A Modular Hybridization of Particle Swarm Optimization and Differential Evolution

Rick Boks
Leiden Institute of Advanced Computer Science
Leiden, The Netherlands
r.m.boks@umail.leidenuniv.nl

Hao Wang
LIP6, Sorbonne Université
Paris, France
hao.wang@lip6.fr

Thomas Bäck
Leiden Institute of Advanced Computer Science
Leiden, The Netherlands
t.h.w.baeck@liacs.leidenuniv.nl

ABSTRACT

In swarm intelligence, Particle Swarm Optimization (PSO) and Differential Evolution (DE) have been successfully applied in many optimization tasks, and a large number of variants, where novel algorithm operators or components are implemented, has been introduced to boost the empirical performance. In this paper, we first propose to combine the variants of PSO or DE by modularizing each algorithm and incorporating the variants thereof as different options of the corresponding modules. Then, considering the similarity between the inner workings of PSO and DE, we hybridize the algorithms by creating two populations with variation operators of PSO and DE respectively, and selecting individuals from those two populations. The resulting novel hybridization, called PSODE, encompasses most up-to-date variants from both sides, and more importantly gives rise to an enormous number of unseen swarm algorithms via different instantiations of the modules therein.

In detail, we consider 16 different variation operators originating from existing PSO- and DE algorithms, which, combined with 4 different selection operators, allow the hybridization framework to generate 800 novel algorithms. The resulting set of hybrid algorithms, along with the combined 30 PSO- and DE algorithms that can be generated with the considered operators, is tested on the 24 problems from the well-known COCO/BBOB benchmark suite, across multiple function groups and dimensionalities.

CCS CONCEPTS

• **Computing methodologies → Continuous space search**.

KEYWORDS

Differential Evolution, Particle Swarm Optimization, Metaheuristics, Swarm Intelligence, Hybridization

ACM Reference Format:
Rick Boks, Hao Wang, and Thomas Bäck. 2020. A Modular Hybridization of Particle Swarm Optimization and Differential Evolution. In *Genetic and Evolutionary Computation Conference Companion (GECCO '20 Companion), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3377929.3398123

GECCO '20 Companion, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7127-8/20/07…$15.00
https://doi.org/10.1145/3377929.3398123

1 INTRODUCTION

In this paper, we delve into two naturally-inspired algorithms, Particle Swarm Optimization (PSO) [5] and Differential Evolution (DE) [19] for solving continuous black-box optimization problems $f : \mathbb{R}^n \to \mathbb{R}$, which is subject to minimization without loss of generality. Here we only consider simple box constraints on $\mathbb{R}^n$, meaning the search space is a hyper-box $[\mathbf{x}^{\min}, \mathbf{x}^{\max}] = \prod_{i=1}^{n} [x_i^{\min}, x_i^{\max}]$.

In the literature, a huge number of variants of PSO and DE has been proposed to enhance the empirical performance of the respective algorithms. Despite the empirical success of those variants, we, however, found that most of them only differ from the original PSO/DE in one or two operators (e.g., the crossover), where usually some simple modifications are implemented. Therefore, it is almost natural for us to consider combinations of those variants. Following the so-called configurable CMA-ES approach [22, 23], we first modularize both PSO and DE algorithms, resulting in a modular framework[1] where different types of algorithmic modules are applied sequentially in each generation loop. When incorporating variants into this modular framework, we first identify the modules at which modifications are made in a particular variant, and then treat the modifications as options of the corresponding modules. For instance, the so-called inertia weight [18], that is a simple modification to the velocity update in PSO, shall be considered as an option of the velocity update module.

This treatment allows for combining existing variants of either PSO or DE and generating non-existing algorithmic structures. It, in the loose sense, creates a *space/family of swarm algorithms*, which is configurable via instantiating the modules, and hence potentially primes the application of algorithm selection/configuration [21] to swarm intelligence. More importantly, we also propose a meta-algorithm called **PSODE** that *hybridizes* the variation operators from both PSO and DE, and therefore gives rise to an even larger space of unseen algorithms. By hybridizing PSO and DE, we aim to unify the strengths from both sides, in an attempt to, for instance, improve the population diversity and the convergence rate. On the well-known Black-Box Optimization Benchmark (BBOB) [7] problem set, we extensively tested all combinations of four different velocity updates (PSO), five neighborhood topologies (PSO), two crossover operators (DE), five mutation operators (DE), and four selection operators, leading up to 800 algorithms. We benchmark those algorithms on all 24 test functions from the BBOB problem set and analyze the experimental results using the so-called IOHprofiler [4], to identify algorithms that perform well on (a subset of) the 24 test functions.

[1] The source code is available at https://github.com/rickboks/pso-de-framework.

This paper is organized as follows: Section 2 summarizes the related work. Section 3 reviews the state-of-the-art variants of PSO. Section 4 covers various cutting-edge variants of DE. In Section 5, we describe the novel modular PSODE algorithm. Section 6 specifies the experimental setup on the BBOB problem set. We discuss the experimental results in Section 7 and finally provide, in Section 8, the insights obtained in this paper as well as future directions.

2 RELATED WORK

A hybrid PSO/DE algorithm has been coined previously [25] to improve the population diversity and prevent premature convergence. This is attempted by using the DE mutation instead of the traditional velocity- and position-update to evolve candidate solutions in the PSO algorithm. This mutation is applied to the particle's best-found solution $\mathbf{p}_i$ rather than its current position $\mathbf{x}_i$, resulting in a steady-state strategy. Another approach [8] follows the conventional PSO algorithm, but occasionally applies the DE operator in order to escape local minima. Particles maintain their velocity after being permuted by the DE operator. Other PSO/DE hybrids include a two-phase approach [16] and a Bare-Bones PSO variant based on DE [15], which requires little parameter tuning.

This work follows the approach of the modular and extensible CMA-ES framework proposed in [22], where many *ES-structures* can be instantiated by arbitrarily combining existing variations of the CMA-ES. The authors of this work implement a Genetic Algorithm to efficiently evolve the ES structures, instead of performing an expensive brute force search over all possible combinations of operators.

3 PARTICLE SWARM OPTIMIZATION

As introduced by Eberhart and Kennedy [5], Particle Swarm Optimization (PSO) is an optimization algorithm that mimics the behaviour of a flock of birds foraging for food. A particle in a swarm of size $M \in \mathbb{N}_{>1}$ is associated with three vectors: the current position $\mathbf{x}_i$, velocity $\mathbf{v}_i$, and its previous best position $\mathbf{p}_i$, where $i \in \{1, \ldots, M\}$. After the initialization of $\mathbf{x}_i$ and $\mathbf{v}_i$, where $\mathbf{x}_i$ is initialized randomly and $\mathbf{v}_i$ is set to $\mathbf{0}$, the algorithm iteratively controls the velocity $\mathbf{v}_i$ for each particle (please see the next subsection) and moves the particle $\mathbf{x}_i$ accordingly:

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i \tag{1}$$

To prevent the velocity from exploding, $\mathbf{v}_i$ is kept in the range $[-v_{\max}\mathbf{1}, v_{\max}\mathbf{1}]$ ($\mathbf{1}$ is a $n \times 1$ vector containing all ones). After every position update, the current position is evaluated, $f_i = f(\mathbf{x}_i)$. Here, $\mathbf{p}_i$ stands for the best solution found by $\mathbf{x}_i$ (thus personal best) while $\mathbf{g}_i$ is used to track the best solution found in the *neighborhood* of $\mathbf{x}_i$ (thus global best). Typically, the termination of PSO can be determined by simple termination criteria, such as the depletion of the function evaluation budget, as well as more complicated ones that reply on the convergence behavior, e.g., detecting whether the average distance between particles has gone below a predetermined threshold. The pseudo-code is given in Alg. 1.

Algorithm 1 Original Particle Swarm Optimization

for $i = 1 \rightarrow M$ **do**
  $f_i^{\text{best}} \leftarrow f(\mathbf{x}_i)$
  $\mathbf{x}_i \leftarrow \boldsymbol{U}(\mathbf{x}^{\min}, \mathbf{x}^{\max}), \quad \mathbf{v}_i \leftarrow \mathbf{0}$ ▷ Initialize
end for
while termination criteria are not met **do**
  for $i = 1 \rightarrow M$ **do**
    $f_i \leftarrow f(\mathbf{x}_i)$ ▷ Evaluate
    if $f_i < f_i^{\text{best}}$ **then**
      $\mathbf{p}_i \leftarrow \mathbf{x}_i, \quad f_i^{\text{best}} \leftarrow f_i$ ▷ Update personal best
    end if
    if $f_i < f(\mathbf{g}_i)$ **then**
      $\mathbf{g}_i \leftarrow \mathbf{x}_i$ ▷ Update global best
    end if
    Calculate $\mathbf{v}_i$ according to Eq. (2)
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$ ▷ Update position
  end for
end while

3.1 Velocity Updating Strategies

As proposed in the original paper [5], the velocity vector in *original* PSO is updated as follows:

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \boldsymbol{U}(\mathbf{0}, \phi_1\mathbf{1}) \otimes (\mathbf{p}_i - \mathbf{x}_i) + \boldsymbol{U}(\mathbf{0}, \phi_2\mathbf{1}) \otimes (\mathbf{g}_i - \mathbf{x}_i), \tag{2}$$

where $\boldsymbol{U}(\mathbf{a}, \mathbf{b})$ stands for a continuous uniform random vector with each component distributed uniformly in the range $[a_i, b_i]$, and $\otimes$ is component-wise multiplication. Note that, henceforth the parameter settings such as ϕ_1, ϕ_2 will be specified in the experimentation part (Section 6). As discussed before, velocities resulting from Eq. (2) have to be clamped in range $[-v_{\max}\mathbf{1}, v_{\max}\mathbf{1}]$. Alternatively, the *inertia weight* [18] $\omega \in [0, 1]$ is introduced to moderate the velocity update without using $v_{\max}$:

$$\mathbf{v}_i \leftarrow \omega\mathbf{v}_i + \boldsymbol{U}(\mathbf{0}, \phi_1\mathbf{1}) \otimes (\mathbf{p}_i - \mathbf{x}_i) + \boldsymbol{U}(\mathbf{0}, \phi_2\mathbf{1}) \otimes (\mathbf{g}_i - \mathbf{x}_i). \tag{3}$$

A large value of ω will result in an exploratory search, while a small value leads to a more exploitative behavior. It is suggested to decrease the inertia weight over time as it is desirable to scale down the explorative effect gradually. Here, we consider the inertia method with fixed as well as decreasing weights.

Instead of only being influenced by the best neighbor, the velocity of a particle in the *Fully Informed Particle Swarm* (FIPS) [14] is updated using the best previous positions of *all* its neighbors. The corresponding equation is:

$$\mathbf{v}_i \leftarrow \chi\Big(\mathbf{v}_i + \frac{1}{|N_i|} \sum_{\mathbf{p} \in N_i} \boldsymbol{U}(\mathbf{0}, \phi\mathbf{1}) \otimes (\mathbf{p} - \mathbf{x}_i)\Big), \tag{4}$$

where N_i is the number of neighbors of particle i and $\chi = 2/(\phi - 2 + \sqrt{\phi^2 - 4\phi})$. Finally, the so-called *Bare-Bones* PSO [11] is a completely different approach in the sense that velocities are not used at all and instead every component x_{ij} ($j = 1, \ldots, n$) of position $\mathbf{x}_i$ is sampled from a Gaussian distribution with mean $(p_{ij} + g_{ij})/2$ and variance $|p_{ij} - g_{ij}|$, where p_{ij} and g_{ij} are the jth component of $\mathbf{p}_i$ and $\mathbf{g}_i$, respectively:

$$x_{ij} \sim \mathcal{N}\left((p_{ij} + g_{ij})/2, |p_{ij} - g_{ij}|\right), \quad j = 1, \ldots, n. \tag{5}$$

3.2 Population Topologies

Five different topologies from the literature have been implemented in the framework:

- *lbest* (local best) [5] takes a ring topology and each particle is only influenced by its two adjacent neighbors.
- *gbest* (global best) [5] uses a fully connected graph and thus every particle is influenced by the best particle of the entire swarm.
- In the *Von Neumann topology* [12], particles are arranged in a two-dimensional array and have four neighbors: the ones horizontally and vertically adjacent to them, with toroidal wrapping.
- The *increasing topology* [20] starts with an *lbest* topology and gradually increases the connectivity so that, by the end of the run, the particles are fully connected.
- The *dynamic multi-swarm topology* (DMS-PSO) [13] creates clusters consisting of three particles each, and creates new clusters randomly after every 5 iterations. If the population size is not divisible by three, every cluster has size three, except one, which is of size $3 + (M \mod 3)$.

4 DIFFERENTIAL EVOLUTION

Differential Evolution (DE) is introduced by Storn and Price in 1995 [19] and uses scaled differential vectors between randomly selected individuals for perturbing the population. The pseudo-code of DE is provided in Alg. 3.

After the initialization of the population (please see the next subsection) $P = \{\mathbf{x}_i\}_{i=1}^{M} \subset \mathbb{R}^n$ (M is again the swarm size), for each individual $\mathbf{x}_i$, a donor vector $\mathbf{v}_i$ (a.k.a. mutant) is generated according to:

$$\mathbf{v}_i \leftarrow \mathbf{x}_{r_1} + F \cdot (\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) \tag{6}$$

where three distinct indices $r_1 \neq r_2 \neq r_3 \neq i \in [1..M]$ are chosen uniformly at random (u.a.r.). Here $F \in [0.4, 1]$ is a scalar value called the *mutation rate* and $\mathbf{x}_{r_1}$ is referred as the *base vector*. Afterwards, a *trial vector* $\mathbf{x}'_i$ is created by means of crossover.

In the so-called binomial crossover, each component x'_{ij} ($j = 1, \ldots, n$) of x'_i is copied from v_{ij} with a probability $Cr \in [0, 1]$ (a.k.a. crossover rate), or when j equals an index $j_{\text{rand}} \in [1..n]$ chosen u.a.r.:

$$x'_{ij} \leftarrow \begin{cases} v_{ij} & \text{if } U(0,1) \leq Cr \text{ or } j = j_{\text{rand}} \\ x_{ij} & \text{otherwise} \end{cases} \tag{7}$$

In exponential crossover, two integers $p, q \in \{1, \ldots, n\}$ are chosen. The integer p acts as the starting point where the exchange of components begins, and is chosen uniformly at random. q represents the number of elements that will be inherited from the donor vector, and is chosen using Algorithm 2.

Algorithm 2 Assigning a value to q

```
q ← 0
do
    q ← q + 1
while ((U(0,1) ≤ Cr) and (q ≤ n))
```

The trial vector $\mathbf{x}'_i$ is generated as:

$$x'_{ij} \leftarrow \begin{cases} v_{ij} \text{ for } j = \langle p \rangle_n, \langle p+1 \rangle_n \ldots \langle p+q-1 \rangle_n \\ x_{ij} \text{ for all other } j \in \{1, \ldots, n\} \end{cases} \tag{8}$$

The angular brackets $\langle\rangle_n$ denote the modulo operator with modulus n. Elitism selection is applied between $\mathbf{x}_i$ and $\mathbf{x}'_i$, where the better one is kept for the next iteration.

Algorithm 3 Differential Evolution using Binomial Crossover

```
x_i ← U(x^min, x^max),  i = 1, ..., M.                ▷ Initialize
while termination criteria are not met do
    for i = 1 → M do
        Choose r1 ≠ r2 ≠ r3 ≠ i ∈ [1..M] u.a.r.
        v_i ← x_{r1} + F(x_{r2} − x_{r3})              ▷ Mutate
        Choose j_rand ∈ [1..n] u.a.r.
        for j = 1 → n do
            if U(0,1) ≤ Cr or j = j_rand then
                x'_ij ← v_ij
            else
                x'_ij ← x_ij
            end if
        end for
        if f(x'_i) < f(x_i) then
            x_i ← x'_i                                 ▷ Select
        end if
    end for
end while
```

4.1 Mutation

In addition to the so-called DE/rand/1 mutation operator (Eq. 6), we also consider the following variants:

(1) DE/best/1 [19]: the base vector is chosen as the current best solution in the population $\mathbf{x}_{\text{best}}$:

$$\mathbf{v}_i \leftarrow \mathbf{x}_{\text{best}} + F \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2})$$

(2) DE/best/2 [19]: two differential vectors calculated using four distinct solutions are scaled and combined with the current best solution:

$$\mathbf{v}_i \leftarrow \mathbf{x}_{\text{best}} + F \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2}) + F \cdot (\mathbf{x}_{r_3} - \mathbf{x}_{r_4})$$

(3) DE/Target-to-best/1 [19]: the base vector is chosen as the solution on which the mutation will be applied and the difference from the current best to this solution is used as one of the differential vectors:

$$\mathbf{v}_i \leftarrow \mathbf{x}_i + F \cdot (\mathbf{x}_{\text{best}} - \mathbf{x}_i) + F \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2})$$

(4) Target-to-$p$best/1 [10]: the same as above except that we take instead of the current best a solution $\mathbf{x}^p_{\text{best}}$ that is randomly chosen from the top $100p\%$ solutions in the population with $p \in (0, 1]$.

$$\mathbf{v}_i \leftarrow \mathbf{x}_i + F \cdot (\mathbf{x}^p_{\text{best}} - \mathbf{x}_i) + F \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2})$$

(5) DE/2-Opt/1 [2]:

$$\mathbf{v}_i \leftarrow \begin{cases} \mathbf{v}_i \leftarrow \mathbf{x}_{r_1} + F(\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) & \text{if } f(\mathbf{x}_{r_1}) < f(\mathbf{x}_{r_2}) \\ \mathbf{v}_i \leftarrow \mathbf{x}_{r_2} + F(\mathbf{x}_{r_1} - \mathbf{x}_{r_3}) & \text{otherwise} \end{cases}$$

4.2 Self-Adaptation of Control Parameters

The performance of the DE algorithm is highly dependent on values of the parameters F and Cr, for which the optimal values are in turn dependent on the optimization problem at hand. The self-adaptive DE variant JADE [10] has been proposed in desire to control the parameters in a self-adaptive manner, without intervention of the user. This self-adaptive parameter scheme is used in both DE and hybrid algorithm instances.

5 HYBRIDIZING PSO WITH DE

Here, we propose a hybrid algorithm framework called **PSODE**, that combines the mutation- and crossover operators from DE with the velocity- and position updates from PSO. This implementation allows combinations of all operators mentioned earlier, in a single algorithm, creating the potential for a large number of possible hybrid algorithms. We list the pseudo-code of PSODE in Alg. 4, which works as follows.

(1) The initial population $P_0 = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\}$ (M stands for the swarm size) is sampled uniformly at random in the search space, and the corresponding velocity vectors are initialized to zero (as suggested in [6]).
(2) After evaluating P_0, we create P_1 by applying the PSO position update to each solution in P_0.
(3) Similarly, P_2 is created by applying the DE mutation to each solution in P_0.
(4) Then, a population P_3 of size M is generated by recombining information among the solutions in P_0 and P_2, based on the DE crossover.
(5) Finally, a new population is generated by selecting good solutions from P_0, P_1, and P_3 (please see below).

Four different selection methods are considered in this work, two of which are elitist, and two non-elitist. A problem arises during the selection procedure: solutions from P_3 have undergone the mutation and crossover of DE that alters their positions but ignores the velocity thereof, leading to an unmatched pair of positions and velocities. In this case, the velocities that these particles have inherited from P_0 may no longer be meaningful, potentially breaking down the inner workings of PSO in the next iteration. To solve this issue, we propose to re-compute the velocity vector according to the displacement of a particle resulting from mutation and crossover operators, namely:

$$\mathbf{v}^{(i)} \leftarrow \mathbf{x}^{(i,3)} - \mathbf{x}^{(i,0)}, \text{ for } i = 1, 2, \ldots, M, \tag{9}$$

where $\mathbf{x}^{(i,3)} \in P_3$ is generated by $\mathbf{x}^{(i,0)} \in P_0$ using aforementioned procedure.

A selection operator is required to select particles from P_0, P_1, and P_3 for the next generation. Note that P_2 is not considered in the selection procedure, as the solution vectors in this population were recombined and stored in P_3. We have implemented four different selection methods: two of those methods only consider population P_1, resulting from variation operators of PSO, and population P_3, obtained from variation operators of DE. This type of selection methods is essentially non-elitist allowing for deteriorations. Alternatively, the other two methods implement elitism by additionally taking population P_0 into account.

We use the following naming scheme for the selection methods:

`[comparison method]/[#`P_i `considered]`

Using this scheme, we can distinguish the four selection methods: pairwise/2, pairwise/3, union/2, and union/3. The "pairwise" comparison method means that the i-th members (assuming the solutions are indexed) of each considered population are compared to each other, from which we choose the best one for the next generation. The "union" method selects the best M solutions from the union of the considered populations. Here, a "2" signals the inclusion of two populations, P_1 and P_3, and a "3" indicates the further inclusion of P_0. For example, the pairwise/2 method selects the best individual from each pair of $\mathbf{x}^{(i,1)}$ and $\mathbf{x}^{(i,3)}$, while the union/3 method selects the best M individuals from $P_0 \cup P_1 \cup P_3$.

Algorithm 4 PSODE

Sample $P_0 = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\}$ uniformly at random in $[\mathbf{x}^{\min}, \mathbf{x}^{\max}]$
Initialize velocities $V \leftarrow \{\mathbf{0}, \ldots, \mathbf{0}\}$.
while termination criteria are not met **do**
  $P_1 \leftarrow \emptyset$
  for $\mathbf{x} \in P_0$ with its corresponding velocity $\mathbf{v} \in V$ **do**
    $\mathbf{v}' \leftarrow$ VELOCITY-UPDATE$(\mathbf{x}, \mathbf{v})$
    $\mathbf{x}' \leftarrow \mathbf{x} + \mathbf{v}'$
    Evaluate $\mathbf{x}'$ on f
    $P_1 \leftarrow P_1 \cup \{\mathbf{x}'\}$
  end for
  $P_2 \leftarrow \emptyset$
  for $\mathbf{x} \in P_0$ **do**
    $\mathbf{x}' \leftarrow$ DE-MUTATION$(\mathbf{x})$
    $P_2 \leftarrow P_2 \cup \{\mathbf{x}'\}$
  end for
  $P_3 \leftarrow \emptyset, V \leftarrow \emptyset$
  for $i = 1 \rightarrow M$ **do**
    $\mathbf{x}' \leftarrow$ DE-CROSSOVER$(\mathbf{x}^{(i,0)}, \mathbf{x}^{(i,2)})$
    calculate $\mathbf{v}'$ for $\mathbf{x}'$ using Eq. 9
    Evaluate $\mathbf{x}'$ on f
    $P_3 \leftarrow P_3 \cup \{\mathbf{x}'\}$
    $V \leftarrow V \cup \{\mathbf{v}'\}$
  end for
  $P_0 \leftarrow$ SELECTION(P_0, P_1, P_3)
end while

6 EXPERIMENT

A software framework has been implemented in C++ to generate PSO, DE and PSODE instances from all aforementioned algorithmic modules, e.g. topologies and mutation strategies. Such a framework is tested on IOHprofiler, which contains the 24 functions from BBOB/COCO [7] that are organized in five function groups: 1) Separable functions 2) Functions with low or moderate conditioning 3) Unimodal functions with high conditioning 4) Multi-modal functions with adequate global structure and 5) Multi-modal functions with weak global structure.

In the experiments conducted, a PSODE instance is considered as a combination of five modules: velocity update strategy, population topology, mutation method, crossover method, and selection method. Combining each option for each of these five modules, we obtain a total of 5 (topologies) × 4 (velocity strategies) × 5 (mutation methods) × 2 (crossover methods) × 4 (selection methods) = 800 different PSODE instances.

By combining the 4 velocity update strategies and 5 topologies, we obtain $4 \times 5 = 20$ PSO instances, and similarly we obtain 5 (mutation methods) × 2 (crossover methods) = 10 DE instances.

Naming Convention of Algorithm Instances. As each PSO, DE, and hybrid instance can be specified by the composing modules, it is named using the abbreviations of its modules: hybrid instances are named as follows:

```
H_[velocity strategy]_[topology]_[mutation]
          _[crossover]_[selection]
```

PSO instances are named as:

```
P_[velocity strategy]_[topology]
```

And DE instances are named as:

```
D_[mutation]_[crossover]
```

Options of all modules are listed in Table 1.

Experiment Setup. The following parameters are used throughout the experiment:

- Function evaluation budget: $10^4 n$.
- Population (swarm) size: $5n$ is used for all algorithm instances, due to the relatively consistent performance that instances show across different function groups and dimensionalities when using this value.
- Hyperparameters in PSO: In Eq. (2) and (3), $\phi_1 = \phi_2 = 1.49618$ is taken as recommended in [3] and for FIPS (Eq. (4)), a setting $\phi = 4.1$ is adopted from [14]. In the fixed inertia strategy, ω is set to 0.7298 while in the decreasing inertia strategy, ω is linearly decreased from 0.9 to 0.4. For the Target-to-$p$best/1 mutation scheme, a value of $p = 0.1$ is chosen, following the findings of [10].
- Hyperparameters in DE: F and Cr are managed by the JADE self-adaptation scheme.
- Number of independent runs per function: 30. Note that only one *function instance* (instance "1") is used for each function.
- Performance measure: *expected running time* (ERT) [17], which is the total number of function evaluations an algorithm is expected to use to reach a given target function value for the first time. ERT is defined as $\frac{\#\text{FEs}(f_{\text{target}})}{\#\text{succ}}$, where $\#\text{FEs}(f_{\text{target}})$ denotes the total number of function evaluations taken to hit f_{target} in all runs, while f_{target} might not be reached in every run, and #succ denotes the number of successful runs.

To present the result, we rank the 830 algorithm instances with regard to their ERT values. This is done by first ranking the instances on the targets $f_{\text{opt}} + \{10^1, \ldots, 10^{-8}\}$ of every benchmark function, and then taking the *average rank* across all targets per function. Finally, the presented rank is obtained by taking the average rank over all 24 test functions. This is done for both dimensionalities. A dataset containing the running time for each independent run and ERT's for each algorithm instance, with the supporting scripts, are available at [1].

[velocity strategy]	[mutation]
B – Bare-Bones PSO	B1 – DE/best/1
F – Fully-informed PSO (FIPS)	B2 – DE/best/2
I – Inertia weight	T1 – DE/target-to-best/1
D – Decreasing inertia weight	PB – DE/target-to-*p*best/1
[crossover]	O1 – 2-Opt/1
B – Binomial crossover	**[selection]**
E – Exponential crossover	U2 – Union/2
[topology]	U3 – Union/3
L – $lbest$ (ring)	P2 – Pairwise/2
G – $gbest$ (fully connected)	P3 – Pairwise/3
N – Von Neumann	
I – Increasing connectivity	
M – Dynamic multi-swarm	

Table 1: Module options and codings of velocity strategy, crossover, initialization, topology, and mutation.

7 RESULTS

Figure 1 depicts the Empirical Cumulative Distribution Functions (ECDF) of the top-5 highest ranked algorithm instances in both 5-D and 20-D. Due to overlap, only 8 algorithms are shown. Tables 2 and 3 show the the Estimated Running Times of the 10 highest ranked instances, and the 10 ranked in the middle in 5-D and 20-D, respectively. ERT values are normalized using the corresponding ERT values of the state-of-the-art Covariance Matrix Adaptation Evolution Strategy (CMA-ES).

Though many more PSODE instances were tested, DE instances generally showed the best performance in both 5-D and 20-D. All PSO instances were outperformed by DE and many PSODE instances. This is no complete surprise, as several studies (e.g. in [9, 24]) demonstrated the relative superiority of DE over PSO.

Looking at the ranked algorithm instances, it is clear to see that some modules are more successful than others. The (decreasing) inertia weight velocity update strategies are dominant among the top-performing algorithms, as well as pairwise/3 selection and binomial crossover. Target-to-$p$best/1 mutation is most successful in 5-D while target-to-best/1 seems a better choice in 20-D. This is surprising, as one may expect the less greedy target-to-$p$best/1 mutation to be more beneficial in higher-dimensional search spaces, where it is increasingly difficult to avoid getting stuck in local optima. The best choice of selection method is convincingly pairwise/3. This seems to be one of the most crucial modules for the PSODE algorithm, as most instances with any other selection method show considerably worse performance. This seemingly high importance of an elitist strategy suggests that the algorithm's convergence with non-elitist selection is too slow, which could be due to the application of two different search strategies. The instances H_I_*_PB_B_P3 and H_I_*_T1_B_P3 appear to be the most competitive PSODE instances, with the topology choice having little influence on the observed performance. The most highly ranked DE instances are DE_T1_B

and D_PB_B, in both dimensionalities. Binomial crossover seems superior to the exponential counterpart, especially in 20 dimensions.

Interestingly, the PSODE and PSO algorithms "prefer" different module options. As an example, the Fully Informed Particle Swarm works well on PSO instances, but PSODE instances perform better with the (decreasing) inertia weight. Bare-Bones PSO showed the overall poorest performance of the four velocity update strategies.

Notable is the large performance difference between the worst and best generated algorithm instances. Some combinations of modules, as to be expected while arbitrarily combining operators, show very poor performance, failing to solve even the most trivial problems. This stresses the importance of proper module selection.

8 CONCLUSION AND FUTURE WORK

We implement an extensible and modular hybridization of PSO and DE, called PSODE, in which a large number of variants from both PSO and DE is incorporated as module options. Interestingly, a vast number of unseen swarm algorithms can be easily instantiated from this hybridization, paving the way for designing and selecting appropriate swarm algorithms for specific optimization tasks. In this work, we investigate, on 24 benchmark functions from BBOB, 20 PSO variants, 10 DE variants, and 800 PSODE instances resulting from combining the variants of PSO and DE, where we identify some promising hybrid algorithms that surpass PSO but fail to outperform the best DE variants, on subsets of BBOB problems. Moreover, we obtained insights into suitable combinations of algorithmic modules. Specifically, the efficacy of the target-to-(p)best mutation operators, the (decreasing) inertia weight velocity update strategies, and binomial crossover was demonstrated. On the other hand, some inefficient operators, such as Bare-Bones PSO, were identified. The neighborhood topology appeared to have the least effect on the observed performance of the hybrid algorithm.

The future work lies in extending the hybridization framework. Firstly, we are planning to incorporate the state-of-the-art PSO and DE variants as much as possible. Secondly, we shall explore alternative ways of combining PSO and DE. Lastly, it is worthwhile to consider the problem of selecting a suitable hybrid algorithm for an unseen optimization problem, taking the approach of automated algorithm selection.

ACKNOWLEDGMENTS

Hao Wang acknowledges the support from the Paris Île-de-France Region.

REFERENCES

[1] Rick Boks, Hao Wang, and Thomas Bäck. 2020. Experimental Results for the study "A Modular Hybridization of Particle Swarm Optimization and Differential Evolution". (May 2020). https://doi.org/10.5281/zenodo.3814197

[2] Cheng-Wen Chiang, Wei-Ping Lee, and Jia-Sheng Heh. 2010. A 2-Opt based differential evolution for global optimization. *Applied Soft Computing* 10, 4 (2010), 1200 – 1207. https://doi.org/10.1016/j.asoc.2010.05.012 Optimisation Methods & Applications in Decision-Making Processes.

[3] M. Clerc and J. Kennedy. 2002. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation* 6, 1 (Feb 2002), 58–73. https://doi.org/10.1109/4235.985692

[4] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M Shir, and Thomas Bäck. 2019. Benchmarking discrete optimization heuristics with IOHprofiler. *Applied Soft Computing* (2019), 106027.

[5] R. Eberhart and J. Kennedy. 1995. A New Optimizer Using Particle Swarm Theory. *Proceedings of the sixth international symposium on micro machine and human science* (1995), 39––43.

[6] A. Engelbrecht. 2012. Particle swarm optimization: Velocity initialization. In *2012 IEEE Congress on Evolutionary Computation*. 1–8. https://doi.org/10.1109/CEC.2012.6256112

[7] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *ArXiv e-prints* arXiv:1603.08785 (2016).

[8] Tim Hendtlass. 2001. A Combined Swarm Differential Evolution Algorithm for Optimization Problems. In *Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Engineering of Intelligent Systems (IEA/AIE '01)*. Springer-Verlag, Berlin, Heidelberg, 11–18.

[9] Mahmud Iwan, R. Akmeliawati, Tarig Faisal, and Hayder M.A.A. Al-Assadi. 2012. Performance Comparison of Differential Evolution and Particle Swarm Optimization in Constrained Optimization. *Procedia Engineering* 41 (2012), 1323 – 1328. https://doi.org/10.1016/j.proeng.2012.07.317 International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012).

[10] Jingqiao Zhang and A. C. Sanderson. 2007. JADE: Self-adaptive differential evolution with fast and reliable convergence performance. In *2007 IEEE Congress on Evolutionary Computation*. 2251–2258. https://doi.org/10.1109/CEC.2007.4424751

[11] J. Kennedy. 2003. Bare bones particle swarms. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*. 80–87. https://doi.org/10.1109/SIS.2003.1202251

[12] J. Kennedy and R. Mendes. 2002. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 2. 1671–1676 vol.2. https://doi.org/10.1109/CEC.2002.1004493

[13] J. J. Liang and P. N. Suganthan. 2005. Dynamic multi-swarm particle swarm optimizer. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.* 124–129. https://doi.org/10.1109/SIS.2005.1501611

[14] R. Mendes, J. Kennedy, and J. Neves. 2004. The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation* 8, 3 (June 2004), 204–210. https://doi.org/10.1109/TEVC.2004.826074

[15] M. G. H. Omran, A. P. Engelbrecht, and A. Salman. 2007. Differential Evolution Based Particle Swarm Optimization. In *2007 IEEE Swarm Intelligence Symposium*. 112–119. https://doi.org/10.1109/SIS.2007.368034

[16] M. Pant, R. Thangaraj, C. Grosan, and A. Abraham. 2008. Hybrid Differential Evolution - Particle Swarm Optimization Algorithm for Solving Global Optimization Problems. In *2008 Third International Conference on Digital Information Management*. 18–24. https://doi.org/10.1109/ICDIM.2008.4746766

[17] K. V. Price. 1997. Differential evolution vs. the functions of the 2/sup nd/ ICEO. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*. 153–157. https://doi.org/10.1109/ICEC.1997.592287

[18] Y. Shi and R. Eberhart. 1998. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*. 69–73. https://doi.org/10.1109/ICEC.1998.699146

[19] Rainer Storn and Kenneth Price. 1995. Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces. *Journal of Global Optimization* 23 (01 1995).

[20] P. N. Suganthan. 1999. Particle swarm optimiser with neighbourhood operator. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, Vol. 3. 1958–1962 Vol. 3. https://doi.org/10.1109/CEC.1999.785514

[21] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. Association for Computing Machinery, New York, NY, USA, 847–855. https://doi.org/10.1145/2487575.2487629

[22] S. van Rijn, H. Wang, M. van Leeuwen, and T. Bäck. 2016. Evolving the structure of Evolution Strategies. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. 1–8. https://doi.org/10.1109/SSCI.2016.7850138

[23] Sander van Rijn, Hao Wang, Bas van Stein, and Thomas Bäck. 2017. Algorithm Configuration Data Mining for CMA Evolution Strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. ACM, New York, NY, USA, 737–744. https://doi.org/10.1145/3071178.3071205

[24] J. Vesterstrom and R. Thomsen. 2004. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, Vol. 2. 1980–1987 Vol.2. https://doi.org/10.1109/CEC.2004.1331139

[25] Wen-Jun Zhang and Xiao-Feng Xie. 2003. DEPSO: hybrid particle swarm with differential evolution operator. In *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483)*, Vol. 4. 3816–3821 vol.4. https://doi.org/10.1109/ICSMC.2003.1244483

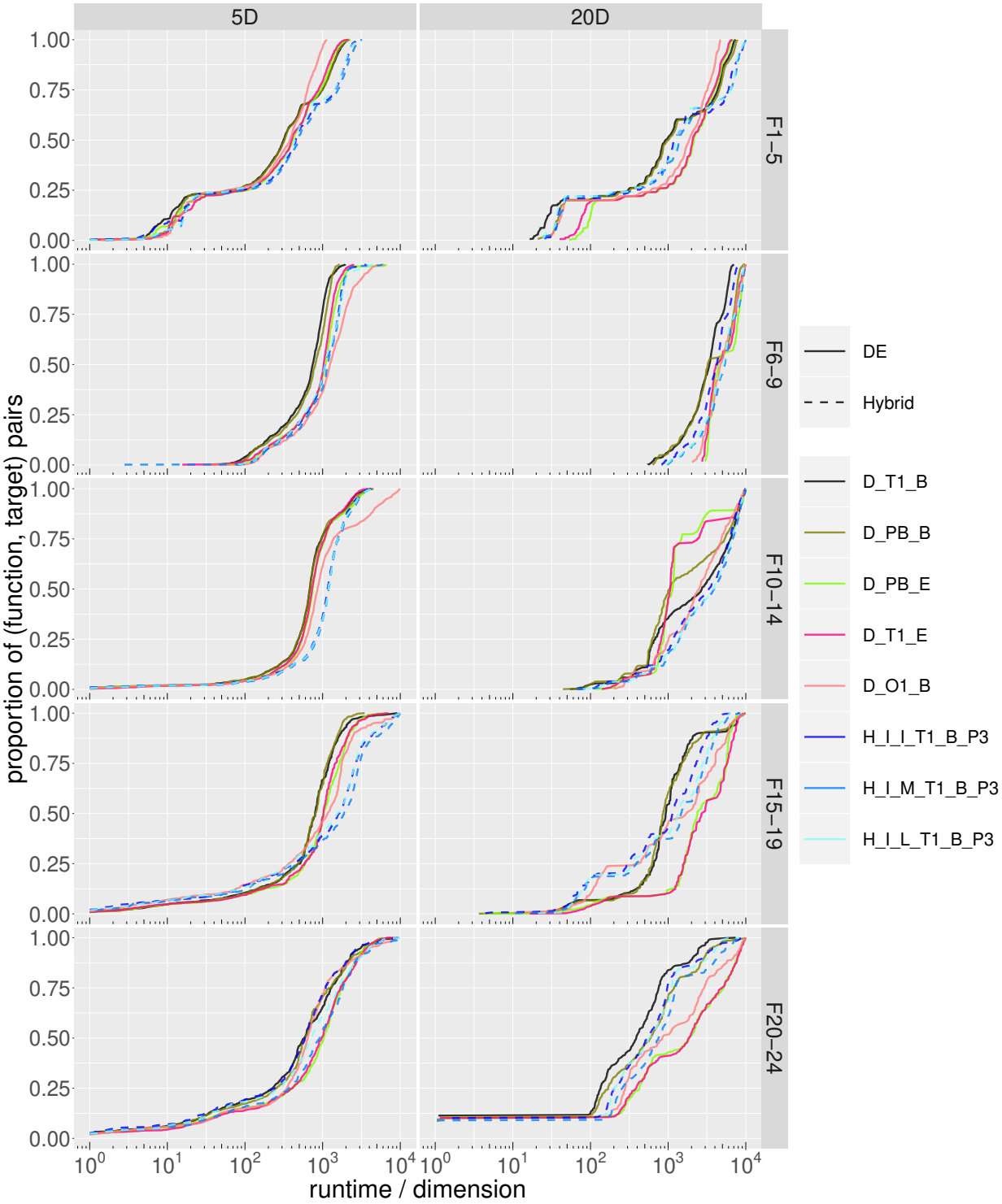

Figure 1: Empirical Cumulative Distribution Functions (ECDFs) of the top-5 ranked algorithms in both $5D$ and $20D$ for each function group defined in BBOB [7]. ECDFs are aggregated over 10 target values $10^{\{1,0,\ldots,-8\}}$ and the ranking is in accordance with Table 2 and 3. Note that only eight algorithms appear here since two algorithms are simultaneously among the top five in both $5D$ and $20D$.

	Algorithm Instance	F1	F2	F6	F8	F11	F12	F17	F18	F21
rank	CMA-ES	658.933	2138.400	1653.667	2834.714	2207.400	5456.867	9248.600	13745.867	74140.538
1	D_T1_B	2.472	1.175	2.261	3.177	1.640	2.362	1.907	9.397	0.592
2	D_PB_B	2.546	1.213	2.321	4.031	1.643	2.580	1.258	5.324	1.072
3	D_PB_E	3.176	1.483	3.635	5.152	1.700	2.750	1.584	4.350	0.305
4	D_T1_E	3.060	1.477	3.583	3.670	1.660	2.281	2.036	9.112	0.352
5	D_O1_B	3.152	1.466	3.717	4.155	6.360	8.818	1.445	8.405	0.383
6	H_I_I_PB_E_P3	3.911	1.830	3.817	3.724	2.951	3.055	3.301	3.021	0.519
7	H_I_I_PB_B_P3	3.685	1.694	3.117	3.115	2.912	3.047	2.102	3.222	1.063
8	H_I_G_PB_B_P3	3.138	1.473	2.813	5.656	2.968	3.099	4.684	3.507	2.251
9	H_I_I_T1_B_P3	3.599	1.700	3.155	5.106	2.837	2.670	2.914	3.975	0.727
10	H_I_N_PB_B_P3	3.480	1.650	3.100	5.061	2.852	2.932	2.453	3.213	1.064
…	…	…	…	…	…	…	…	…	…	…
411	H_I_N_PB_B_P2	4.761	2.268	4.744	12.933	∞	∞	3.113	53.561	2.738
412	H_D_N_T1_E_U3	29.656	38.499	22.459	25.214	5.091	9.053	22.333	8.645	1.247
413	H_B_L_B2_E_U3	25.515	13.345	91.998	10.758	4.203	5.516	16.277	∞	0.960
414	H_F_L_O1_E_U3	19.585	9.980	94.563	18.771	5.529	12.265	161.662	7.416	3.586
415	H_B_G_T1_E_P3	4.736	2.288	6.532	10.503	∞	45.093	2.808	36.108	3.474
416	H_B_N_B1_B_U2	6.531	3.029	8.313	6.918	93.749	13.117	28.817	∞	19.629
417	H_D_I_T1_E_P2	5.506	2.545	5.917	12.812	∞	∞	7.791	34.691	3.433
418	H_D_M_O1_E_U3	21.270	10.963	33.571	12.992	5.882	7.250	12.577	5.760	1.192
419	H_B_G_O1_B_P2	4.091	1.764	4.959	∞	∞	∞	157.845	∞	2.253
420	H_F_L_T1_E_U3	26.450	15.383	17.706	12.174	4.609	9.334	16.892	53.541	1.822

Table 2: On $5D$, the normalized Expected Running Time (ERT) values of the top-10 ranked and 10 algorithms ranked in the middle among all 830 algorithms. The ranking is firstly determining on each test problem with respect to ERT and then averaged over all 24 test problem. For the reported ERT values, the target $f_{\mathbf{opt}} + 10^{-7}$ is used. All ERT values are normalized per problem with respect to a reference CMA-ES, shown in the first row of algorithms.

	Algorithm Instance	F1	F2	F6	F8	F11	F12	F17	F18	F21
rank	CMA-ES	830.800	16498.533	4018.600	19140.467	12212.267	15316.733	5846.400	17472.333	801759
1	D_T1_B	7.377	0.864	5.912	3.702	2.678	4.699	3.144	3.604	0.385
2	D_PB_B	7.731	0.901	6.884	6.766	3.833	5.999	3.158	1.719	0.193
3	H_I_I_T1_B_P3	10.988	1.195	7.894	4.153	6.596	7.656	3.988	3.081	0.298
4	H_I_M_T1_B_P3	12.621	1.434	9.714	5.296	8.389	8.152	4.979	3.138	0.186
5	H_I_L_T1_B_P3	11.402	1.299	9.271	5.146	8.170	7.422	4.771	3.406	0.341
6	H_I_N_T1_B_P3	10.641	1.202	8.218	4.705	7.253	7.928	4.325	2.741	0.338
7	H_D_M_T1_B_P3	12.865	1.476	10.100	6.036	8.119	8.768	5.345	3.450	0.354
8	D_B2_B	7.983	0.885	∞	5.862	10.401	6.455	9.258	44.240	0.829
9	H_D_G_T1_B_P3	9.031	1.074	7.910	4.419	5.690	8.078	4.079	7.838	0.695
10	H_D_N_T1_B_P3	11.307	1.287	9.057	4.801	9.854	5.949	4.517	4.288	0.303
…	…	…	…	…	…	…	…	…	…	…
411	H_D_L_T1_B_U2	39.225	6.262	∞	312.925	∞	35.178	∞	∞	0.728
412	H_F_M_T1_B_U2	55.045	5.655	∞	∞	∞	34.213	∞	∞	0.360
413	H_B_M_T1_E_P2	39.181	4.393	∞	∞	∞	41.771	∞	∞	0.369
414	P_F_N	53.733	∞	1480.838	∞	88.421	∞	∞	∞	0.163
415	H_I_M_T1_B_U2	40.014	7.379	∞	313.468	∞	35.252	∞	∞	0.546
416	H_I_N_PB_B_U3	70.776	362.611	86.426	∞	18.979	∞	339.045	113.442	0.433
417	H_I_M_B1_E_P2	33.073	3.734	∞	∞	∞	72.629	35.327	∞	0.876
418	H_I_G_B2_B_U2	43.424	8.498	∞	104.122	∞	∞	∞	∞	7.367
419	H_B_G_PB_B_U2	41.308	16.007	∞	∞	∞	50.786	∞	∞	1.054
420	H_B_N_PB_B_P3	33.984	4.203	∞	∞	∞	32.929	∞	∞	1.314

Table 3: On $20D$, the normalized Expected Running Time (ERT) values of the top-10 ranked and 10 algorithms ranked in the middle among all 830 algorithms. The ranking is firstly determining on each test problem with respect to ERT and then averaged over all 24 test problem. For the reported ERT values, the target $f_{\mathbf{opt}} + 10^{-1}$ is used. All ERT values are normalized per problem with respect to a reference CMA-ES, shown in the first row of algorithms.