Self-Adapting Particle Swarm Optimization for continuous black box optimization

Michał Okulewicz[a,*], Mateusz Zaborski[a], Jacek Mańdziuk[a,b]

[a] *Warsaw University of Technology, Warsaw, Poland*
[b] *AGH University of Science and Technology, Krakow, Poland*

Abstract

This paper introduces a new version of a hyper-heuristic framework: Generalized Self-Adapting Particle Swarm Optimization with samples archive (M-GAPSO). This framework is based on the authors previous works on hybridization of optimization algorithms and enhancing population based optimization with model based optimization. The paper presents the structure of the proposed framework and analyzes the impact of its modules on the final system performance. M-GAPSO hybridizes Particle Swarm Optimization, Differential Evolution and model based optimizers. A ratio of particular algorithms within a population is regulated by an adaptation scheme. The applicability of the proposed hybrid method to black-box optimization is verified on 24 continuous benchmark functions from the COCO test set and 29 functions from the CEC-2017 test set. On the BBOB test set a hybrid of PSO and DE with adaptation obtained 11 significantly better and 2 significantly worse results on 5 and 20 dimensional functions than the basic DE. Further inclusion of the model based optimizers led to 15 significantly better and 2 significantly worse results compared to the PSO-DE hybrid. On the CEC-2017 test set, M-GAPSO was significantly better than both Red Fox Optimization and Dual Opposition-Based Learning for Differential Evolution (DOBL) on 7 functions in 30 dimensions and 12 functions in 50 dimensions.

Keywords: hyper-heuristics, meta-heuristics, global optimization

*Corresponding author.
Email addresses: M.Okulewicz@mini.pw.edu.pl (Michał Okulewicz), M.Zaborski@mini.pw.edu.pl (Mateusz Zaborski), jacek.mandziuk@pw.edu.pl (Jacek Mańdziuk)

Preprint submitted to Applied Soft Computing 10.1016/j.asoc.2022.109722 December 3, 2022

1. Introduction

The quest for a general purpose optimization algorithms, which started with the works on evolutionary computations [1, 2], resulted in creation of a few excellent optimization methods like Differential Evolution (DE) [3] or Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [4]. Those methods have been thoroughly studied and gradually improved over the years following their initial presentation [5, 6, 7, 8].

Recently, further improvement of metaheuristics within the families of DE, CMA-ES and biologically inspired methods has been observed. Among the DE family of algorithms, a Dual Opposition-Based Learning for Differential Evolution (DOBL) [9] has yielded good results on CEC-2017 data set, while NL-SHADE-RSP presented very good results on CEC-2021 [10]. Within the CMA-ES family, the lq-CMA-ES algorithm [11] has dominated other approaches on the COCO benchmark set, extending previous state-of-the-art version of BIPOP-CMA-ES approach [12] with a surrogate model technique. Likewise, surrogate model extensions can increase performance of a DE-based algorithm when the available number of fitness function evaluations is restricted [13, 14]. Finally, among the recent biologically inspired algorithms, a Red Fox optimization (RFO), which mimics the behavior and hunting patterns of red foxes, is already getting attention [15].

The search for a universal optimization algorithm is a challenging task because optimization performance strongly depends on the type of an optimized function [16]. Additionally, the works on theoretical convergence of random sampling based methods (i.e. Genetic Algorithms (GA) [17], Simulated Annealing (SA) [17] and Particle Swarm Optimization (PSO) [18, 19]) are not necessarily helpful in practical setting of parameters for a particular problem.

One of the methods used in order to escape the problem of overfitting a particular algorithm for a given type of optimization problem is to use a hyper-heuristic approach. Hyper-heuristics have been formally introduced in [20] and further popularized in [21] as "(meta)heuristics working on (meta)heuristics". Initial motivation was to develop a way to effectively select a set of easy to implement heuristics, while maintaining a domain barrier between the selection mechanism and the optimized problem.

In the domain of hyper-heuristics most of the approaches focus on the applications to the discrete domain problems. However, there have been also a few notable approaches for the continuous optimization problems. The work

Table 1 A summary of the related prior contribution of the authors

2016: [26]	Initial concept of generalizing swarm-optimization paradigm within a social simulation scheme
2018: [27]	First version of GAPSO hybridizing and adapting; utilization of various PSO and DE algorithms
2019: [28]	Initial (conference) work on model based optimization approaches within GAPSO framework
2020: [29]	Extended (journal) work on model based optimization approaches within GAPSO framework
2020: [30]	arXiv preprint presenting M-GAPSO framework

[22] is an example of DE application to continuous constrained optimization, where hyper-heuristic uses roulette wheel selection to choose among 12 basic DE variants for the current iteration. The importance of diversity among utilized low-level meta-heuristics such as PSO, DE, GA and CMA-ES is pointed out in [23]. The Artificial Bee Colony (ABC) and Krill Herd (KH) can be utilized as low-level optimizers [24]. The choice of particular optimizer for the agent is done on the basis of the agent rank in terms of its fitness. The work [25] introduce an approach called hyperSPAM that consists of CMA-ES as initial search algorithm and S and Rosenbrock (R) algorithms as the second-phase low-level optimizers during hyper-heuristic search.

The authors propose a hyper-heuristc framework as an extension of their previous works on algorithms hybridization [26, 27, 28, 29]. The first paper discussed the possibility of finding an optimal team of optimization agents based on Belbin's Team Roles Inventory. The second article introduced the initial version of GAPSO algorithm as a hybrid of several variants of the PSO and DE methods. The third and the fourth publications analyzed the impact and behavior of the model-based optimization methods within the M-GAPSO framework. Furthermore, the authors presented the pseudocode and a series of experiments discussed in this work in the arXiv preprint [30]. Table 1 summarizes the authors' prior contribution.

The focus of this paper is to present the design of GAPSO framework enhanced with an archive of function samples (M-GAPSO) and to analyze the impact of its extension by adding a global adaptation scheme. Furthermore, the work advocates the relevance of hybrid optimization methods which above all can gain from synergistic combination of advantages of the compound methods. The proposed M-GAPSO design separates auxil-

iary techniques (e.g. population initialization after algorithm's stagnation) from the actual optimization engine (e.g. PSO's particles sampling strategy). The resulting system consists of multiple groups of components, each serving a particular purpose within the optimization environment. This allows the components of the framework to be developed independently, as easily swapped for different versions in order to analyze their contribution to the overall performance.

1.1. Contribution and organization of the paper

Proposed M-GAPSO framework provides a platform for cooperation of various existing optimization approaches. The main contribution of this paper is threefold:

- Presenting the design of a hyper-heuristic framework in which sampling methods and model-based optimization approaches seamlessly cooperate within a common environment. This work significantly extends the previous version of this framework [27] by adding model based optimization and alternative adaptation mechanism.
- Analyzing the impact (in terms of results quality and computational load) of including an adaptation scheme within the hyper-heuristic environment.
- Improving performance of the base algorithms in terms of the number of fitness function evaluations.

The rest of the paper is organized as follows. Section 2 presents the related literature. Section 3 introduces the principles of M-GAPSO algorithm (taking into account the original GAPSO) and discusses its components. Results on the Comparing Continuous Optimizers in a Black-Box Setting (COCO) benchmark set [31] are presented in Section 4. Conclusions and directions for future work are discussed in Section 5.

2. Related work

This section presents the background of PSO, DE and model based optimization, together with auxiliary elements of optimization methods (i.e. restarts, hybridization and adaptation).

2.1. Particle Swarm Optimization

Particle Swarm Optimization (PSO) algorithm was introduced in [32]. The key concept is to utilize swarm (population) of particles so that they search the space in a certain way. Each particle has its own movement rules enhanced by information derived from its neighbor particles. The particle i has its own velocity vector that changes with each iteration. The velocity vector $\boldsymbol{v}_i^{t+1}$ takes into account: a random component, an inertia effect (velocity vector $\boldsymbol{v}_i^t$ from previous iteration) and two memorized points. The fist memorized point ($\boldsymbol{p}_i^t$) is the best position, according to an objective function, of particle i found until iteration t. While, the second ($\boldsymbol{l}_i^t$) is the best position that comes from whole neighborhood of particle i until iteration t.

The neighbor definition and velocity update formula depend on the specific version of PSO. For one of the popular variants, SPSO-2007 [33], the neighborhood of particle i consist of 3 particles with the following indices: $i-1\ mod(S)$, i, $i+1\ mod(S)$, where S is the swarm size. The exact velocity update formula is presented in the following equation:

$$\boldsymbol{v}_i^{t+1} = \omega \boldsymbol{v}_i^t + U(0, c_1)(\boldsymbol{p}_i^t - \boldsymbol{x}_i^t)) + U(0, c_2)(\boldsymbol{l}_i^t - \boldsymbol{x}_i^t) \tag{1}$$

where: ω is velocity inertia factor, c_1 is cognitive factor, c_2 is social factor, $\boldsymbol{x}_i^t$ is position of a particle in iteration t and $U(0, c_r), r = 1, 2$ are uniform random variables sampled independently in each iteration.

Finally, the position of particle i for iteration $t+1$ (next sampling point) is obtained as follows:

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1} \tag{2}$$

2.2. Differential Evolution

Differential evolution (DE) is a population-based optimization algorithm firstly presented in [3]. Each iteration consist of the three main phases applied to each individual: mutation, crossover and selection. Similarly to PSO, DE has many variants differing mainly in the mutation phase. As an example, DE/best/1/bin variant is described below. In the mutation phase, in iteration $t+1$ (originally generation $G+1$) for each individual $\boldsymbol{x}_i^t$ (parent vector) a mutated vector $\boldsymbol{y}_i^{t+1}$ is generated according to:

$$\boldsymbol{y}_i^{t+1} = \boldsymbol{x}_{best}^t + F(\boldsymbol{x}_{r_1}^t - \boldsymbol{x}_{r_2}^t) \tag{3}$$

where: $F > 0$ is a constant, $\boldsymbol{x}_{best}^t$ is a current best vector and $\{r_1, r_2\}$ are random mutually different indexes of individuals from population.

In the crossover phase the parent vector $\boldsymbol{x}_i^t$ is recombined with the mutated vector $\boldsymbol{y}_i^{t+1}$ so that a trial vector $\boldsymbol{u}_i^{t+1}$ is obtained. Each element d from trial vector $\boldsymbol{u}_i^{t+1}$ comes from parent vector $\boldsymbol{x}_i^t$ or mutated vector $\boldsymbol{y}_i^{t+1}$ according to the assumed probability. One randomly chosen element $u_{i,d_{rand}}^{t+1}$ from trial vector $\boldsymbol{u}_i^{t+1}$ takes the value of $y_{i,d_{rand}}^{t+1}$ to ensure that the trial vector will differ in at least one position from the parent vector $\boldsymbol{x}_i^t$. Formally, each element $u_{i,d}^{t+1}$ from trial vector can be expressed as follows:

$$u_{i,d}^{t+1} = \begin{cases} y_{i,d}^{t+1}, & \text{if } rand_d(0,1) \leq CR \text{ or } d = d_{rand}. \\ x_{i,d}^t, & \text{otherwise,} \end{cases} \tag{4}$$

where: $CR > 0$ is a constant that denotes probability of crossover.

Finally, the trial vector $\boldsymbol{u}_i^{t+1}$ is evaluated on an objective function and selected as a population member instead of parent vector, only if its value is less than or equal to parent's value. Otherwise, the parent vector $\boldsymbol{x}_i^t$ remains unchanged for iteration $t+1$ (please see eq. (5)).

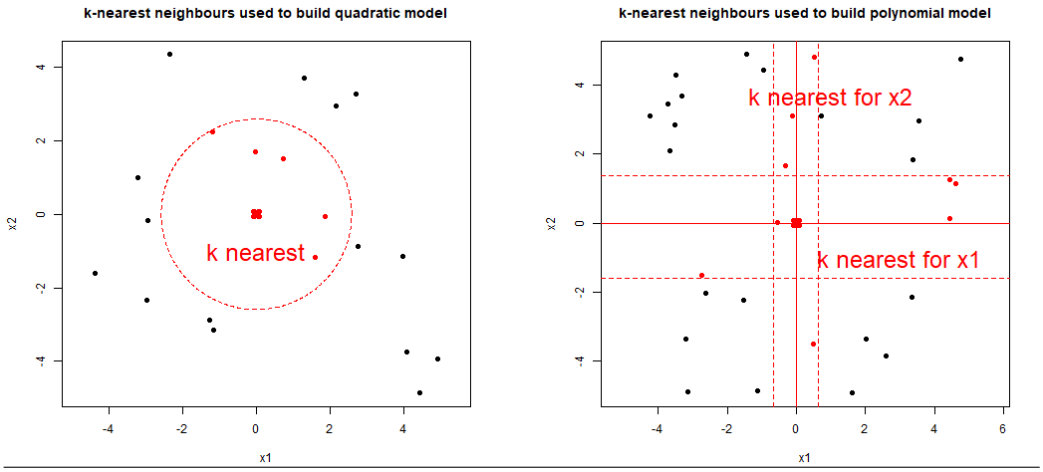
$$\boldsymbol{x}_i^{t+1} = \begin{cases} \boldsymbol{u}_i^{t+1}, & \text{if } f(\boldsymbol{u}_i^{t+1}) \leq f(\boldsymbol{x}_i^t). \\ \boldsymbol{x}_i^t, & \text{otherwise.} \end{cases} \tag{5}$$

2.3. Model-based optimization

The model-based enhancements are described in detail in authors' previous work [29]. In brief summary, two meta-models (quadratic and polynomial) are incorporated into optimization process. They behave like a particle that designates its next sampling position by following a few steps described below.

First, they gather from archive a collection of already evaluated samples. Separate rules are applied to separate meta-models. The quadratic model particle requires k nearest samples in the sense of the Euclidean metric to fit meta-model $\hat{f}_Q(\boldsymbol{x})$. Whereas, the polynomial model particle requires D independent collections of samples for each dimension d of the D dimensions to fit D independent meta-models $\hat{f}_{P_d}(x_d)$. Each collection related with dimension d is composed of k samples that are outspread along dimension d and possibly tight in relation to the rest $D-1$ dimensions. In the other words, k samples that minimizes Euclidean distance metric excluding the dimension d are gathered. The difference of both gathering methods is depicted in Fig. 1. In order to have a more efficient way of finding nearest samples than linear search, all samples in the archive are indexed using an R-Tree [34].

Figure 1 Comparison of samples data sets used for fitting quadratic and polynomial models



Second step is fitting an adequate meta-model. The quadratic and polynomial models are expressed by eqs. (6) and (7), respectively. Ordinary Least Squares method is used for coefficient estimation in both cases.

$$\hat{f}_Q(\boldsymbol{x}) = \sum_{d=1}^{D} \left(a_d x_d^2 + b_d x_d \right) + c \tag{6}$$

$$\hat{f}_{P_d}(x_d) = \sum_{i=1}^{p} a_{i,d} x_d^i + c_d \tag{7}$$

Finally, the optimum of meta-model $\boldsymbol{x}^*$ is designated so the model particle changes its position to this optimum and is evaluated using the true objective function (see eq. (8)).

$$\boldsymbol{x}_{model}^{t+1} = \boldsymbol{x}^* = \arg\min \hat{f}(\boldsymbol{x}) \tag{8}$$

where: $\hat{f}(\boldsymbol{x}) \in \{\hat{f}_Q(\boldsymbol{x}), [\hat{f}_{P_1}(x_1), \ldots, \hat{f}_{P_D}(x_D)]\}$

Calculation of the quadratic model optimum is performed by obtaining a composition of D independent parabola vertices (or a proper boundary points, if necessary) in accordance with $\hat{f}_Q(\boldsymbol{x})$. Alternatively, the polynomial model optimum is obtained using grid search (1000 points) separately for all D coordinates using D independent $\hat{f}_{P_d}(x_d)$ polynomial models.

2.4. Generalizing optimization algorithms

On the subject of generalizing metaheuristic algorithms, the work [35] presented utilization of the term "adaptive memory programming" (AMP), proposing a unified view on several of those optimization methods. AMP initially has been used in connection with Tabu Search [36]. Other metaheuristic algorithms (i.e. Genetic Algorithm, Scatter Search and Ant Systems) follow the same design patterns and their performance depends on the problem structure in a similar way [35]. However, no unified approach has been proposed for a hybrid implementation of those methods.

In the area of hybrid optimization algorithms utilizing PSO and DE, there are a few effective solutions, extending their performance beyond each of the component methods alone. For instance, the work [37] presented a hybrid algorithm combining PSO and DE, motivated by the fact that PSO activity is prone to stagnation when particles are unable to improve their personal best positions. In such a case DE has been applied to update particles best positions and make the swarm jump out of the stagnation phase. Random switching between PSO and DE (with single algorithm applied within a given iteration) was proposed in [38]. Additionally, this method observed the convergence of the population. If the population converged too much, the individuals locations were randomized with the multivariate Gaussian distribution.

PSO can also be hybridized with Genetic Programming [39], Grammatical Evolution [40] or specialized algorithms designed for solving particular optimization problems [41].

2.5. Restarts

Complex and multi-modal functions cause the convergence of an algorithm to local rather than global optimum. Therefore an efficient global optimization method needs to decide when such a convergence occurs and assess if the optimization process should be restarted. One of the widely known restart mechanisms is implemented into JADE optimization algorithm [5]. In a basic JADE approach a spread of population locations is considered and combined with the frequency of global optimum estimation improvements. The following two conditions must occur jointly in order for the restart to be performed. Firstly, a diversity measure $Div(X) = \frac{1}{D} \sum_{d=1}^{D} Var(X_d)$ of population $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ is smaller than a threshold Θ, where N is population

size, D problem size and X_d set of coordinates in dimension d of all population members. Secondly, the objective function value has not been improved in the last T_{max} iterations.

2.6. Algorithms adaptation scheme

One of the methods for improving the algorithm performance is to create a scheme for adjusting algorithm parameters during an optimization process. Those schemes could be applied to the internal control of the algorithm parameters or to select the algorithm to be applied in a hyper-heuristic setting. This section presents selected concepts related to PSO, DE and hybrid algorithms.

There are many variants and modifications of PSO whose performance strictly depends on the optimized problem. A framework PSO-X [42] is designed to facilitate automatic generation of appropriate PSO configurations using training instances and *irace* tool [43]. A configuration, once selected, remains unchanged during the whole optimization run.

An alternative approach is to adapt the algorithm configuration during the optimization process. Such an Adaptive PSO algorithm based on the swarm diameter has been proposed in [44].

The method classifies the observed swarm diameter as one of the four states of the algorithm: *exploration*, *exploitation*, *convergence* and *jumping-out*. If a given state is detected, parameters c_1 and c_2 of the PSO velocity update (eq. (1)) are adjusted accordingly. Overall, the method relies on the particular ease of controlling exploration-exploitation balance within the PSO.

Within the DE family of algorithms the most successful method of parameter adaptation is Success-History based parameter Adaptation for Differential Evolution (SHADE) described in [45]. The method stores several sets of well-performing F and CR parameters controlling the DE's mutation (eq. (3)) and cross-over (eq. (4)). During the optimization process one pair of F_i and CR_i is sequentially selected and the perturbed version of these parameters is used to obtain a new DE sample. Lehmer means [46] calculated for the perturbed parameters, which resulted in improvement of the individual value, replace the old values within the ith slot.

For the case of adaptation within hybrid algorithms the focus is usually on the performance of component algorithms. According to [47], selection of an algorithm can be done in a probabilistic manner by *PM-AdapSS* method. In this method each probability is designated by measuring the impact of given

algorithm k at iteration t understood as a reward value $r_{k,t}$. The reward value takes into consideration performance of all N_k offspring individuals generated by algorithm k and is calculated using the fitness value $f(\boldsymbol{x}_i)$ of ith offspring, the fitness value $f(\boldsymbol{x}_i^{parent})$ of ith offspring's parent and the best solution $f(\boldsymbol{x}_{best})$ found so far:

$$r_{k,t} = \frac{1}{N_k} \sum_{i=1}^{N_k} \frac{f(\boldsymbol{x}_{best}) \cdot \max(f(\boldsymbol{x}_i^{parent}) - f(\boldsymbol{x}_i), 0)}{f(\boldsymbol{x}_i)} \tag{9}$$

Subsequently, $r_{k,t}$ values are used in roulette setup to decide how many individuals would be governed by each evaluated algorithm.

3. Description of M-GAPSO

This section describes the proposed M-GAPSO framework which is based on the PSO algorithm, but allows the usage of virtually any other optimization method whose "particles" act independently. Additionally, the algorithm's restart manager, the archive (external memory) of samples and the algorithm's search space manager are defined as auxiliary modules independent from the core optimization part.

The underlying features of M-GAPSO design can be listed as follows:

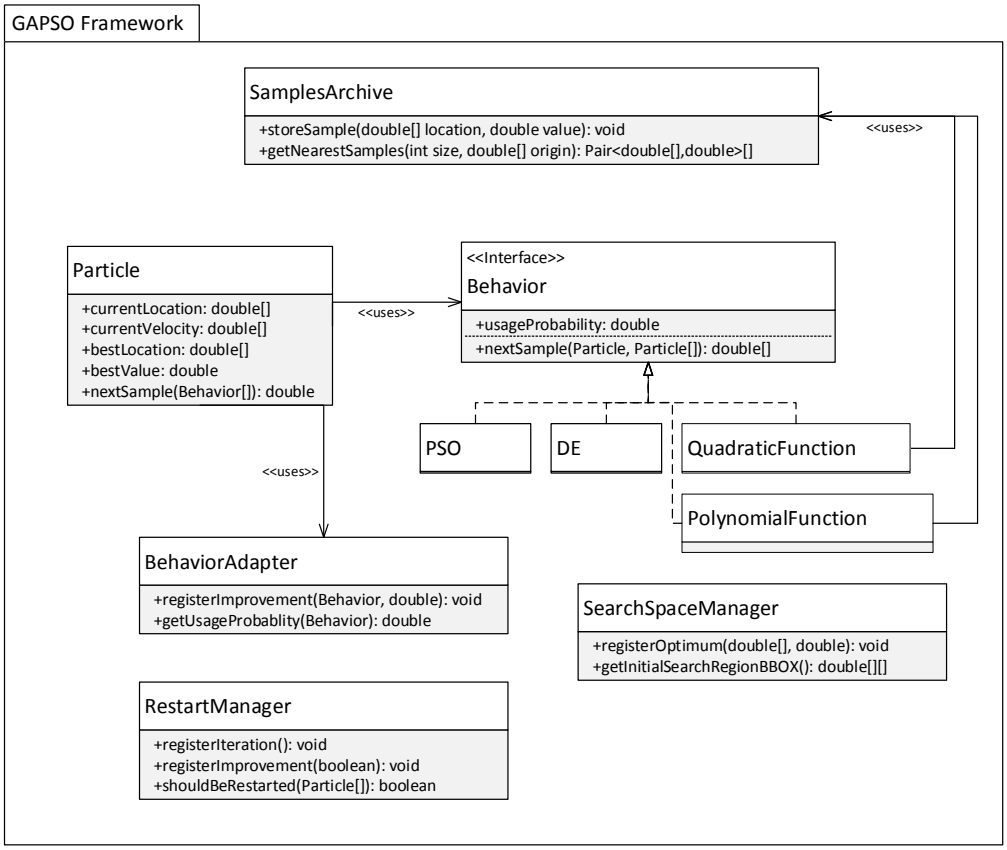
- it relies on well-researched optimization algorithms,
- effectively utilizes samples already gathered by means of randomized search procedures (i.e. PSO, DE),
- guides the search process of the algorithm on the basis of already found local optima,
- keeps independence of the constituting modules to the highest possible extent.

The main components of M-GAPSO are presented in the UML class diagram (Fig. 2) and discussed in detail in the following subsections.

3.1. General swarm–based optimization framework

A starting point for GAPSO was a multi-agent view of the PSO algorithm [26]. In PSO particles can be seen as independent optimization agents,

Figure 2 The UML class diagram of M-GAPSO



each exposing its historically best location only and maintaining its own current location and velocity. This view led to the most important GAPSO design choice: a separation of particles' locations from their velocity update formula. Subsequently, regarding velocity update just as a method of specifying the next location to be sampled by the algorithm, allowed utilization of any population-based self-governing optimization algorithm in GAPSO. Apart from various swarm-based approaches, it also meant the possibility of including other than PSO-like sampling equations (e.g. DE) in GAPSO framework, which has been presented in [27]. Additionally, the pool of available search space samplers (algorithms) has been extended with model-based optimizers in [29]. The above-described design can be observed in the class

diagram (Fig. 2) as *PSO*, *DE*, *Quadratic Function* and *Polynomial Function* classes implement the *Behavior* interface. Objects of classes implementing that interface can be applied to modify the location of the *Particle* on the bases of current state of population and (optionally) data stored in *Samples Archive*.

Application of particular *Behavior* object is managed by the *Behavior Adapter*. This object registers the performance of each of the considered types of behavior and uses it to update the respective probability of selecting a given behavior as a sampling mechanism in future iterations.

M-GAPSO extends the previous version of GAPSO [27] with three main components, that can be implemented and configured independently from the core optimization algorithms:

Samples Archive - serves as a cache and a source of additional information about optimized function.

Restart Manager - observes the population state in terms of values, location and improvement history and decides when the method should be restarted.

Search Space Manager - decides where the algorithm should be initialized within the problem space.

The purpose of the *Samples Archive* is to enable efficient utilization of function values gathered during the solution search process.

The *Restart Manager*, together with the *Search Space Manager*, guide the optimization algorithm towards promising areas of the problem space, when further exploitation of the currently searched area seems to be pointless. With such a design it is possible to extend the GAPSO framework and test the impact of optimization improvements somewhat independently of the main optimization engine.

The M-GAPSO operates similarly to the population based methods like the PSO or DE, as depicted in the Algorithm 1 pseudocode.

Algorithm 1 M-GAPSO high–level pseudocode

F is optimized $\mathbb{R}^D \to \mathbb{R}$ function, $Bounds$ is an $\left(\mathbb{R}^2\right)^D$ vector
$Swarm$ is a set of PSO particles, $Behavior$ is particle's velocity update rule
$Initializer$ is particle's initial location sampler
$SamplesArchive$ is an RTree based samples' index
$BehaviorAdapter$ collects optimum value improvement data
$RestartManager$ observes swarm state and performance
$Bounds \leftarrow f.getBounds()$ ▷ **Initially the whole area is considered**
$PerformanceMonitor.behaviourProbabilities \leftarrow initialProbabilities$
$LocalOptima \leftarrow \emptyset$ ▷ **Set of optima estimations**
while Stopping criterion not met **do**
  for $Particle \in Swarm$ **do**
    $Particle.x \leftarrow Initializer.nextSample(Bounds)$
    $Sample \leftarrow F.evaluate(Particle.x)$
    $BehaviorAdapter.registerValue(Sample)$
  end for
  for $Particle \in Swarm$ **do**
    ▷ **Initial velocity is computed on the basis of two random particles**
    $Particle.v \leftarrow \frac{(Particle_{rand1}.x - Particle_{rand2}.x)}{2.0}$
  end for
  while $RestartManager.shouldOptimizationContinue(Swarm)$ **do**
    for $Particle \in Swarm$ **do**
      ▷ **The number and application order of behaviors is mixed in each iteration**
      $Behaviour \leftarrow BehaviorAdapter.sampleBehaviourPool()$
      $Particle.x \leftarrow Behavior.sampleNext(Particle, Swarm, SamplesArchive)$
      ▷ **Velocity must be managed for the sake of PSO-like behaviors**
      $Particle.v \leftarrow Particle.x - Particle.x_{previous}$
      ▷ **Samples Archive is also used as cache**
      if $SamplesArchive.stored(Particle.x)$ **then**
        $Sample \leftarrow SamplesArchive.retrieve(Particle.x)$
      else
        $Sample \leftarrow F.evaluate(Particle.x)$
        $SamplesArchive.store(Sample)$
      end if
      $BehaviorAdapter.registerImprovement(Sample, Behaviour)$
    end for
    $BehaviorAdapter.recomputeBehaviourProbabilities()$
  end while
  $LocalOptima \leftarrow LocalOptima \cup Swarm.bestSample$
  $Bounds \leftarrow Initializer(LocalOptima)$ ▷ **Guides the search process to the areas expected to uncover new function optima**
end while

At the beginning of the optimization process (and after each restart), the population (including particles' velocity) is initialized randomly over the selected area of search space. The difference is within the sampling procedure where, for the sake of generality, the sampled location is computed first and the velocity is updated as a result (as not all the optimizers utilize the concept of velocity needed by PSO). After each iteration the restart conditions are verified, and if necessary the algorithm is restarted and the found best value is stored in a set of local optima.

On a final note, GAPSO is maintained as an open source application on the MIT license and is available at:
`https://bitbucket.org/pl-edu-pw-mini-optimization/basic-pso-de-hybrid/`.

3.2. Samples archive

In order to store and efficiently retrieve samples, M-GAPSO utilizes a multi-dimensional R-Tree index [34]. Due to performance reasons capacity of the R-Tree index is limited and set by the user. After reaching the maximum capacity, the index is restarted from scratch.

Samples archive is utilized in two scenarios. Mainly, for efficient retrieval of the nearest samples as described in Section 2.3. Subsequently those samples are utilized to fit a quadratic or polynomial function model. Secondly, samples archive serves as a cache memory, so that in the event of the algorithm trying to sample the same location (as it does happen when the swarm has nearly collapsed), it retrieves the function value from memory, saving some budget of the fitness function computations.

3.3. Restart management

M-GAPSO uses an enhanced version of JADE [5] restart manager. In M-GAPSO the *RestartManager* registers iteration count intervals between global optimum updates, considers a spread of personal best locations of particles (eq. (10)) and additionally a spread of personal best locations values (eq. (11)). The last feature was added in order to better handle step functions, where the population spread can be quite large, even though the population reached a sort of a frozen state, with each particle having exactly the same personal best value.

$$\forall_{i,j=1,\ldots,N} \max_{d=1}^{D}(|particle_i.best[d] - particle_j.best[d]|) < \Theta_{locations} \qquad (10)$$

$$\forall_{i,j=1,\ldots,N} |f(particle_i.best) - f(particle_j.best)| < \Theta_{values} \tag{11}$$

Restart will be performed if eq. (10) or eq. (11) is satisfied and the global best value has not been improved for a certain number of evaluations.

3.4. Initialization scheme

The initialization scheme relies on selecting a smaller bounding box as the initial search area on the basis of previously estimated function optima. The process of creating the bounding boxes can be summarized as follows:

1. Restart Manager detects that algorithm is stuck
2. The best sample x found during the last run is selected and considered an estimation of a local optimum location
3. If x is different from all previously found local optima estimations, then
   (a) x is added to the collection of local optima estimations
   (b) The bounding box (hyper-rectangle) B containing this sample is selected from the collection of bounding boxes created so far
   (c) Dimension d in which B has the maximal difference between upper and lower bound is selected
   (d) B is split into B_1 and B_2 by a plain defined by coordinate d of x
   (e) B is removed from the collection of bounding boxes
   (f) B_1 and B_2 are added to the collection of bounding boxes

This way, after each algorithm restart, there is a smaller non-degenerated D-dimensional search space. This encourages the algorithm to explore regions were no significant local optima have been detected so far.

Figure 3 presents a sample step of M-GAPSO re-initialization and multiple local optima estimations as the final result of such a procedure. The initialization procedure includes the following possibilities:

- starting from the original (full) bounding box defined for the optimized function,
- starting from a random bounding box defined by boundaries derived from locations of the local optima estimations,
- starting from a small bounding box centered around the high-quality optimum estimation.

Figure 3 M-GAPSO outer loop sampling scheme. Black dots mark locations of local optima estimations, a black rectangle marks the area for initial swarm location in a given run, and a red rectangle marks the resulting distribution of samples in that run

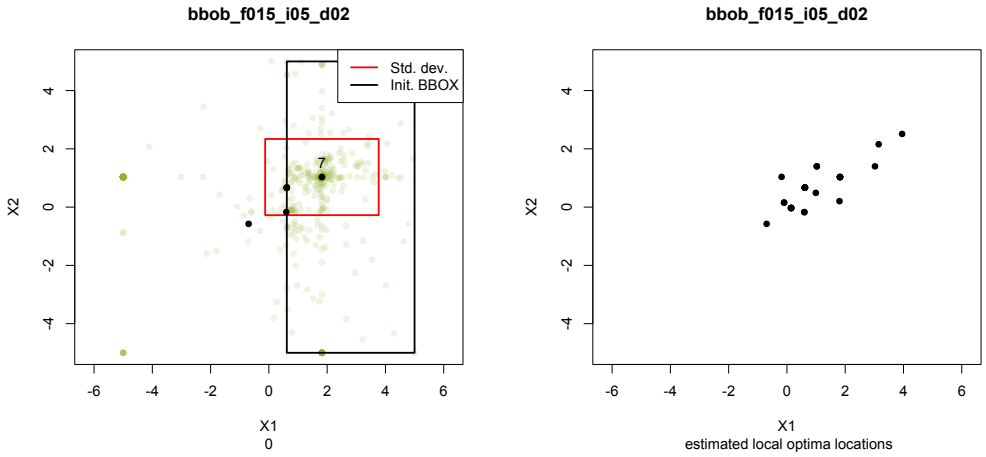
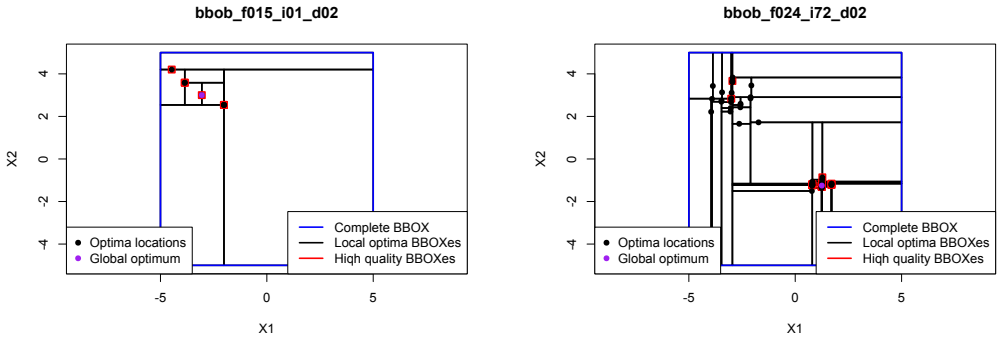


Figure 4 Possible M-GAPSO initialization bounding boxes generated during the optimization process for the functions with (*f15*) and without (*f24*) a general structure



One of those actions is selected at random with probabilities set by the user of M-GAPSO. Figure 4 visualizes possible bounding boxes for initializing the M-GAPSO population, created on the basis of optima estimations during the previous optimization process.

3.5. Optimization behaviors

One of the main conclusions from the authors' initial work on GAPSO [27] was that the highest synergy among optimization behaviors could be observed when DE and PSO are utilized, instead of a pool of various PSO variants (i.e. Standard PSO, Charged PSO, Fully-Informed PSO). Additionally, the authors already tested the ability to improve the results by adding model-based optimizers into the mix [28, 29]. Therefore, the behavior pool implemented in M-GAPSO consists of SPSO-2007, DE/best/1/bin and quadratic and polynomial function model-based optimizers.

In order to ensure generality of the proposed framework particle's location and velocity are managed in the following way:

$$particle.x^{t+1} \leftarrow behavior.sample(particle, swarm, samples.archive) \quad (12)$$

$$particle.v^{t+1} \leftarrow particle.x^{t+1} - particle.x^{t} \quad (13)$$

The sample obtained in eq. (12) comes either from PSO velocity and location update (eqs. (1) and (2)), DE mutation and crossover (eqs. (3) and (4)) or an (estimated) optimum from a fitted function model (eq. (8)). Although some algorithms may have already computed the particles velocity internally, for the sake of uniformity, the velocity is also computed in eq. (13). Observe, that one of the consequences of the above approach is the loss of previously computed velocity, resulting in its reset. However, the new velocity still makes sense from the point of view of SPSO-2007 behavior. If a particle's move has been successful (i.e. $particle.x_{best}$ is updated) and in next iteration PSO behavior is applied to this particle, it will continue to move roughly in the direction that already improved its value - which is the purpose of PSO's velocity. This direction might be perturbed only by the attraction vector of the best neighbor location.

Subsequent steps include:

1. Computing the value for new sample $particle.val_{t+1} \leftarrow f(particle.x_{t+1})$
2. Exchanging $particle.x_{best}$ for $particle.x_{t+1}$ if $particle.val_{t+1}$ is better than $particle.val_{best}$
3. Storing $particle.val_{t+1}$ and $f(particle.x_{t+1})$ in an R-Tree indexed memory (samples archive)

3.6. Adaptation scheme

Adaptation scheme in M-GAPSO for each optimization behavior takes into account its contributions to the improvement of the **global** best value (with respect to that value at the beginning of each iteration). These contributions are aggregated by a moving average scheme (separately for each optimization behavior) and are normalized in order to account for the number of times a given behavior was applied.

In implementing this adaptation scheme the authors have followed one of the methods discussed in [47] presented in Section 2.6. In order to adapt the number of behaviors (algorithms) applied within a single M-GAPSO iteration, proposed method first computes the impact of each behavior b before iteration t, while taking into account past $history.depth$ iterations:

$$w_{b,t} = \frac{\sum_{i=t-history.depth}^{t-1} \left(\sum_{x_{b,i,j} \in \{\text{sampled by behavior } b\}} \max(0, f(g_{best,i}) - f(x_{b,t,j})) \right)}{\sum_{i=t-history.depth}^{t-1} |\{\text{sampled by behavior } b\}|} \tag{14}$$

Therefore, the probability of selecting behavior b in iteration t is equal to:

$$p_{b,t} = \frac{w_{b,t}}{\sum_{b' \in \{behaviors\}} w_{b',t}} \tag{15}$$

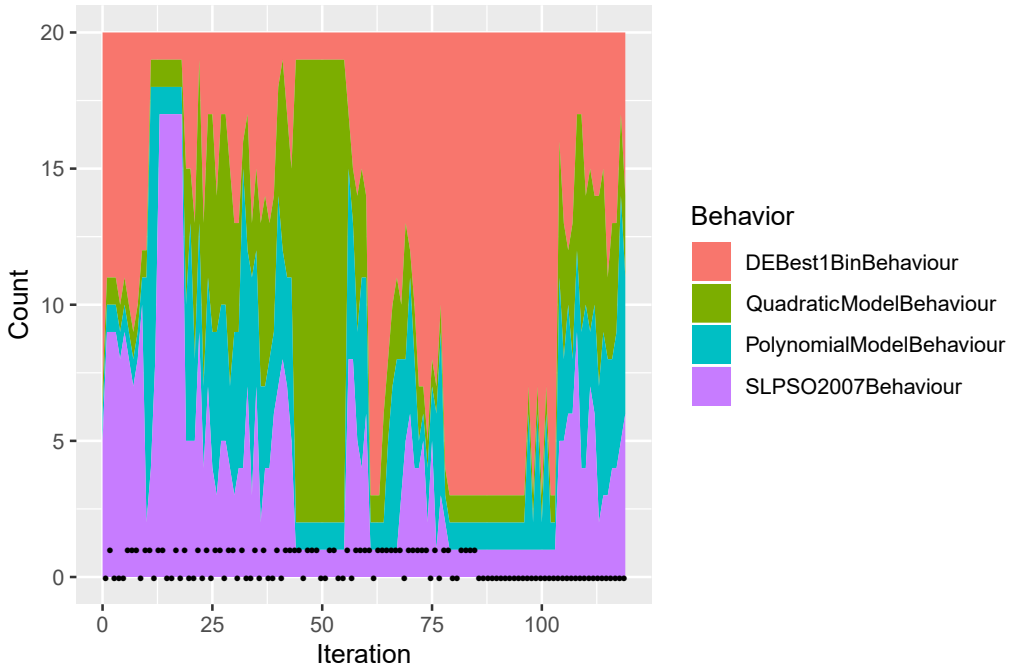
with the following additional rules:

1. Each behavior must be applied to at least one particle in any given iteration
2. If $\sum_{b' \in \{behaviors\}} w_{b',t} = 0$ then $\forall_{b' \in \{behaviors\}} p_{b',t} = \frac{1}{|\{behaviors\}|}$

Note, that this procedure is different from the initial GAPSO adaptation scheme [27], which considered the average improvement of the **local** optimum estimation (i.e. against the former value of $f(particle.x_{best})$ for each particle).

Figure 5 presents the effects of applying adaptation scheme in a single run of the algorithm. In the first few iterations the fractions of using various behaviors oscillate around predefined values set by the user. Afterwards the currently best performing behavior starts to dominate over the others in terms of the frequency of its application. In the final phase, all behaviors

Figure 5 Fractions of using various behaviors during an example algorithm's run (i.e. till a restart). Black dots denote whether in a given iteration a global optimum value was improved



have pairwise equal probabilities of their application due to prolonged lack of improvement. If such a situation lasts for a certain number of iterations, the algorithm is reset (cf. Section 3.3).

Table 2 COCO noiseless functions used in experimental evaluation (all within the range $[-5, 5]^D$ and fully described in [48])

Id	Function	Rotated
$f1$	Sphere	No
$f2$	Elipsoidal	No
$f3$	Rastrigin	No
$f4$	Büche-Rastrigin	No
$f5$	Linear Slope	No
$f6$	Attractive Sector	Yes
$f7$	Step Ellipsoidal	Yes
$f8$	Rosenbrock	No
$f9$	Rosenbrock	Yes
$f10$	Ellipsoidal	Yes
$f11$	Discus	Yes
$f12$	Bent Cigar	Yes
$f13$	Sharp Ridge	Yes
$f14$	Different Powers	Yes
$f15$	Rastrigin	Yes
$f16$	Weierstrass	Yes
$f17$	Schaffers $f7$	Yes
$f18$	Schaffers $f7$ Functions	Yes
$f19$	Composite Griewank-Rosenbrock $f8f2$	Yes
$f20$	Schwefel	Yes
$f21$	Gallagher's Gaussian 101-me Peaks	Yes
$f22$	Gallagher's Gaussian 21-hi Peaks	Yes
$f23$	Katsuura	Yes
$f24$	Lunacek bi-Rastrigin	Yes

4. Experimental evaluation of M-GAPSO

The assessment of M-GAPSO performance was made on 24 continuous functions from the COCO benchmark set, described comprehensively in [49], and 29 functions of CEC-2017 benchmark set [50]. Additionally, possible relations between an optimal algorithm set and the function classes of the COCO test set are discussed in the last subsection.

4.1. Performance analysis of M-GAPSO components on COCO test set

For each function in COCO set the algorithm has been run once for each of 15 functions' instances in each dimension, which is a standard procedure for this benchmark set. Each function instance has different parameters for scaling, transition and rotation (except for the separable functions), hence in this setup the results are aggregated across various instances of the same function instead of multiple runs for a single instance.

The instances differ mainly due to the search space translations, which results in the presence of optima in various locations and diverse objective function values. Among search space translations there are both linear and non-linear ones. Moreover, rotations of selected functions are applied in the process of creating instances. All functions used in the evaluation are presented in Table 2, with an annotation if the function has been subject to rotation transformations or not.

Using this particular benchmark allows for a straightforward comparison with results obtained by various other optimization algorithms, as the benchmark comes with a database of results sent for evaluation in BBOB workshops accompanying GECCO (and occasionally CEC) conference series.

The goal of the experiments was to assess the efficacy of adaptation mechanism implemented within M-GAPSO framework in conjunction with PSO, DE and model-based optimizers hybridization.

M-GAPSO is compared with the PSO implementation from [51] and DE implementation from [5], which were also tested on the COCO, thus offering a fair baseline results. The following four M-GAPSO configurations are tested:

PD - simple PSO-DE hybrid,

PDa - PSO-DE hybrid with adaptation,

PDLP - PSO-DE and model based optimizers hybrid,

PDLPa/PDLa - PSO-DE and model based optimizers hybrid with adaptation[1].

The goal of testing these four configurations is to address the following questions:

[1]for 40 and more dimensions functions, due to long computation time, only PSO-DE and quadratic function model-based optimizer hybrid with adaptation (denoted PDLa) was computed

- How performance of the algorithm is affected by an addition of model based optimization behaviors?
- How does performance of the algorithm change when adaptation is applied?

Table 3 M-GAPSO framework settings with all possible features (**PDLPa**)

Core M-GAPSO parameters	
Population size (COCO set)	$10D$
Population size (CEC-2017 set)	$\max(4D, 100)$
Fitness function evaluations budget	$0.5 * 10^5 D$
Initial PSO behavior weight $w_{PSO,0}$	1000
Initial DE behavior weight $w_{DE,0}$	1000
Initial quadratic model–based behavior weight $w_{LM,0}$	1
Initial polynomial model–based behavior weight $w_{PM,0}$	1
Samples archive index size (COCO set)	20000
Samples archive index size (CEC-2017 set)	5000
Auxiliary techniques parameters	
Adaptation iterations count *history.depth*	10
Stagnation iterations count	20
Population convergence threshold $\Theta_{locations}$	10^{-4}
Values convergence threshold Θ_{values}	10^{-8}
High-quality optimum estimation bounding box relative size	3%
Optimization algorithms parameters	
PSO cognitive factor c_1	1.4
PSO social factor c_2	1.4
PSO velocity inertia factor ω	0.64
DE cross-over probability	0.9
DE mutation scaling factor F	0.0 - 1.4
Samples archive size	$2 * 10^5$
Quadratic model nearest samples count	$5D$
Polynomial model degree	4
Polynomial model nearest samples count	$4D + 1$

Table 3 presents the default values of M-GAPSO parameters for the full features configuration (**PDLPa**). Parameter *history.depth* controls the algorithm adaptation mechanism described in Section 3.6. If no adaptation is

to be performed the *history.depth* is set to 0. If the model based optimizers are not to be used both $w_{LM,0}$ and $w_{PM,0}$ need to be set to 0. The core M-GAPSO parameters and the parameters of auxiliary techniques are meant to be user defined, as their settings shape the hyper-heuristic. Particular methods parameters were set to follow reasonable parameter settings from the literature.

Figs. 6–11 present the average convergence plots of M-GAPSO optimization process on $2D$, $3D$, $5D$, $10D$, $20D$ and $40D$ COCO benchmark functions. Tables 4 and 5 present the number of functions with at least one successful trial within the maximal optimization budget and total ratio of successful trials. A successful trial means obtaining the function value which differs less from the optimal value than the designated target threshold. Thresholds 10^1, 10^{-1}, 10^{-4} and 10^{-8} are analyzed, which is the standard approach within COCO benchmark set noiseless functions testbed.

Table 4 Number of distinct function types obtained by each algorithm for the respective optimization precision target (Δ Value) on 5D and 20D COCO benchmark functions, respectively for $0.5 \times 10^5 D$ evaluations

Dim.	ΔVal.	PSO	DE	PSO+DE	Adapted PSO+DE	PSO+DE +LM+PM	Adapted PSO+DE +LM+PM
5D	10^1	**24**	**24**	**24**	**24**	**24**	**24**
5D	10^{-1}	22	**23**	**23**	**23**	**23**	**23**
5D	10^{-4}	17	**23**	22	21	22	22
5D	10^{-8}	13	**23**	21	20	21	20
20D	10^1	20	19	20	20	**22**	**22**
20D	10^{-1}	9	**14**	12	13	13	**14**
20D	10^{-4}	8	9	12	11	11	**13**
20D	10^{-8}	5	8	10	10	10	**11**

Table 5 Percentage rates of successfully completed runs by each algorithm for the respective optimization precision target (Δ Value) on 5D and 20D COCO benchmark functions, respectively for $0.5 \times 10^5 D$ evaluations

Dim.	ΔVal.	PSO	DE	PSO+DE	Adapted PSO+DE	PSO+DE +LM+PM	Adapted PSO+DE +LM+PM
5D	10^{1}	0.95	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
5D	10^{-1}	0.54	0.91	0.86	0.86	0.90	**0.92**
5D	10^{-4}	0.38	**0.88**	0.79	0.72	0.83	0.84
5D	10^{-8}	0.28	**0.82**	0.73	0.69	0.78	0.77
20D	10^{1}	0.66	0.75	0.82	0.83	0.87	**0.92**
20D	10^{-1}	0.22	0.44	0.47	**0.51**	0.47	0.50
20D	10^{-4}	0.15	0.30	**0.44**	**0.44**	**0.44**	**0.44**
20D	10^{-8}	0.09	0.28	**0.40**	**0.40**	0.39	0.39

Figure 6 Results of various M-GAPSO configurations against baseline PSO [51] and DE [52] on 2D functions from COCO benchmark set.

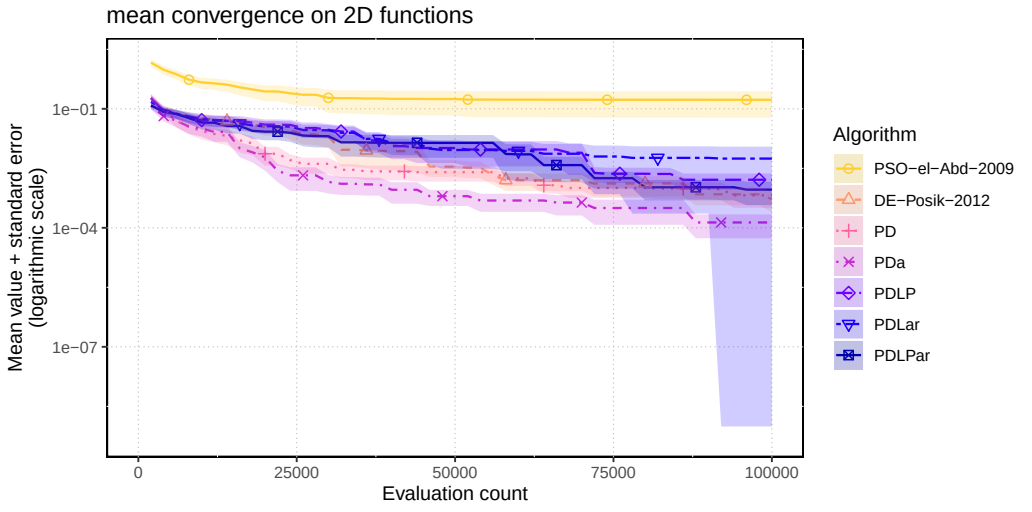


Figure 7 Results of various M-GAPSO configurations against baseline PSO [51] and DE [52] on 3D functions from COCO benchmark set.

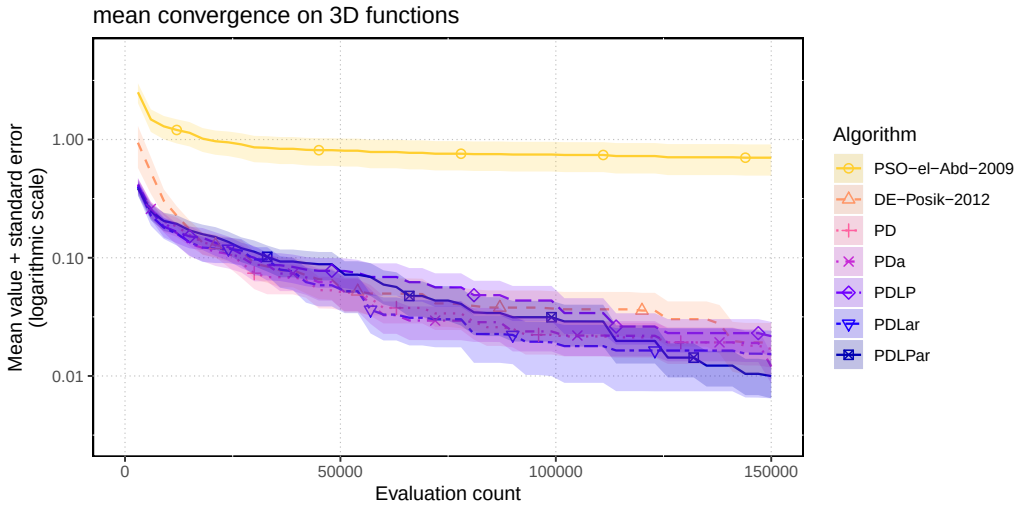


Figure 8 Results of various M-GAPSO configurations against baseline PSO [51] and DE [52] on 5D functions from COCO benchmark set.

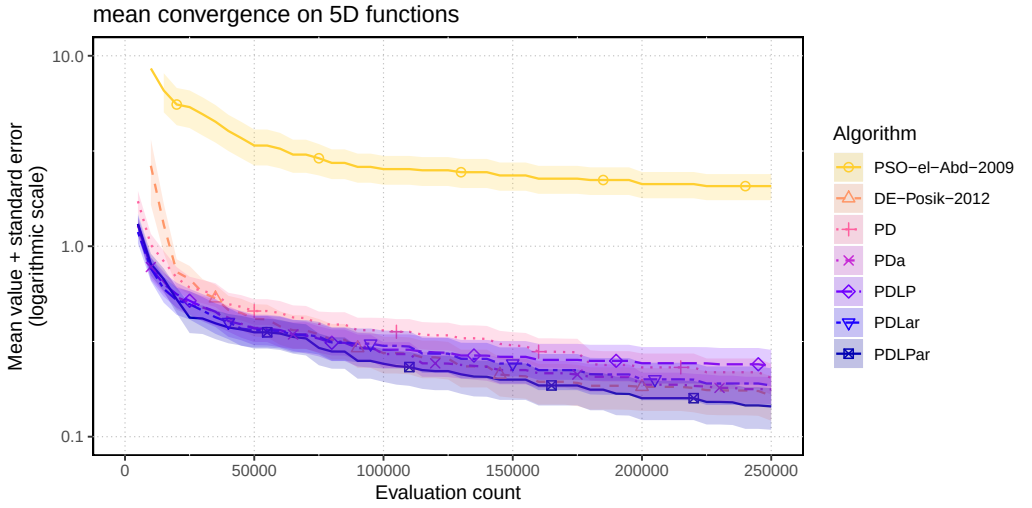


Figure 9 Results of various M-GAPSO configurations against baseline PSO [51] and DE [52] on 10D functions from COCO benchmark set.

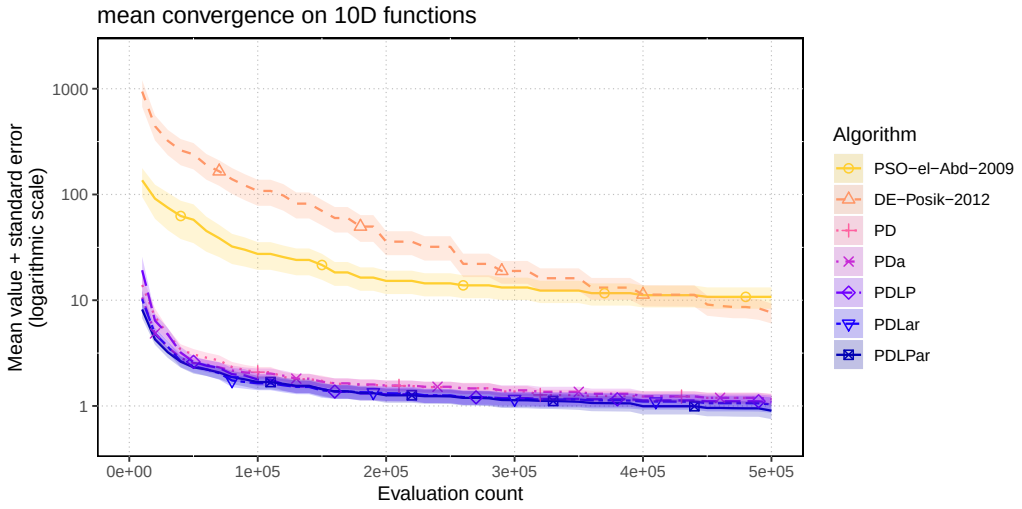


Figure 10 Results of various M-GAPSO configurations against baseline PSO [51] and DE [52] on 20D functions from COCO benchmark set.

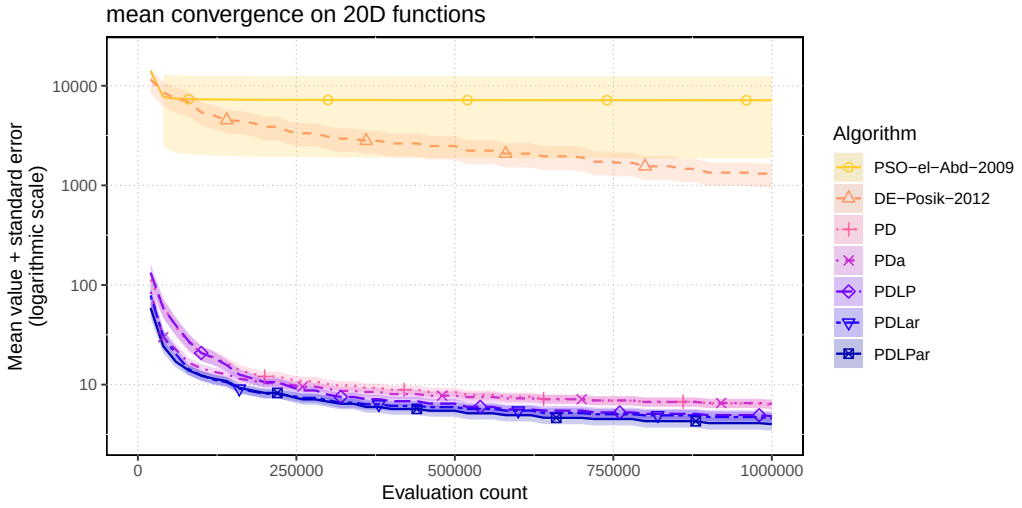
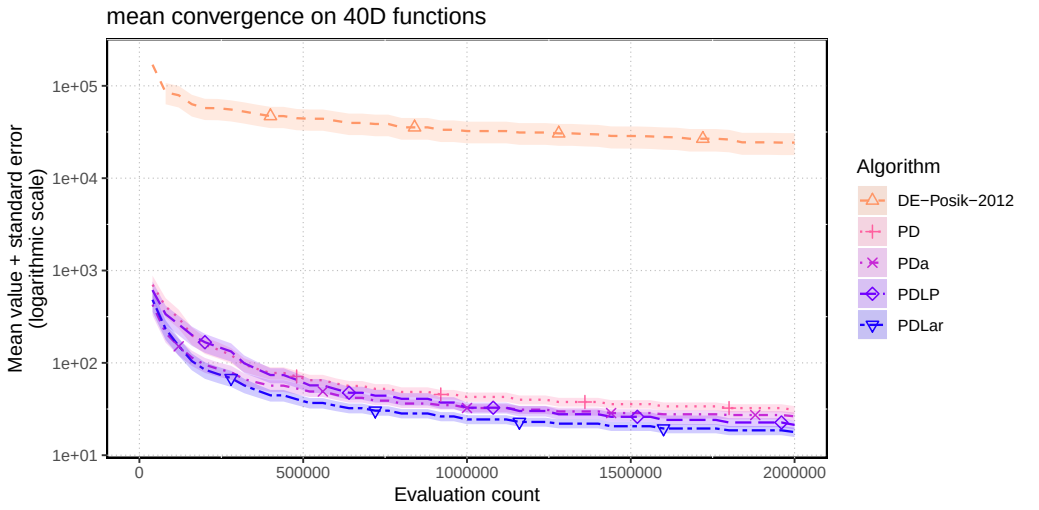


Figure 11 Results of various M-GAPSO configurations against baseline PSO [51] and DE [52] on 40D functions from COCO benchmark set.

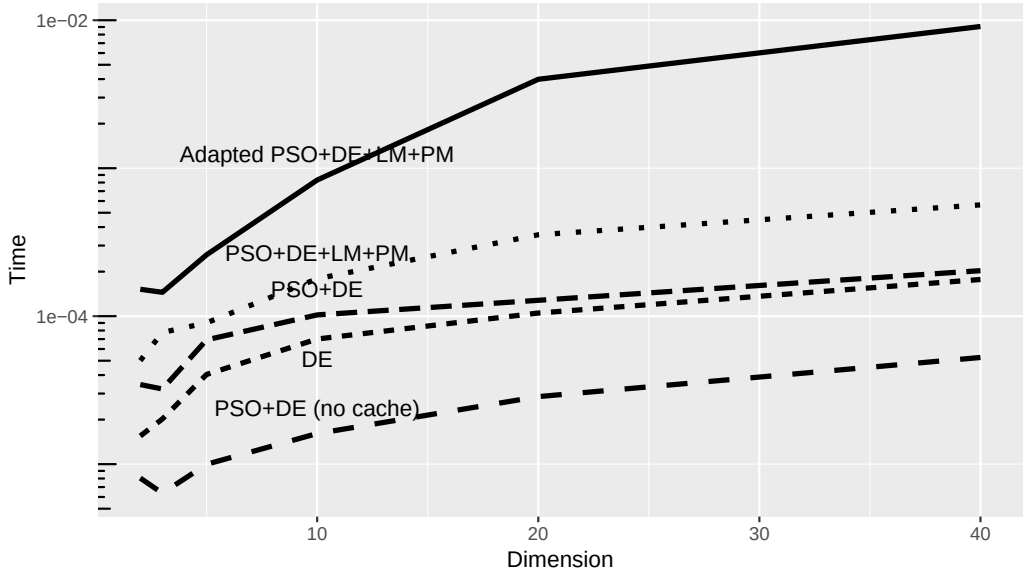


Appendix A presents an analysis of the quality of the best solutions obtained for various function classes. The results of various M-GAPSO configurations are pair-wise compared and tested for statistical significance of improvement.

In order to gain knowledge related to the computational requirements introduced by adding each technique, the running time per fitness function evaluation (RTFF) is presented in Fig. 12. Please note that RTFF reflects a total algorithm computation time divided by the number of fitness function evaluations. Therefore, it takes into account all operations related to adaptation, model building and model optimum selection.

With DE algorithm (with M-GAPSO's function cache enabled) considered to be a baseline method, the following observations can be made: (1) Switching off samples' caching results in around 3 times faster computations. (2) Mixing DE with PSO does not affect the speed significantly, while including model-based behaviors leads to 4 times slower computation than that in the baseline experiment. (3) The most significant disadvantage, in terms of the average computation time normalized by the number of single function evaluations, is caused by simultaneous inclusion of model based approaches and the adaptation mechanism. The reason for that slowdown are the algorithm iterations in which no improvement was observed for a certain amount of time. As none of the four behaviors was preferred, each of them was

Figure 12 The average time necessary to select and evaluate a single point within the search space for various M-GAPSO configurations with respect to function dimension on COCO benchmark set



selected with probability 0.25. Therefore, around a half of the function evaluations was a result of model based approaches (and these methods, because of samples retrieval and model fitting, are inherently slower than sampling based PSO and DE behaviors).

4.2. Performance analysis of M-GAPSO components on the CEC-2017 test set

In order to validate the results obtained on the COCO test set, the M-GAPSO has been additionally run on CEC-2017 test set, described in detail in the technical report [50]. CEC-2017 consists of 29 bound-constraint functions belonging to one of the four categories: unimodal functions (F_1), simple multimodal functions (F_3 - F_9), hybrid functions (F_{10} - F_{19}), and composition functions (F_{20} - F_{30}). Search space is bounded to $[-100, 100]^D$. Explicit definitions of the functions are included in the technical report. Each function comes in four variants: 10, 30, 50, and 100 dimensional. Evaluation of the algorithm for a specific function and dimension assumes 51 independent runs, for which the final function value is stored.

CEC-2017 assumes $10^4 \cdot D$ optimization budget, where D is function dimensionality. Default evaluation procedure, derived from the technical report [50], was applied for plain **PSO**, **DE** and M-GAPSO in **PDa** and **PDLa** configurations. The final measure $Score$, with the maximal value of 100, is a sum of $Score1$ and $Score2$. $Score1$ is a transformation of the SE value. SE represents the weighted sum of mean final function values obtained in all problems (for each function and each dimension an average of 51 independent runs is taken). $Score2$ is a transformation of the SR value. SR is calculated as a weighted ranking measure (among other algorithms) calculated for each problem. The ranking measure is computed using an average value of 51 independent runs.

Table 6 presents the results of **PDa** and **PDLa** variants of M-GAPSO compared with pure **DE** and **PSO**. The advantage of the hybridization over standalone **DE** and **PSO** algorithms is clearly visible. **PDa** achieved a final score of 97.67, **PDLa** 95.50, while **DE** and **PSO** reached 82.92 and 67.68, respectively. The difference between **PDa** and **PDLa** is marginal. **PDa** is slightly better in the case of ranking measure, while **PDLa** obtained lower mean error values.

Table 6 Scores achieved by M-GAPSO (PDa and PDLa variants) in relation to baseline DE and PSO algorithms on the CEC-2017 benchmark set.

Score	**DE**	**PSO**	**PDa**	**PDLa**
SE	19.25	24.33	16.52	15.75
SR	73.35	87.25	61.65	67.75
Score 1	40.90	32.35	47.67	50.00
Score 2	42.02	35.33	50.00	45.50
Score	**82.92**	**67.68**	**97.67**	**95.50**

A complexity of the algorithms was computed according to the CEC-2017 technical report [50]. T_0 is the time of a test program run. T_1 is the time of pure $2 \cdot 10^5$ evaluations of F_{18} function. T_2 is the average running time of the optimization algorithm for F_{18} function using $2 \cdot 10^5$ evaluation budget. $(T_2 - T_1)/T_0$ is the final complexity measure. The results are presented in Table 7.

In addition, **PDa** and **PDLa** variants of M-GAPSO were compared with recent **RFO** [15] and **DOBL** [9] algorithms which reported results on 30D and 50D for the CEC-2017 test set.

Table 7 Complexity of M-GAPSO calculated according CEC-2017 (i.e. for one benchmark representative). T_0 is the time of a test program run. T_1 - time of pure $2 \cdot 10^5$ evaluations of F_{18} function. T_2 - the average running time of the algorithm for F_{18} with $2 \cdot 10^5$ evaluation budget. $(T_2 - T_1)/T_0$ is the final complexity.

D	*Algorithm*	$T_0[s]$	$T_1[s]$	$T_2[s]$	$(T_2 - T_1)/T_0$
10	PSO	0.00285	0.21	1.27	375
	DE			1.15	330
	PDa			1.39	415
	PDLa			8.48	2903
30	PSO	0.00285	0.49	1.86	482
	DE			1.58	385
	PDa			1.71	458
	PDLa			9.35	3113
50	PSO	0.00285	0.90	4.63	1310
	DE			2.82	991
	PDa			3.81	1020
	PDLa			17.12	5692
100	PSO	0.00285	2.84	15.62	4486
	DE			13.39	3702
	PDa			12.95	3548
	PDLa			51.45	17061

DOBL results were taken directly from [9], while **RFO** results were computed following the implementation description from [15] with a slight correction that improved the efficacy of the RFP[2].

Tables 8 and 9 present the mean and the standard deviation values of final function values from 51 repetitions of each problem for $10^4 \cdot D$ optimiation budget. In addition, the t-test was applied for each problem to check statistically significant differences between the results. The results of M-GAPSO

[2]A description of RFO presented in the original paper [9] seems to have a typo in Eq (9). This equation describes a reproduction of a new individual based on alpha couple (two best individuals from a population). Changing the original formulation to the ordinary interpolation significantly improved the results of this algorithm.

which are significantly better (p-*value*=0.05) than both **RFO** and **DOBL** are marked with $\star$. Likewise, **RFO** and **DOBL** results are marked if they are significantly better than the remaining results.

Table 8 Results achieved by M-GAPSO (PDa and PDLa variant) in relation to RFO and DOBL for $30D$ problems using CEC-2017 benchmark. Best mean obtained for each function is bolded. Results of M-GAPSO significantly better (t-test) than both RFO and DOBL are marked with $\star$.

F.	PDa		PDLa		RFO		DOBL	
	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
1	**0.0e+00**	0.0e+00	**0.0e+00**	0.0e+00	4.6e+03	5.2e+03	**0.0e+00**	0.0e+00
2	**0.0e+00**$\star$	0.0e+00	**0.0e+00**$\star$	0.0e+00	9.1e+02	5.9e+02	1.6e+03	2.3e+03
3	**2.9e+00**$\star$	8.2e+00	1.3e+01	2.4e+01	1.0e+02	2.3e+01	1.6e+01	2.8e+01
4	4.4e+01	6.9e+00	5.8e+01	1.1e+01	2.0e+02	5.2e+01	**2.8e+01**$\star$	2.5e+01
5	9.2e+00	2.8e+00	5.3e+00	6.4e+00	5.4e+01	7.8e+00	**2.1e-04**$\star$	6.2e-05
6	3.9e+01$\star$	4.0e+00	**3.9e+01**$\star$	2.7e+00	6.0e+02	2.2e+02	1.2e+02	4.7e+01
7	4.1e+01	8.4e+00	5.5e+01	1.2e+01	1.8e+02	5.0e+01	**3.2e+01**$\star$	3.1e+01
8	6.0e+02	3.4e+02	8.0e+02	3.9e+02	4.4e+03	1.3e+03	**0.0e+00**$\star$	0.0e+00
9	3.4e+03$\star$	7.0e+02	**3.2e+03**$\star$	6.2e+02	4.8e+03	1.0e+03	6.4e+03	8.6e+02
10	6.6e+01	1.9e+01	6.9e+01	2.3e+01	2.3e+02	5.9e+01	**1.3e+01**$\star$	1.3e+01
11	5.2e+03$\star$	2.9e+03	**4.7e+03**$\star$	5.0e+03	1.5e+07	1.4e+07	3.3e+04	6.8e+04
12	8.5e+02	7.7e+02	2.1e+02	1.4e+02	1.2e+05	8.7e+04	**8.6e+01**$\star$	3.0e+01
13	1.0e+02	3.5e+01	9.6e+01	2.3e+01	2.4e+04	1.5e+05	**1.8e+01**$\star$	1.3e+01
14	1.9e+02	8.9e+01	1.1e+02	6.6e+01	5.8e+04	3.9e+04	**7.8e+00**$\star$	4.1e+00
15	1.2e+03	2.8e+02	1.1e+03	3.7e+02	1.6e+03	3.5e+02	**6.9e+02**$\star$	3.3e+02
16	3.4e+02	1.8e+02	3.4e+02	1.3e+02	7.9e+02	2.4e+02	**8.4e+01**$\star$	3.9e+01
17	**1.6e+02**$\star$	1.3e+02	2.3e+02$\star$	2.4e+02	1.8e+05	1.6e+05	9.2e+03	8.8e+03
18	1.0e+02	5.1e+01	5.4e+01	1.9e+01	7.5e+05	5.8e+05	**1.1e+01**$\star$	6.4e+00
19	4.1e+02	1.9e+02	3.3e+02	1.3e+02	8.0e+02	2.0e+02	**3.6e+01**$\star$	2.9e+01
20	2.8e+02	1.4e+01	2.8e+02	1.5e+01	3.9e+02	4.1e+01	**2.3e+02**$\star$	2.1e+01
21	6.8e+02	1.4e+03	3.3e+02	9.4e+02	3.7e+03	2.5e+03	**1.0e+02**$\star$	6.1e-12
22	4.9e+02	3.7e+01	5.1e+02	4.2e+01	5.7e+02	6.8e+01	**3.6e+02**$\star$	7.7e+00
23	4.9e+02	1.4e+01	5.4e+02	3.5e+01	5.9e+02	4.4e+01	**4.3e+02**$\star$	9.2e+00
24	3.9e+02	1.8e+00	3.9e+02	1.8e+00	4.2e+02	2.5e+01	**3.8e+02**	3.1e+00
25	2.4e+03	5.0e+02	1.3e+03	1.1e+03	3.9e+03	1.1e+03	**4.8e+02**$\star$	3.2e+02
26	5.7e+02	3.0e+01	5.9e+02	2.8e+01	6.0e+02	4.8e+01	**5.4e+02**$\star$	2.0e+01
27	**3.1e+02**$\star$	3.4e+01	3.1e+02$\star$	2.5e+01	4.6e+02	3.4e+01	3.3e+02	4.9e+01
28	1.0e+03	1.9e+02	1.1e+03	2.2e+02	1.7e+03	6.3e+02	**5.8e+02**$\star$	7.8e+01
29	**2.6e+03**	5.1e+02	2.7e+03	5.1e+02	2.4e+06	1.7e+06	2.6e+03	3.4e+02
	6/7$\star$		**5**/6$\star$		**0**/0$\star$		**21**/19$\star$	

Table 9 Results achieved by M-GAPSO (PDa and PDLa variant) in relation to RFO and DOBL for $50D$ problems using CEC-2017 benchmark. Best mean obtained for each function is bolded. Results of M-GAPSO significantly better (t-test) than both RFO and DOBL are marked with $\star$.

F.	PDa		PDLa		RFO		DOBL	
	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
1	**0.0e+00**$\star$	0.0e+00	1.4e-09$\star$	6.0e-09	2.5e+06	2.1e+06	9.0e+02	1.2e+03
2	1.9e-02$\star$	6.8e-02	**0.0e+00**$\star$	0.0e+00	1.9e+04	5.8e+03	4.5e+04	9.5e+03
3	**1.6e+01**$\star$	2.6e+01	1.8e+01$\star$	3.1e+01	2.4e+02	4.4e+01	1.0e+02	5.2e+01
4	8.3e+01	9.9e+00	1.4e+02	1.7e+01	3.8e+02	6.3e+01	**5.1e+01**$\star$	4.2e+01
5	1.3e+01	3.2e+00	1.5e+01	1.2e+01	6.2e+01	7.2e+00	**4.1e-02**$\star$	1.1e-02
6	7.3e+01$\star$	9.3e+00	**7.2e+01**$\star$	5.8e+00	1.4e+03	4.9e+02	2.0e+02	7.2e+01
7	8.0e+01	1.2e+01	1.4e+02	1.9e+01	3.9e+02	5.1e+01	**5.6e+01**$\star$	3.8e+01
8	2.9e+03	1.1e+03	3.9e+03	1.2e+03	1.3e+04	2.8e+03	**8.2e-01**$\star$	5.8e+00
9	6.2e+03$\star$	9.9e+02	**6.1e+03**$\star$	8.8e+02	8.6e+03	1.2e+03	1.2e+04	6.2e+02
10	1.2e+02	2.5e+01	1.5e+02	4.5e+01	4.4e+02	9.3e+01	**5.5e+01**$\star$	1.8e+01
11	1.5e+04$\star$	9.2e+03	**7.1e+03**$\star$	5.1e+03	1.1e+08	8.3e+07	7.1e+05	4.0e+05
12	1.9e+03	8.1e+02	**6.3e+02**$\star$	3.1e+02	1.6e+05	1.8e+05	8.3e+02	8.9e+02
13	1.6e+02	4.2e+01	2.1e+02	3.7e+01	7.4e+04	4.9e+04	**5.0e+01**$\star$	1.2e+01
14	3.7e+02	1.4e+02	2.6e+02	1.1e+02	5.9e+04	3.9e+04	**8.6e+01**$\star$	4.2e+01
15	2.0e+03	4.7e+02	2.0e+03	4.4e+02	2.8e+03	8.3e+02	**5.0e+02**$\star$	3.9e+02
16	1.4e+03	3.5e+02	1.5e+03	4.0e+02	2.3e+03	8.0e+02	**1.1e+03**$\star$	2.9e+02
17	1.3e+03$\star$	1.2e+03	**6.5e+02**$\star$	5.9e+02	7.3e+05	5.0e+05	1.6e+05	2.0e+05
18	1.2e+02$\star$	3.7e+01	**9.3e+01**$\star$	2.4e+01	2.2e+06	1.6e+06	5.7e+02	1.3e+03
19	9.2e+02	3.0e+02	9.0e+02	2.3e+02	1.6e+03	3.3e+02	**8.4e+02**$\star$	3.2e+02
20	3.8e+02	2.1e+01	3.8e+02	2.4e+01	6.2e+02	9.6e+01	**2.5e+02**$\star$	2.4e+01
21	6.5e+03	1.8e+03	6.8e+03	1.3e+03	9.1e+03	1.2e+03	**1.2e+03**$\star$	3.2e+03
22	7.4e+02	6.6e+01	8.0e+02	9.7e+01	9.3e+02	1.5e+02	**4.5e+02**$\star$	1.4e+01
23	6.7e+02	2.2e+01	7.3e+02	7.4e+01	8.8e+02	9.0e+01	**5.3e+02**$\star$	1.2e+01
24	**4.9e+02**$\star$	2.9e+01	5.0e+02$\star$	3.3e+01	6.2e+02	4.3e+01	5.1e+02	4.1e+01
25	3.8e+03	4.8e+02	2.5e+03	2.0e+03	6.7e+03	1.5e+03	**1.7e+03**$\star$	1.5e+02
26	1.1e+03	1.5e+02	1.2e+03	2.1e+02	1.1e+03	1.5e+02	**5.3e+02**$\star$	1.7e+01
27	**4.8e+02**$\star$	2.1e+01	4.8e+02$\star$	2.7e+01	1.0e+03	1.1e+03	4.9e+02	2.1e+01
28	1.7e+03	4.1e+02	2.0e+03	4.3e+02	4.1e+03	2.3e+03	**5.3e+02**$\star$	2.8e+02
29	6.4e+05$\star$	6.2e+04	**6.2e+05**$\star$	4.8e+04	6.7e+07	1.6e+07	2.3e+06	4.6e+05
	4/11$\star$		**8**/12$\star$		**0**/0$\star$		**17**/17$\star$	

For 30 dimensional functions **PDa** obtained the lowest mean for 6 functions, while **PDLa** for 5 functions. At the same time, **PDa** and **PDLa** were significantly better than **RFO** and **DOBL** for 7 and 6 functions, respectively. The best overall algorithm was **DOBL** which achieved the lowest mean in 21 cases and statistically significant advantage, compared to all other algorithms, in 19 cases. **RFO** never obtained the best result but was comparative

to **DOBL** and M-GAPSO for a few functions.

Generally, the results for 50 dimensional functions are similar to those for 30 dimensional. **DOBL** is still the best algorithm in the comparison. It reached the best mean for 17 functions and was significantly better than the remaining metaheuristics in each of these 17 cases. Both M-GAPSO variants improved their results over the 30 dimensional problems. **PDa** was significantly better than both **DBOL** and **RFO** in 11 cases, while **PDLa** in 12 cases. The results of **RFO**, when comparing to both M-GAPSO configurations and **DBOL**, remained at the same level as in the case of 30 dimensions.

4.3. Analysis of the adaptation module results

This section discusses the ratio of behaviors applied while running M-GAPSO in PDLPa configuration on the COCO test set. This ratio reflects the impact of a given behavior on the results during a given algorithm's run. These ratios are aggregated into five function classes and presented in Table 10. Please note that $f1$ and $f5$ are excluded from this analysis since they were solved within the first iteration and the data gathered by the adaptation mechanism was insufficient.

Please refer to Table 2 for the list of functions within each of the five classes (separable, low conditioned, high conditioned, multimodal, and multimodal without global structure).

The following observations can be made based on the collected data:

1. Overall, in average, DE behavior is superior within all function classes.
2. For functions with non-separable variables ($f10$ - $f14$) all other behaviors are almost useless.
3. For all other function classes PSO is utilized at the level similar to DE.
4. Model-based optimizers are particularly useful for separable functions, because in order to keep the models small they do not include any interactions between variables.
5. For the other two classes of multimodal functions ($f15$ - $f24$) model-based optimizers are useful in precise locating the local optima when their estimated location is known [29].

5. Conclusions

The work presents a successful attempt at designing a hyper-heuristic framework able to accommodate various optimization algorithms. The frame-

Table 10 Average behavior application ratio within M-GAPSO on the COCO benchmark set

Functions	Function class	PSO	DE	QM	PM
$f2$ - $f4$	separable	0.14	0.38	0.26	0.23
$f6$ - $f9$	low-condition	0.15	0.64	0.10	0.10
$f10$ - $f14$	high-condition	0.08	0.80	0.06	0.05
$f15$ - $f19$	multimodal with global structure	0.17	0.50	0.20	0.14
$f20$ - $f24$	multimodal without global structure	0.19	0.44	0.20	0.16

work is designed in an extendable way, thanks to the abstract approach to algorithm programming interface. Additionally, the knowledge required by the adaptation module is limited to the information about the performance of the algorithm only.

Achieved results support the claim that hybridizing PSO and DE could improve the performance (especially for functions of at least $10D$) in comparison with applying each of these methods alone. Adding model-based optimizer visibly improves PSO-DE hybrid in higher dimensions, while for $5D$ (and less), simple DE still comes better off in some cases. Good scaling properties are confirmed by experiments on the CEC-2017 test set, with PDLa configuration achieving the results comparable to DOBL algorithm for $50D$ functions. On COCO benchmark set, adding adaptation mechanism improves the results in either case (i.e. PDa configuration is more effective than PD, and PDLPa is superior to PDLP). However, it should be noted that while PSO-DE hybridization and adaptation does not significantly impact computational load, the same cannot be said about the algorithms incorporating the model-based optimizer.

In the case of CEC-2017 benchmarking, the outcomes are similar to those for COCO. The concept of hybridization proved to be beneficial also for this benchmark. Both variants of M-GAPSO (PDa and PDLa) outperformed standalone DE and PSO algorithms. The final results of PDa and PDLa are similar, although PDa is slightly better in the ranking measure, while PDLa is better regarding the averaged error.

Significant part of the improvement comes from changing the underlying philosophy of the method. In the initial version of GAPSO [27] reset and adaptation mechanisms focused on preventing the potential swarm collapse (during the algorithm's run), which was motivated by a theoretical per-

spective of global optimization algorithms. Within this view, the algorithm should be constructed in a way which ensures that the limit of probability $P(x^* \in X_n)$ of including a global optimum x^* in the set of tested solutions X_n approaches 1, as the number of algorithm's iterations n approaches infinity [17].

M-GAPSO approaches this differently, with individual particles focusing only on the task at hand (relatively quick convergence to high-quality local optimum), and the burden of exploration lies on higher level mechanisms: the convergence detector ($RestartManager$) and re-initialization module ($SearchSpaceManager$).

The authors' future research will be concentrated on utilization of the knowledge of multiple local optima estimations. As could have been observed in Fig. 3 positions of local optima gathered during the optimization process may form predictable structures which the authors plan to exploit in their future research.

M-GAPSO should greatly benefit from a more structured approach to selecting search space boundaries in subsequent algorithm runs. Therefore, the authors believe that **multiple and relatively short algorithm runs, combined with predictive setup of the algorithm search space boundaries are the key factors in further improvement of M-GAPSO results.**

Acknowledgments

The authors would like to thank Mikołaj Małkiński, Piotr Podbielski and Łukasz Lepak, students of the Faculty of Mathematics and Information Science Faculty of Warsaw University of Technology, for their help in porting CEC-2017 benchmark functions as a Java library using JNI technology.

References

[1] Kenneth Alan De Jong. Analysis of the behavior of a class of genetic adaptive systems. Phd thesis, University of Michigan, 1975.

[2] John Henry Holland. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT Press, 1992.

[3] Rainer Storn and Kenneth Price. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. Journal of Global Optimization, 11(4):341–359, 1997.

[4] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). Evolutionary Computation, 11(1):1–18, mar 2003.

[5] Petr Poaík and Václav Klema. JADE, an adaptive differential evolution algorithm, benchmarked on the BBOB noiseless testbed. In Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion - GECCO Companion '12, page 197, New York, New York, USA, 2012. ACM Press.

[6] Ilya Loshchilov, Marc Schoenauer, and Michele Sèbag. BI-population CMA-ES Algorithms with Surrogate Models and Line Searches. In GECCO '13 Companion Proceedings of the 15th annual conference companion on Genetic and evolutionary computation, pages 1177–1184, 2013.

[7] Janez Brest, Mirjam Sepesy Maucec, and Borko Boskovic. iL-SHADE: Improved L-SHADE algorithm for single objective real-parameter optimization. In 2016 IEEE Congress on Evolutionary Computation (CEC), pages 1188–1195. IEEE, jul 2016.

[8] Takahiro Yamaguchi and Youhei Akimoto. Benchmarking the novel CMA-ES restart strategy using the search history on the BBOB noiseless testbed. In GECCO '17 Proceedings of the Genetic and Evolutionary Computation Conference Companion, pages 1780–1787, 2017.

[9] Jiahang Li, Yuelin Gao, Kaiguang Wang, and Ying Sun. A dual opposition-based learning for differential evolution with protective mechanism for engineering optimization problems. Applied Soft Computing, 113:107942, 2021.

[10] Vladimir Stanovov, Shakhnaz Akhmedova, and Eugene Semenkin. NL-SHADE-RSP Algorithm with Adaptive Archive and Selective Pressure for CEC 2021 Numerical Optimization. In 2021 IEEE Congress on Evolutionary Computation (CEC), pages 809–816, 2021.

[11] Nikolaus Hansen. A global surrogate assisted cma-es. In Proceedings of the genetic and evolutionary computation conference, pages 664–672, 2019.

[12] Nikolaus Hansen. Benchmarking a bi-population cma-es on the bbob-2009 function testbed. In Proceedings of the 11th annual conference companion on genetic and evolutionary computation conference: late breaking papers, pages 2389–2396, 2009.

[13] Mateusz Zaborski and Jacek Mańdziuk. Improving LSHADE by means of a pre-screening mechanism. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '22, pages 884–892. Association for Computing Machinery, 2022.

[14] Mateusz Zaborski and Jacek Mańdziuk. LQ-R-SHADE: R-SHADE with quadratic surrogate model. In Proceedings of the 21st International Conference on Artificial Intelligence and Soft Computing (ICAISC'22), 2022.

[15] Dawid Połap and Marcin Woźniak. Red fox optimization algorithm. Expert Systems with Applications, 166:114107, 2021.

[16] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. IEEE transactions on evolutionary computation, 1(1):67–82, 1997.

[17] A. E. Eiben, E. H. L. Aarts, and K. M. Van Hee. Global convergence of genetic algorithms: A markov chain analysis. In Hans-Paul Schwefel and Reinhard Männer, editors, Parallel Problem Solving from Nature, pages 3–12, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[18] R. Poli. Mean and Variance of the Sampling Distribution of Particle Swarm Optimizers During Stagnation. IEEE Transactions on Evolutionary Computation, 13(4):712–721, aug 2009.

[19] Frans Van Den Bergh and Andries Petrus Engelbrecht. A convergence proof for the particle swarm optimiser. Fundamenta Informaticae, 105(4):341–374, 2010.

[20] Peter Cowling, Graham Kendall, and Eric Soubeiga. A Hyperheuristic Approach to Scheduling a Sales Summit. In Practice and Theory of Automated Timetabling III. PATAT 2000. Lecture Notes in Computer Science, pages 176–190. Springer, Berlin, Heidelberg, 2001.

[21] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In Handbook of Metaheuristics, pages 457–474. Kluwer Academic Publishers, Boston, 2003.

[22] José Carlos Villela Tinoco and Carlos A. Coello Coello. hypDE: A Hyper-Heuristic Based on Differential Evolution for Solving Constrained Optimization Problems. In EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II pp, pages 267–282. Springer-Verlag, 2013.

[23] Jacomine Grobler, Andries P Engelbrecht, Graham Kendall, and V S S Yadavalli. Heuristic space diversity control for improved meta-hyper-heuristic performance. INFORMATION SCIENCES, 300:49–62, 2015.

[24] R. Damaševičius and M. Woźniak. State Flipping Based Hyper-Heuristic for Hybridization of Nature Inspired Algorithms. In ICAISC 2017: Artificial Intelligence and Soft Computing, pages 337–346, 2017.

[25] Fabio Caraffini, Ferrante Neri, and Michael Epitropakis. HyperSPAM : A study on hyper-heuristic coordination strategies in the continuous domain. Information Sciences, 477:186–202, 2019.

[26] Michał Okulewicz. Finding an Optimal Team. In Position Papers of the 2016 Federated Conference on Computer Science and Information Systems, pages 205–210. Polish Information Processing Society, oct 2016.

[27] Mateusz Uliński, Adam Żychowski, Michał Okulewicz, Mateusz Zaborski, and Hubert Kordulewski. Generalized Self-adapting Particle Swarm Optimization Algorithm. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 3242, pages 29–40. Springer, Cham, 2018.

[28] Mateusz Zaborski, Michał Okulewicz, and Jacek Mańdziuk. Generalized Self-Adapting Particle Swarm Optimization algorithm with model-based optimization enhancements. In Proceedings of 2nd PPRAI Conference, pages 380–383, 2019.

[29] Mateusz Zaborski, Michał Okulewicz, and Jacek Mańdziuk. Analysis of statistical model-based optimization enhancements in generalized self-adapting particle swarm optimization framework. Foundations of Computing and Decision Sciences, 45, 2020.

[30] Michał Okulewicz, Mateusz Zaborski, and Jacek Mańdziuk. Generalized Self-Adapting Particle Swarm Optimization algorithm with archive of samples. https://arxiv.org/abs/2002.12485, 2020. [arXiv preprint].

[31] Nikolaus Hansen, Dimo Brockhoff, Olaf Mersmann, Tea Tusar, Dejan Tusar, Ouassim Ait ElHara, Phillipe R Sampaio, Asma Atamna, Konstantinos Varelas, Umut Batu, Duc Manh Nguyen, Filip Matzner, and Anne Auger. COmparing Continuous Optimizers: numbbo/COCO on Github, 2019.

[32] James Kennedy and Russell C. Eberhart. Particle Swarm Optimization. Proceedings of IEEE International Conference on Neural Networks. IV, pages 1942–1948, 1995.

[33] Maurice Clerc. Standard particle swarm optimisation, 2012.

[34] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In Proceedings of the 1990 ACM SIGMOD international conference on Management of data, pages 322–331, 1990.

[35] Éric D Taillard, Luca M Gambardella, Michel Gendreau, and Jean-Yves Potvin. Adaptive memory programming: A unified view of metaheuristics. European Journal of Operational Research, 135(1):1–16, nov 2001.

[36] Fred Glover. Tabu search and adaptive memory programming—advances, applications and challenges. In Interfaces in computer science and operations research, pages 1–75. Springer, 1997.

[37] Hui Liu, Zixing Cai, and Yong Wang. Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Applied Soft Computing, 10(2):629–640, mar 2010.

[38] Xiaobing Yu, Jie Cao, Haiyan Shan, Li Zhu, and Jun Guo. An adaptive hybrid algorithm based on particle swarm optimization and differential evolution for global optimization. The Scientific World Journal, 2014:215472, feb 2014.

[39] Riccardo Poli, William B Langdon, and Owen Holland. Extending Particle Swarm Optimisation via Genetic Programming. In European Conference on Genetic Programming, pages 291–300. Springer, 2005.

[40] Péricles Barbosa Miranda and Ricardo Bastos Prudêncio. GEFPSO: A framework for PSO optimization based on Grammatical Evolution. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, pages 1087–1094, 2015.

[41] Henry Zapata, Niriaska Perozo, Wilfredo Angulo, and Joyne Contreras. A Hybrid Swarm Algorithm for Collective Construction of 3D Structures. Int. J. Artif. Intell., 18(1):1–18, 2020.

[42] CL Camacho Villalón, Marco Dorigo, and Thomas Stützle. PSO-X: A Component-Based Framework for the Automatic Design of Particle Swarm Optimization Algorithms. IEEE Transactions on Evolutionary Computation, 2021.

[43] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives, 3:43–58, 2016.

[44] Zhi-Hui Zhan, Jun Zhang, Yun Li, and Henry Shu-Hung Chung. Adaptive particle swarm optimization. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society, 39(6):1362–1381, dec 2009.

[45] Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. In 2013 IEEE congress on evolutionary computation, pages 71–78. IEEE, 2013.

[46] Peter S Bullen. Handbook of means and their inequalities, volume 560. Springer Science & Business Media, 2013.

[47] Mudita Sharma, Manuel López-Ibánez, and Dimitar Kazakov. Performance Assessment of Recursive Probability Matching for Adaptive Operator Selection in Differential Evolution. In International Conference on Parallel Problem Solving from Nature, pages 321–333. Springer, 2018.

[48] Ouassim Elhara, Konstantinos Varelas, Duc Nguyen, Tea Tusar, Dimo Brockhoff, Nikolaus Hansen, and Anne Auger. Coco: the large scale black-box optimization benchmarking (bbob-largescale) test suite. arXiv preprint arXiv:1903.06396, 2019.

[49] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. Optimization Methods and Software, 36(1):114–144, 2021.

[50] N.H. Awad, M.Z. Ali, P.N. Suganthan, Liang J.J., and Qu B.Y. Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session Competition on Constrained Real-Parameter Optimization, 2017. `https://github.com/P-N-Suganthan/CEC2017-BoundContrained`.

[51] Mohammed El-Abd and Mohamed S Kamel. Black-box optimization benchmarking for noiseless function testbed using particle swarm optimization. In Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, pages 2269–2274, 2009.

[52] Petr Pošík and Václav Klemš. Benchmarking the differential evolution with adaptive encoding on noiseless functions. In Proceedings of the 14th annual conference companion on Genetic and evolutionary computation, pages 189–196, 2012.

Appendix A. Detailed comparison of M-GAPSO configurations and baseline DE results

The appendix analyses statistical significance of the results. Tables A.11 - A.15 present mean values and standard errors for each function calculated for the best function values found. Additionally, in each table the results of Wilcoxon Signed Rank Test between pairs of experiments are provided.

Tables A.16 - A.19 present a ranking of configurations. The lower the value the higher the number of instances for which a given approach found better solutions.

Table A.11 Comparison of mean function values and standard errors for final solutions of DE-Posik-2012 and PD with results of Wilcoxon Signed Rank Test on COCO benchmark set. "+" denotes significantly better result of PD, "=" denotes no significant difference, and "-" denotes significantly better result of DE-Posik-2012

	D5 DE	D5 PD	D5	D20 DE	D20 PD	D20
f1	0±0	0±0	+	0±0	0±0	+
f2	0±0	0±0	+	0±0	0±0	+
f3	0±0	0.13±0.35	=	0.46±0.64	26.4±4.6	-
f4	0±0	0.53±0.51	=	3.52±1.12	37.28±7.1	-
f5	0±0	0±0	-	0±0	0±0	-
f6	0±0	0±0	+	0±0	0±0	+
f7	0±0	0±0	=	0.68±0.83	3.15±1.23	-
f8	0±0	0±0	+	0.27±1.03	0±0	+
f9	0±0	0±0	+	11.87±0.78	0±0	+
f10	0±0	0±0	+	31214.65±12132.71	3.64±3.37	+
f11	0±0	0±0	+	16.4±6.52	0±0	+
f12	0±0	0±0	+	2.11±2.89	0±0	+
f13	0±0	0±0	+	0.98±2.11	0.06±0.09	+
f14	0±0	0±0	+	0±0	0±0	+
f15	0.13±0.35	0.99±0.53	-	86.72±8.63	37.75±6.16	+
f16	0±0	0±0	=	17.11±2.25	2.18±0.84	+
f17	0±0	0±0	-	0±0	1.65±0.92	-
f18	0±0	0±0.01	-	0.03±0.04	5.4±4.55	-
f19	0.08±0.05	0.16±0.13	=	3.6±0.37	0.61±0.32	+
f20	0±0	0.02±0.06	=	0.13±0.07	1.09±0.1	-
f21	0±0	0±0	+	0.17±0.49	0.05±0.18	+
f22	0±0	0±0	+	1.14±1.18	0.23±0.34	+
f23	0.37±0.26	0.02±0.03	+	1.96±0.21	0.28±0.1	+
f24	3.36±2.32	3.02±1.46	=	104.88±10.13	30.05±8.84	+

Table A.12 Comparison of mean function values and standard errors for final solutions of PD and PDa with results of Wilcoxon Signed Rank Test on COCO benchmark set. "+" denotes significantly better result of PD, "=" denotes no significant difference, and "-" denotes significantly better result of PD

	D5 PD	D5 PDa	D5	D20 PD	D20 PDa	D20
f1	0±0	0±0	=	0±0	0±0	+
f2	0±0	0±0	=	0±0	0±0	+
f3	0.13±0.35	0.27±0.46	=	26.4±4.6	27±6.15	=
f4	0.53±0.51	0.86±0.35	=	37.28±7.1	35.42±6.2	=
f5	0±0	0±0	=	0±0	0±0	=
f6	0±0	0±0	+	0±0	0±0	=
f7	0±0	0±0	+	3.15±1.23	3.87±1.33	=
f8	0±0	0±0	+	0±0	0±0	+
f9	0±0	0±0	=	0±0	0±0	=
f10	0±0	0±0	=	3.64±3.37	0.12±0.17	+
f11	0±0	0±0	+	0±0	0±0	=
f12	0±0	0±0	=	0±0	0±0	=
f13	0±0	0±0	=	0.06±0.09	0.02±0.03	=
f14	0±0	0±0	=	0±0	0±0	+
f15	0.99±0.53	0.99±0.53	=	37.75±6.16	42.85±7.68	-
f16	0±0	0±0	=	2.18±0.84	2.73±0.64	=
f17	0±0	0±0	-	1.65±0.92	1.23±0.42	=
f18	0±0.01	0.01±0.01	=	5.4±4.55	4.39±1.5	=
f19	0.16±0.13	0.08±0.07	+	0.61±0.32	0.62±0.31	=
f20	0.02±0.06	0.03±0.08	=	1.09±0.1	1.04±0.11	=
f21	0±0	0±0	=	0.05±0.18	0±0	=
f22	0±0	0±0	=	0.23±0.34	0.45±0.69	=
f23	0.02±0.03	0.01±0.02	=	0.28±0.1	0.25±0.09	=
f24	3.02±1.46	1.99±1.28	+	30.05±8.84	33.99±7.68	=

Table A.13 Comparison of mean function values and standard errors for final solutions of PD and PDLP with results of Wilcoxon Signed Rank Test on COCO benchmark set. "+" denotes significantly better result of PD, "=" denotes no significant difference, and "-" denotes significantly better result of PD

	D5 PD	D5 PDLP	D5	D20 PD	D20 PDLP	D20
f1	0±0	0±0	+	0±0	0±0	+
f2	0±0	0±0	=	0±0	0±0	=
f3	0.13±0.35	0±0	+	26.4±4.6	6.57±1.95	+
f4	0.53±0.51	0±0	+	37.28±7.1	13.2±3.39	+
f5	0±0	0±0	+	0±0	0±0	+
f6	0±0	0±0	=	0±0	0±0	=
f7	0±0	0±0	=	3.15±1.23	2.3±0.79	+
f8	0±0	0±0	=	0±0	0±0	=
f9	0±0	0±0	=	0±0	0±0	=
f10	0±0	0±0	=	3.64±3.37	2.89±4.04	=
f11	0±0	0±0	=	0±0	0±0	=
f12	0±0	0±0	=	0±0	0±0	=
f13	0±0	0±0	=	0.06±0.09	0.09±0.14	=
f14	0±0	0±0	=	0±0	0±0	=
f15	0.99±0.53	0.93±0.7	=	37.75±6.16	39.64±7.4	=
f16	0±0	0±0	=	2.18±0.84	2.21±0.7	=
f17	0±0	0±0	=	1.65±0.92	1.21±0.58	=
f18	0±0.01	0.04±0.09	=	5.4±4.55	3.79±1.82	=
f19	0.16±0.13	0.11±0.13	=	0.61±0.32	0.75±0.36	=
f20	0.02±0.06	0±0	=	1.09±0.1	0.89±0.12	+
f21	0±0	0±0	=	0.05±0.18	0.06±0.24	=
f22	0±0	0±0	=	0.23±0.34	0.4±0.69	=
f23	0.02±0.03	0±0.01	=	0.28±0.1	0.3±0.11	=
f24	3.02±1.46	4.56±1.26	-	30.05±8.84	41.08±4.07	-

Table A.14 Comparison of mean function values and standard errors for final solutions of PDa and PDLPa with results of Wilcoxon Signed Rank Test on COCO benchmark set. "+" denotes significantly better result of PD, "=" denotes no significant difference, and "-" denotes significantly better result of PDa

	D5 PDa	D5 PDLPa	D5	D20 PDa	D20 PDLPa	D20
f1	0±0	0±0	+	0±0	0±0	+
f2	0±0	0±0	+	0±0	0±0	+
f3	0.27±0.46	0±0	+	27±6.15	1.53±1.24	+
f4	0.86±0.35	0±0	+	35.42±6.2	3.58±2.19	+
f5	0±0	0±0	+	0±0	0±0	+
f6	0±0	0±0	=	0±0	0±0	=
f7	0±0	0±0	=	3.87±1.33	3.53±1.45	=
f8	0±0	0±0	-	0±0	0±0	=
f9	0±0	0±0	=	0±0	0±0	=
f10	0±0	0±0	=	0.12±0.17	0.14±0.34	=
f11	0±0	0±0	=	0±0	0±0	=
f12	0±0	0±0	=	0±0	0±0	=
f13	0±0	0±0	=	0.02±0.03	0.02±0.03	=
f14	0±0	0±0	=	0±0	0±0	=
f15	0.99±0.53	0.8±0.77	=	42.85±7.68	40.66±9.28	=
f16	0±0	0±0	+	2.73±0.64	2.09±0.77	+
f17	0±0	0±0	+	1.23±0.42	1.39±0.49	=
f18	0.01±0.01	0.01±0.03	=	4.39±1.5	3.92±2.18	=
f19	0.08±0.07	0.04±0.03	+	0.62±0.31	0.43±0.21	=
f20	0.03±0.08	0±0	=	1.04±0.11	0.74±0.2	+
f21	0±0	0±0	=	0±0	0.05±0.18	=
f22	0±0	0±0	=	0.45±0.69	0.72±0.71	-
f23	0.01±0.02	0.01±0.03	=	0.25±0.09	0.33±0.11	=
f24	1.99±1.28	2.6±1.9	=	33.99±7.68	37.03±4.44	=

Table A.15 Comparison of mean function values and standard errors for final solutions of PDLP and PDLPa with results of Wilcoxon Signed Rank Test on COCO benchmark set. "+" denotes significantly better result of PD, "=" denotes no significant difference, and "-" denotes significantly better result of PDLP

	D5 PDLP	D5 PDLPa	D5	D20 PDLP	D20 PDLPa	D20
f1	0±0	0±0	=	0±0	0±0	=
f2	0±0	0±0	+	0±0	0±0	+
f3	0±0	0±0	+	6.57±1.95	1.53±1.24	+
f4	0±0	0±0	+	13.2±3.39	3.58±2.19	+
f5	0±0	0±0	=	0±0	0±0	=
f6	0±0	0±0	+	0±0	0±0	=
f7	0±0	0±0	=	2.3±0.79	3.53±1.45	-
f8	0±0	0±0	=	0±0	0±0	+
f9	0±0	0±0	+	0±0	0±0	=
f10	0±0	0±0	=	2.89±4.04	0.14±0.34	+
f11	0±0	0±0	=	0±0	0±0	=
f12	0±0	0±0	=	0±0	0±0	+
f13	0±0	0±0	=	0.09±0.14	0.02±0.03	=
f14	0±0	0±0	=	0±0	0±0	+
f15	0.93±0.7	0.8±0.77	=	39.64±7.4	40.66±9.28	=
f16	0±0	0±0	+	2.21±0.7	2.09±0.77	=
f17	0±0	0±0	-	1.21±0.58	1.39±0.49	=
f18	0.04±0.09	0.01±0.03	=	3.79±1.82	3.92±2.18	=
f19	0.11±0.13	0.04±0.03	+	0.75±0.36	0.43±0.21	+
f20	0±0	0±0	=	0.89±0.12	0.74±0.2	+
f21	0±0	0±0	+	0.06±0.24	0.05±0.18	=
f22	0±0	0±0	=	0.4±0.69	0.72±0.71	=
f23	0±0.01	0.01±0.03	=	0.3±0.11	0.33±0.11	=
f24	4.56±1.26	2.6±1.9	+	41.08±4.07	37.03±4.44	+

Table A.16 Sum of ranks of a given methods among classes of 5D functions of COCO benchmark set

class	DE	PD	PDa	PDLP	PDLPa
separable	276	272	277	153	**94**
low-condition	275	176	119	199	**131**
high-condition	323	198	**177**	213	214
multimodal with global structure	**150**	238	279	238	189
multimodal no global structure	323	215	185	231	**171**

Table A.17 Sum of ranks of a given methods among 5D functions of COCO benchmark set

f	DE	PD	PDa	PDLP	PDLPa
1	70	57	53	**15**	**15**
2	67	45	48	46	**19**
3	64	55	43	41	**22**
4	54	50	63	33	**20**
5	21	65	70	**18**	**18**
6	71	39	33	50	**32**
7	61	49	**27**	54	34
8	71	43	**23**	47	41
9	72	45	36	48	**24**
10	68	36	**28**	47	46
11	68	46	**30**	40	41
12	54	**37**	44	41	49
13	71	43	**32**	45	34
14	62	**36**	43	40	44
15	**28**	47	42	41	36
16	45	43	58	48	31
17	**16**	46	75	40	48
18	**15**	45	60	59	46
19	46	57	44	50	**28**
20	65	46	38	44	**32**
21	71	38	35	51	**30**
22	71	45	**31**	41	37
23	70	40	46	35	**34**
24	46	46	**35**	60	38

Table A.18 Sum of ranks of a given methods among classes of 20D functions of COCO benchmark set

class	DE	PD	PDa	PDLP	PDLPa
separable	213	316	277	178	**100**
low-condition	227	188	**147**	176	162
high-condition	366	219	**155**	220	165
multimodal with global structure	255	216	236	210	**208**
multimodal no global structure	285	208	199	214	**188**

Table A.19 Sum of ranks of a given methods among 20D functions of COCO benchmark set

f	DE	PD	PDa	PDLP	PDLPa
1	74	61	45	**15**	**15**
2	73	52	30	51	**19**
3	**18**	67	68	45	26
4	23	69	66	45	**18**
5	25	67	68	**22**	**22**
6	64	47	**33**	44	37
7	**16**	50	64	39	56
8	72	48	**22**	51	32
9	75	43	**28**	42	37
10	75	55	**24**	47	**24**
11	75	**35**	**35**	**35**	45
12	75	40	**30**	49	31
13	66	41	43	45	**30**
14	75	48	**23**	44	35
15	75	**27**	46	35	42
16	75	35	47	**34**	**34**
17	**15**	60	48	47	55
18	**15**	57	55	48	50
19	75	37	40	46	**27**
20	**15**	66	64	47	33
21	69	46	**34**	40	35
22	51	33	**31**	38	42
23	75	39	**33**	38	40
24	75	**24**	37	51	38