**
	20	0	—	**98**	**32832**
	50	0	—	**92**	**194424**
Biased Reflected Griewank	2	100	25714	**100**	**11264**
	5	10	246017	**90**	**99064**
	10	22	109612	**70**	**174597**
	20	0	—	**82**	**108616**
	50	0	—	**88**	**274068**
Biased Reflected Rosenbrock	2	**100**	**15724**	100	19866
	5	46	399983	**100**	**176777**
	10	0	—	**50**	**287760**
Biased Reflected Schwefel	2	**100**	**1586**	100	5836
	5	**100**	**30927**	90	92731
	10	2	374913	**48**	**95876**
	20	0	—	**6**	**252480**

[1]The best result for each problem is in bold (Median FEs break ties in % of Runs).
[2]This table omits problems for which no run of either algorithm found all global optima.

presents two other metrics: (1) the percent of runs that found all the global optima, and (2) the median number of function evaluations required for those runs to complete. Given a fixed computational budget, the algorithm that tends to find more of the desired optima, or that succeeds at finding all the desired optima more often, is more efficient. When two algorithms tie in those regards, we can identify the more efficient one as the one whose successes required fewer function evaluations. The percent of runs that found all the global optima must take precedence over the median number of function evaluations required for those runs to complete, because the latter metric can be strongly biased by the exclusion of failed runs. For example, it would be unfair to compare the efficiency of one algorithm in its best 5 of 50 runs to the efficiency of another algorithm in its best 40 of 50 runs.

The more efficient result for each problem in Table 8 is highlighted in bold. In addition to TDME's superior performance on Category B and C problems, the *D*-dependent pattern from Table 4 reappears here: HillVallEA tends to be more efficient than TDME on 1D and 2D problems, whereas TDME tends to be more efficient in higher dimensionalities. Of the four Category B and C problems in Table 8 that were won by HillVallEA, three were in 2D and one was in 5D. All the 10D, 20D, and 50D problems and almost all the 5D problems were won by TDME. Overall, HillVallEA was the more efficient algorithm on 15 of the 20 1D and 2D problems in Table 8, whereas TDME was the more efficient algorithm on 26 of the 30 problems in higher dimensionalities.

4.5 Comparison Summary

TDME's mean peak ratio exceeded HillVallEA's in all three function categories. On the niching-oriented (Category A) problem set, TDME found 3% more global optima than HillVallEA, while on the evolution- and dual-oriented sets (Categories B and C), it found almost twice as many as HillVallEA. On average across all 86 MMO benchmark problems, TDME found 25% more global optima than HillVallEA (see Table 9). Moreover, TDME's peak ratio was never less than 75% of HillVallEA's on any of the 86 problems, whereas HillVallEA's peak ratio was less than 75% of TDME's on 24 problems and less than 5% of TDME's on 9 problems. A histogram of the factor by which TDME's mean peak ratio exceeded HillVallEA's on the 86 individual comparison problems is shown in Figure 2.

On the other hand, on low-dimensional problems where both HillVallEA and TDME easily found all the global optima within the computational budget, HillVallEA tended to do so with fewer function evaluations. Broadly speaking, HillVallEA was more efficient on 1D and 2D problems while TDME was more efficient when $D \geq 3$.

Table 9: Summary performance comparison of TDME and HillVallEA.

	Peak Ratio (%)			
Problem Set	HillVallEA	TDME	Factor	TDME Wins/Ties/Losses
Original CEC benchmark (Table 2)	88.9	88.9	1.00	2/17/1
Category A (56 problems, Tables 2–4)	64.2	**65.9**	1.03	15/32/9
Category B (15 problems, Table 5)	48.8	**92.7**	1.90	11/4/0
Category C (15 problems, Table 7)	41.1	**73.6**	1.79	10/4/1
All 86 MMO benchmark problems	57.5	**71.9**	1.25	36/40/10

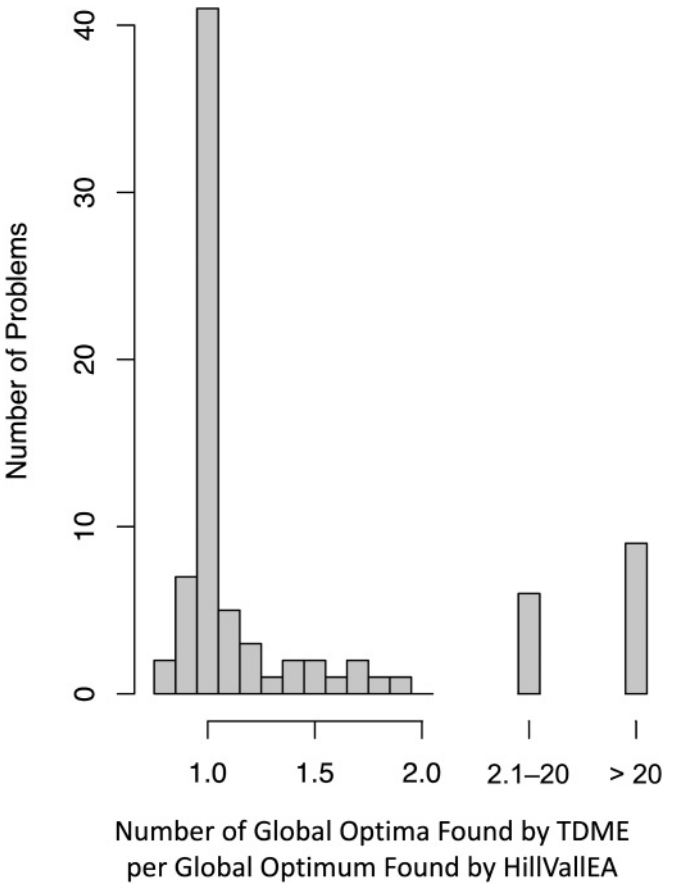

Figure 2: A histogram of the multiplicative factor by which TDME's peak ratio was greater than HillVallEA's on the 86 individual comparison problems, comprising all three function categories.

Notably, TDME was also competitive with unimodal optimizers on the evolution-oriented problems in which they specialized.

5 Conclusion

TDME has been shown to be an exceptionally efficient, reliable, and versatile multimodal optimizer. Compared to the reigning MMO competition champion, HillVallEA, TDME was less efficient on rapidly solved one- and two-dimensional problems but more efficient on higher-dimensional problems. Overall, it matched HillVallEA's score on the competition benchmark and exceeded HillVallEA's score on an expanded 86-problem benchmark that includes more diverse problem types. On problems in more than two dimensions, TDME was usually the better choice and never a poor one, finding at least 75% as many global optima as HillVallEA on every individual problem. In contrast, there were several problems on which HillVallEA found fewer than 5% as many global optima as TDME.

TDME's versatility was evidenced by the fact that it performed on par with HillVallEA on HillVallEA's preferred functions and on par with three unimodal optimizers on their preferred functions. Moreover, TDME was able to optimize several new hybrid test problems much more efficiently than both HillVallEA and the unimodal optimizers. TDME is therefore the algorithm more likely to successfully optimize unknown, real-world problems that might have more than one global (or otherwise desirable) optimum, and it is a promising candidate for further testing and development by the research community.

Acknowledgments

This study was funded by the Genomic Science Program of the United States Department of Energy's Office of Biological and Environmental Research, award #DE-SC0016440, and by the National Science Foundation's Division of Environmental Biology, award #1754803, and through the National Science Foundation's Biology Integration Institutes Program, award #2022070.

References

Ahrari, A., Deb, K., and Preuss, M. (2017). Multimodal optimization by covariance matrix self-adaptation evolution strategy with repelling subpopulations. *Evolutionary Computation*, 25:439–471. 10.1162/evco_a_00182, PubMed: 27070282

Das, S., Abraham, A., Chakraborty, U. K., and Konar, A. (2009). Differential evolution using a neighborhood-based mutation operator. *IEEE Transactions on Evolutionary Computation*, 13(3):526–553. 10.1109/TEVC.2008.2009457

Das, S., Konar, A., and Chakraborty, U. K. (2005). Two improved differential evolution schemes for faster global search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 991–998.

Das, S., Maity, S., Qu, B.-Y., and Suganthan, P. N. (2011). Real-parameter evolutionary multimodal optimization—A survey of the state-of-the-art. *Swarm and Evolutionary Computation*, 1:71–88. 10.1016/j.swevo.2011.05.005

Das, S., Mullick, S. S., and Suganthan, P. N. (2016). Recent advances in differential evolution—An updated survey. *Swarm and Evolutionary Computation*, 27:1–30. 10.1016/j.swevo.2016.01.004

Del Ser, J., Osaba, E., Molina, D., Yang, X.-S., Salcedo-Sanz, S., Camacho, D., Das, S., Suganthan, P. N., Coello, C.A.C., and Herrera, F. (2019). Bio-inspired computation: Where we stand and what's next. *Swarm and Evolutionary Computation*, 48:220–250. 10.1016/j.swevo.2019.04.008

Deng, L., Zhang, L., Sun, H., and Qiao, L. (2019). DSM-DE: A differential evolution with dynamic speciation-based mutation for single-objective optimization. *Memetic Computing*, 12:73–86. 10.1007/s12293-019-00279-0

Epitropakis, M. (2019). GECCO 2019 @ Prague: Competition on niching methods for multimodal optimization. http://www.epitropakis.co.uk/gecco2019/

Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., and Brockhoff, D. (2021). COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144. 10.1080/10556788.2020.1808977

Li, X., Engelbrecht, A., and Epitropakis, M. G. (2013). *Benchmark functions for CEC 2013 special session and competition on niching methods for multimodal function optimization*. Technical Report. Evolutionary Computation and Machine Learning Group, RMIT University.

Li, Y., and Zhang, J. (2011). A new differential evolution algorithm with dynamic population partition and local restart. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1085–1092.

Maree, S. C., Alderliesten, T., and Bosman, P.A.N. (2019). *Benchmarking HillVallEA for the GECCO 2019 competition on multimodal optimization*. Technical Report. Amsterdam UMC.

Maree, S. C., Alderliesten, T., Thierens, D., and Bosman, P.A.N. (2018a). *Benchmarking the hill-valley evolutionary algorithm for the GECCO 2018 competition on niching methods multimodal optimization*. Technical Report. Amsterdam UMC.

Maree, S. C., Alderliesten, T., Thierens, D., and Bosman, P.A.N. (2018b). Real-valued evolutionary multi-modal optimization driven by hill-valley clustering. In *Proceedings of the Genetic and Evolutionary Computation Conference*. arXiv:1810.07085

Molina, D., LaTorre, A., and Herrera, F. (2018). An insight into bio-inspired and evolutionary algorithms for global optimization: Review, analysis, and lessons learnt over a decade of competitions. *Cognitive Computation*, pp. 1–28.

Neri, F., and Tirronen, V. (2009). Recent advances in differential evolution: A survey and experimental analysis. *Artificial Intelligence Review*, 33:61–106. 10.1007/s10462-009-9137-2

Omran, M.G.H., Engelbrecht, A., and Salman, A. (2009). Bare bones differential evolution. *European Journal of Operational Research*, 196(1):128–139. 10.1016/j.ejor.2008.02.035

Price, K., Storn, R. M., and Lampinen, J. A. (2005). *Differential evolution: A practical approach to global optimization*. Springer-Verlag.

Qin, A. K., Huang, V. L., and Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13:398–417. 10.1109/TEVC.2008.927706

Salman, A., Engelbrecht, A. P., and Omran, M. G. H. (2007). Empirical analysis of self-adaptive differential evolution. *European Journal of Operational Research*, 183(2):785–804. 10.1016/j.ejor.2006.10.020

Slowik, A., and Kwasnicka, H. (2020). Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, 32:12363–12379. 10.1007/s00521-020-04832-8

Sreedhar, M., Reddy, S.A.N., Chakra, S. A., Kumar, T. S., Reddy, S. S., and Kumar, B. V. (2020). A review on advanced optimization algorithms in multidisciplinary applications. In *Select Proceedings of ICIME: Recent Trends in Mechanical Engineering*, pp. 745–755.

Storn, R., and Price, K. V. (1995). *Differential evolution—A simple and efficient adaptive scheme for global optimization over continuous spaces*. Technical Report TR- 95-012. ICSI.

Yaman, A., Iacca, G., and Caraffini, F. (2019). A comparison of three differential evolution strategies in terms of early convergence with different population sizes. *AIP Conference Proceedings*, 2070:020002.

Zhang, J., and Sanderson, A. C. (2009). JADE: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13:945–958. 10.1109/TEVC.2009.2014613

Zielinski, K., Weitkemper, P., Laur, R., and Kammeyer, K.-D. (2006). Examination of stopping criteria for differential evolution based on a power allocation problem. *10th International Conference on Optimization of Electrical and Electronic Equipment*, Brasov, Romania.