Information Sciences 715 (2025) 122237

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

FDS: Fractal decomposition based direct search approach for continuous dynamic optimization

Arcadi Llanza [a,b], Nadiya Shvai [b], Amir Nakib [a,*]

[a] University Paris Est Créteil (Laboratoire LISSI), Paris, France
[b] Cyclope.ai, Paris, France

ARTICLE INFO

Keywords:
Metaheuristics
Dynamic optimization
Black-box optimization
Fractal decomposition
Continuous optimization

ABSTRACT

Dynamic optimization problems (DOPs) are known to be challenging due to the variability of their objective functions and constraints over time. The complexity of these problems increases further when the frequency of landscape change and the dimensionality of the search space are large. In this work, we propose a novel fractal decomposition-based method designed for DOPs, called FDS. It is a new single solution metaheuristic that introduces a new hypersphere-based space decomposition for efficient exploration, an archive for diversity control, and a pseudo-gradient-based local search (called GraILS) for fast exploitation. Extensive experiments on the well-known and the standard benchmark (the Moving Peak Benchmark: MPB) demonstrate that FDS consistently outperforms state-of-the-art competitors. Furthermore, FDS shows high robustness across diverse scenarios, maintaining superior performance despite variations in key benchmark parameters, such as the severity of landscape shifts, the number of peaks, the dimensionality of the problem, and the frequency of change. FDS achieves the highest average rank across all experiments and demonstrates dominant performance in 19 out of 23 scenarios. The implementation of FDS is available via the following GitHub repository: https://github.com/alc1218/FDS.

1. Introduction

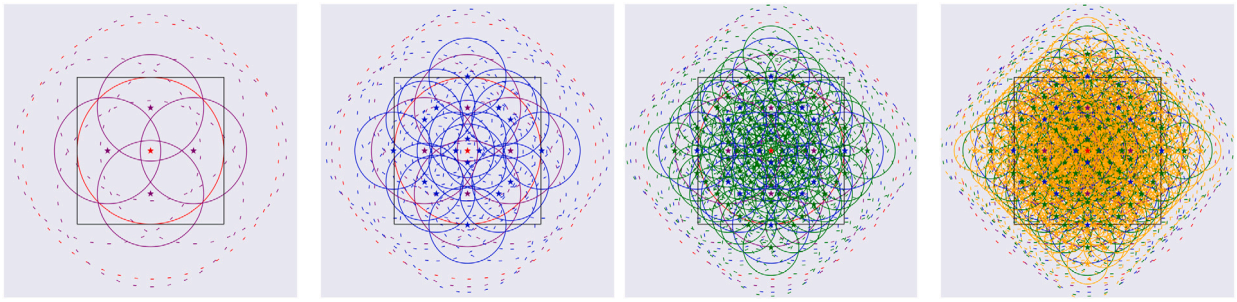
Over the last decade, optimization in dynamic environments has attracted growing interest due to its practical relevance [1]. Indeed, many real-world problems are Dynamic Optimization Problems (DOPs), i.e. their objective function and constraints change over time: typical examples can be found in video processing, vehicle routing, inventory management, and scheduling. Formally speaking, a DOP can be expressed as given in (1), where $f(\vec{x},t)$ is the objective function of a minimization problem, $h_j(\vec{x},t)$ denotes the j^{th} equality constraint, $g_k(\vec{x},t)$ denotes the k^{th} inequality constraint, and u and v are the numbers of equality and inequality constraints, respectively:

$$\begin{aligned} \min \quad & f(\vec{x},t) \\ \text{s.t.} \quad & h_j(\vec{x},t)=0 \text{ for } j=1,2,...,u \\ & g_k(\vec{x},t)\le 0 \text{ for } k=1,2,...,v. \end{aligned} \tag{1}$$

* Corresponding author at: University Paris Est Créteil (Laboratoire LISSI), Paris, France.
E-mail addresses: arcadi.llanza-carmona@u-pec.fr (A. Llanza), nadiya.shvai@cyclope.ai (N. Shvai), nakib@u-pec.fr (A. Nakib).

https://doi.org/10.1016/j.ins.2025.122237
Received 11 November 2024; Received in revised form 19 April 2025; Accepted 19 April 2025

Available online 23 April 2025



(a) 2 levels decomposition (b) 3 levels decomposition (c) 4 levels decomposition (d) 5 levels decomposition

Fig. 1. Visualization of the search space decomposition using the fractal hyperspheres strategy, demonstrating the hierarchical structure and distribution of hyperspheres within the defined boundary.

The dynamic aspect of the problem (1) is reflected in the introduction of the time variable t, which indicates a possible change in functions f, h_j and g_k over time.

These variations can take different forms: in particular, the landscape of the objective function may change after a certain time (frequency of change), the dimension of the problem (number of decision variables), and the size of the search space. *In this paper, the search space considered is only the decision variable space, and the problem itself is considered to be a black-box.* This work considers the online metaheuristic optimization algorithms, *i.e.* algorithms that produce new solutions dynamically in order to react to changes over time.

Particularly complicated cases of DOPs occur when the frequency of change is high and the number of dimensions is high (large-scale problem). These problems are also known as Hard Continuous Dynamic Optimization Problems (HDOPs).

To address these challenges, various population-based approaches have been proposed, with evolutionary algorithms and particle swarm optimization being among the most studied. However, these methods often involve significant design complexity and are tailored to specific problems or scenarios, limiting their generalization to different DOPs. An overview of the relevant literature is provided in Section 2.

To overcome these limitations, this paper proposes a novel approach based on fractal decomposition. The key contributions are as follows:

- A single-solution metaheuristic for solving DOPs,
- A fractal-based search space decomposition using hyperspheres (see Fig. 1),
- A local search method combining Intensive Local Search with a pseudo-gradient descent technique to accelerate convergence,
- An explicit archive mechanism to enhance search history analysis across time periods.

To evaluate the performance of algorithms, benchmarks for Dynamic Optimization Problems (DOPs) are often based on a parametric family of real continuous functions, where variations in the parameters simulate a dynamic environment. To date, relatively few benchmarks have been proposed. The most widely used benchmark for algorithm evaluation is the Moving Peaks Benchmark (MPB) [2], which has become the standard in DOP research.

A recent survey on benchmarks for DOPs can be found in [3]. In this study, experiments were performed using MPB to validate the proposed approach. This choice was motivated by the wide dissemination of this benchmark in the literature.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 introduces in detail the FDS algorithm. Afterwards, section 4 recalls the used benchmark to evaluate the performance of the proposed method. Then, section 5 discusses the different experiments, and the obtained results. Finally, in section 6 the conclusion and work under progress are presented.

2. Related work

Population based metaheuristics have been widely used to optimize DOPs. In this section, only the standard and most efficient algorithms based on the different approaches: Evolutionary Algorithms (EA), Differential Evolution (DE), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Hybrid and Local Search (LS). For a comprehensive review of the state-of-the-art methods, the reader is referred to the recent review proposed by Yazdani et al. [1].

2.1. Evolutionary algorithm

The chaotic Genetic Algorithm (cGA) was introduced by Mohammadpour et al. (2019) in [4]. cGA uses a new memory with diversity maximization. Particularly, cGA proposes a strategy for updating memory and memory retrieval.

Etaati et al. (2022) presented a full-featured cooperative Coevolutionary Memory-based Artificial Immune System (CM-AIS) enhanced by a new clonal selection algorithm in [5]. CM-AIS decomposes the n-dimensional population into n one-dimensional subpopulations. Then, each subpopulation is evaluated separately using a set of context vectors called short-term memory. Fur-

thermore, a memory-based approach called long-term memory is proposed to store and retrieve information when a fitness change occurs.

2.2. Differential evolution

DE optimizes iteratively any cost function by improving a solution against a given quality metric.

A DE algorithm for DOPs (DynDE) was proposed by Mendes et al. in [6]. This algorithm uses a multi-population approach while maintaining each of the populations in one of the objective function's peaks.

A self-adaptive multi-population DE algorithm (jDE) proposed by Brest et al. in [7] uses an exclusion radius to reset overlapping sub-populations.

A Global Replacement-based Differential Evolution (GRDE) with neighbor-based memory was proposed by Zhu et al. (2018) in [8]. In the proposed algorithm, the entire population is composed of several sub-populations, which are evolved independently and excluded each other via a minimal Euclidean-distance. The originality of the algorithm relies on the selection operator in the DE procedure and the strategy of storing historical solutions (not only the historical best solutions) to be used in the later environment.

Kordestani et al. (2019) introduced a framework for enhancing multi-population algorithms for DOPs in [9]. This framework involves integrating scheduling into multi-population methods to allocate more function evaluations to the most effective sub-populations. Two methods are used in this framework, each employing a distinct approach for scheduling the sub-populations. The first method, DynDE + PI, combines sub-population quality and diversity to form a single feedback parameter for identifying the best performing sub-population. The second method, DynDE + LA, utilizes learning automata as the central unit for conducting the scheduling operation.

2.3. Particle swarm optimization

The majority of dynamic optimization algorithms find their essence on the PSO method. It is a population-based approach in which multiple individual (agents), called particles, move in the search space.

Species in a Particle Swarm Optimizer (SPSO) was introduced by Parrott et al. in [10]. It includes a tracking peaks procedure to prevents overcrowding. Many improvement and variations of this method have been proposed in the literature, such as [11].

A multi-strategy ensemble particle swarm optimization (MEPSO) has been proposed by Du et al. in [12] that used a two swarm particles: the first diversifies using a Gaussian LS, while the second intensifies by applying a differential mutation.

In [13], Novoa et al. proposed an operator to control particle trajectories. This method reduces the waste of non-performing particles in different swarms.

Adaptive Multi-Swarm Optimizer (AMSO) was proposed by Li et al. (2014) in [14]. AMSO permits maintaining population diversity and adapting the number of populations without requiring a change detection mechanism.

Kazemi et al. (2019) presented a note on the exclusion operator in multi-swarm PSO algorithms in [15]. The exclusion operator plays a crucial role in delineating the search territory of each population in multi-population optimization algorithms, aiming to maintaining overall population diversity and prevent redundant search efforts. Kazemi introduces four strategies (Algorithm1, Algorithm2, Algorithm3, and Algorithm4) designed to mitigate the limitations associated with the exclusion operator.

sslPSO was proposed by Shen et al. (2020) in [16]. A species conservation combined with a spatial neighborhood best searching strategy (type of local search) is integrated into canonical PSO to finally create sslPSO.

In [15], Kazemi et al. proposed 4 algorithms extended from the base method mPSO. These modifications were designed with the aim of improving the efficiency and reducing the wastage of computing resources due to exclusion zones. Particularly, this work has focused on the problematic where many peaks (local optimum) are available at the same time.

In [17], Liu et al. presented a multipopulation algorithm based on the Affinity Propagation Clustering Particle Swarm Optimization (APCPSO). This method automatically creates sub-populations by message-passing process, which can avoid some extra parameters. In this strategy, the best particles in each sub-population are first stored in a memory. Then, a local search is applied for helping the memory to quickly locate new peaks, if an environmental change has occurred.

Li et al. introduced a rapid Density Peak Clustering-based Particle Swarm Optimizer (DPCPSO) (2024) in [18]. DPCPSO incorporates three key components. First, a fast density peak clustering to generate multiple sub-populations, helping in peak identification. Second, a stagnation detection to address the diversity loss. Third, an optimal particle calibration strategy is proposed for swiftly finding optimal solutions in changing environments. Additionally, the hill climbing method quickly identifies new peaks when the environment changes.

2.4. Ant colony optimization

Dynamic Hybrid Continuous Interacting Ant Colony (DHCIAC) was proposed by Dréo et al. in [19]. DHCIAC is based on an hybrid population-based ant colony algorithm that considers an interacting ant colony with a Nelder-Mead algorithm.

A charged ant colony algorithm for continuous dynamic optimization (CANDO) was presented by Tfaili et al. in [20]. To maintain the ant population diversification, CANDO attributes a repulsive electrostatic charge to each ant. This allows them to keep a distance from each other. The algorithm uses a weighted continuous Gaussian distribution instead of a discrete distribution to solve DOPs.

Differential Ant-Stigmergy Algorithm (DASA) was proposed by Korosec et al. [21]. DASA is a stigmergy-based algorithm for solving optimization problems with continuous variables.

2.5. Hybrid and local search

A Local search algorithm is defined as an heuristic method that moves from solution to solution in the search space by applying local changes, until a (sub)optimal solution is reached.

Multi-phase Multi-individual version of the Extremal Optimization algorithm (MMEO) was proposed by Moser et al. in [22]. The idea behind Extremal Optimization was inspired from [23] initially proposed by Boettcher et al. MMEO uses extremal optimization component to select initial solutions for the LS procedure, that is based on the hill-climbing technique.

In [24], Pelta et al. presented a decentralized multi-agent strategy that focuses on systems of cooperation and diversity mechanisms.

Multiple Local Search algorithm for Dynamic Optimization (MLSDO) was proposed by Lepagnot et al. in [25]. MLSDO uses multiple coordinated local searches. It keeps an archive of found optima that is used when an environment change is detected.

The Bi-flight cuckoo search algorithm (BfCS-wVN) was introduced by Kordestani et al. (2018) in [26]. BfCS-wVN has changed the Levy flight technique to adaptive flight based on two models that are learned by an automaton in order to control the global and LS abilities. At the global search step, the cuckoo mechanism is used to generate new candidates in areas that have not yet been explored. At the LS step, the nesting scheme variable and the Hooke-Jeeves pattern search are used to search intensively around the most promising areas of the search space.

Collaborative Evolutionary-Swarm Optimization (CESO) algorithm is presented in [27], by Lung et al. CESO combines the search abilities of an EA and PSO to track moving optima in DOPs.

Evolutionary Swarm Cooperative Algorithm (ESCA) has been introduced by Lung et al. in [28]. ESCA is an extension of the previously introduced method CESO. ESCA uses two populations of EAs to maintain search diversity, and one PSO population to exploit the area around the best position in the search phase.

A Cluster-based Clonal Selection Algorithm (CCSA) was introduced by Zhang et al. (2019) in [29]. The population in CCSA is initially partitioned into multiple clusters based on spatial locations. Each cluster is then evolved independently using a learning-based clonal selection algorithm. This algorithm incorporates a learning strategy within the cluster and introduces interactions among clusters into a hypermutation operator to enhance search capability. Furthermore, a memory mechanism is introduced to store previous search information, which can be reused for optimal tracking after environmental changes, contributing to the algorithm adaptability.

Qin et al. (2021) presented in [30] an Adaptive multi-swarm algorithms called AdaMPSO and AdaMDE. This framework is based on a multi-swarm approach where a basic adaptation is the combination of a group of active swarms and a group of inactive swarms. The first group focuses in exploring new areas of the search space, and the second group is dedicated to preserve the current optima. An inactive swarm will become active and search for new optima again when an environmental change occurs. Finally, this framework uses a local search to the best individual of every stagnated swarm.

A Hybrid Multi-Population Reinitialization Strategy (HMRS) was proposed by Raghul et al. (2023) in [31]. HMRS is a combination of a distributed differential evolution framework and a re-initialization strategy. It reproduces sub-populations in new environments influenced by archival information.

Rezvanian et al. (2024) introduced a Cellular Automata-based Artificial Immune System named CaAIS in [32]. This algorithm employs antibodies, representing nominal solutions, distributed across a cellular grid corresponding to the search space. These antibodies generate hyper-mutation clones at various times through interactions with neighboring cells, producing diverse solutions. Local interactions between neighboring cells facilitate the propagation of near-best parameters and near-optimal solutions throughout the search space. The most effective antibodies are retained within each cell, while weaker antibodies are replaced until the stopping criteria are met.

3. Proposed method

In this work, the goal is to develop a robust and adaptive optimization algorithm capable of efficiently handling dynamic environments.

The Fractal Decomposition-Based approach was proposed in [33]. It divides the feasible search space into sub-regions with the same geometrical pattern. At each iteration, the most promising regions are selected and then decomposed. However, this method has not yet been adapted to DOPs.

The fractal decomposition based strategy is based on the following procedures:

1. A fractal space decomposition via a given pattern.
2. An heuristic to select promising regions.
3. A local search as intensification strategy (IS) to find the best solution so far.

In this section, the proposed algorithm, called FDS, is presented and its components are detailed. FDS uses hypersphere pattern to decompose the search space as in [33], and a new local search method called GrAILS that is used as the intensification strategy. The latter combines a classical pattern search, and a pseudo-gradient descent.

In addition to the strategies, the following methods have been included to effectively deal with DOPs:

- An archive that is used during the search.

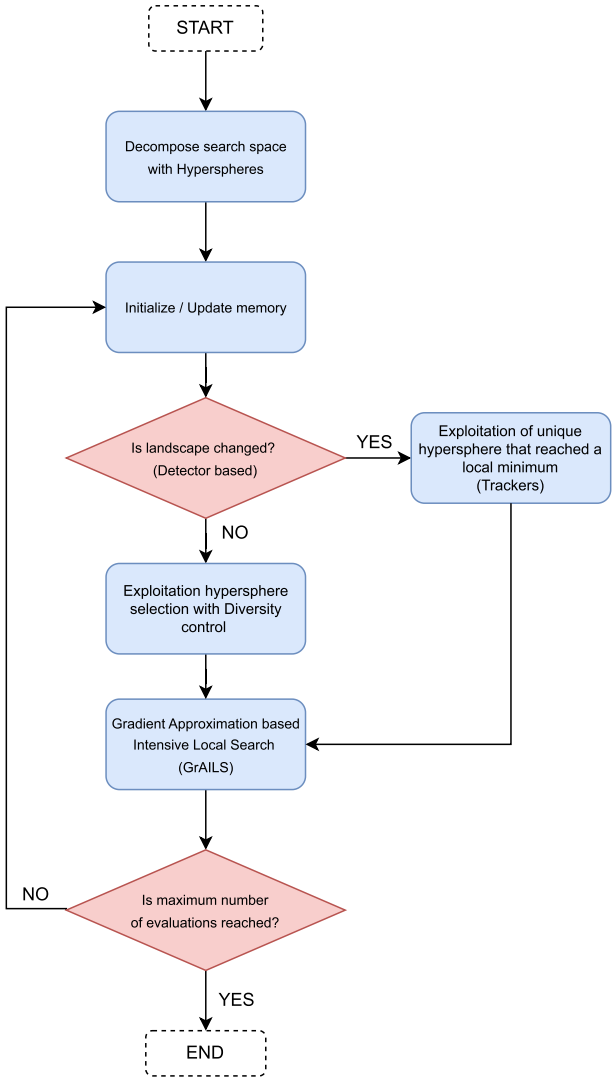


Fig. 2. The "Diversity Control" procedure guides the exploration phase, ensuring balanced search space coverage, while the "GrAILS" procedure handles the exploitation phase.

- A tracking technique based on detectors to ensure that a change in the landscape is identified and the best solutions are recovered fast.

3.1. FDS: global description

FDS starts with the *fractal-based space decomposition* which results in a search tree of hyperspheres (see subsection 3.2). The hyperspheres centers constitute a set of potential starting points for the exploitation phase. When a hypersphere is exploited, a single local optimum is reached. Therefore, by exploiting many hyperspheres, FDS can reach many local optima.

The choice of the next exploitation starting point (hypersphere center) is given to the hyperspheres furthest from those that have not yet been already exploited to ensure the diversity. For the exploitation phase a *local search with gradient approximation based on pattern search* is used (see subsection 3.4). Finally, to share the information about locations of the best solutions found so-far to the next landscape, the *tracking* is used (see subsection 3.6). The latter uses explicit memory to store data about the locations of the solutions and their corresponding fitness. The illustration of FDS flowchart is presented in Fig. 2.

3.2. Space decomposition

The proposed space decomposition enables better exploration by effectively subdividing the search space into smaller regions based on a fractal strategy. Then, the search space is modeled by a set of key positions (centers of the hyperspheres) that follow a fractal distribution.

Assume that the search space is defined by $I = [L_1, U_1] \times [L_2, U_2] \times \ldots \times [L_D, U_D] \subset \mathbb{R}^D$. Without loss of generality, we can consider normalized search space $I = [-0.5, 0.5]^D$.

1st level of decomposition. We initialize with one hypersphere with center $C_0 = (0, \ldots, 0)$ in the origin and radius $r_0 = 0.5$.

k-th level of decomposition. Let point C_{k-1} be a center of a hypersphere with radius r_{k-1} on the $k-1$ level of decomposition. Then the centers of its $2D$ child hyperspheres can be obtained as in Eq. (2):

$$\begin{aligned} C_k^{i+} &= C_{k-1} + \delta \cdot r_{k-1} \cdot \vec{e}_i, \quad i = 1..D, \\ C_k^{i-} &= C_{k-1} - \delta \cdot r_{k-1} \cdot \vec{e}_i, \quad i = 1..D, \end{aligned} \tag{2}$$

where $(\vec{e}_1, ..., \vec{e}_D)$ are the unit basis vectors

$$\vec{e}_j = (\underbrace{0, \ldots, 0}_{j-1}, 1, \underbrace{0, \ldots, 0}_{D-j}), \quad j = 1..D,$$

and $\delta = \frac{\sqrt{2}}{1+\sqrt{2}} \approx 0.5858$. The inflated radius r_k is calculated as

$$r_k = \alpha(1 - \delta) r_{k-1},$$

where inflation coefficient α is used to ensure full space coverage. Following Nakib et al. [33], we use $\alpha = 2.185$.

Note that as every hypersphere has $2D$ child hyperspheres, the *k-th* level of decomposition contains $(2D)^{(k-1)}$ hyperspheres. Compared to the grid decomposition where each hypercube is split in half along each axis, the latter yields a much larger amount of $2^{D(k-1)}$ hypercubes.

For $2D$ search space the hypersphere space decomposition up to the level 5 is illustrated by Fig. 1.

Finally, to relax the rigidness in the fractal decomposition structure, a random rotation in Euclidean space has been applied to each hypersphere *w.r.t.* its parent hypersphere. The rotation matrix (transformation matrix) Q is obtained as the orthonormal matrix from the QR decomposition of a random $D \times D$ matrix $A = QR$ every time a landscape change has been identified.

3.3. Diversity control procedure (selecting promising regions)

Diversity control procedure used in FDS is motivated by the idea of giving preference to the unexplored areas of the search space.

More precisely, priority is given to the furthest hyperspheres from those already exploited. The selection of the best hyperspheres procedure is presented in the following:

1st hypersphere selection: The root hypersphere (that has center $C_0 = (0, \ldots, 0)$ in the normalized search space) is exploited first.

k-th hypersphere selection: Let $\mathcal{W}_{ex} = \{C^{(1)}, \ldots, C^{(k-1)}\}$ be a set of centers of exploited (visited) hyperspheres, and $\mathcal{W}_{non\_ex}$ be a set of centers of non-exploited (non-visited) hyperspheres. Then, the goal is to select from ($\mathcal{W}_{non\_ex}$) the furthest non-exploited hypersphere to be exploited:

$$\begin{aligned} C^{(k)} &= \underset{C \in \mathcal{W}_{\text{non\_ex}}}{\arg\max}\ \text{dist}(C, \mathcal{W}_{\text{ex}}) \\ &= \underset{C \in \mathcal{W}_{\text{non\_ex}}}{\arg\max} \min_{C^{\text{ex}} \in \mathcal{W}_{\text{ex}}} \text{dist}(C, C^{\text{ex}}), \end{aligned} \tag{3}$$

where $dist$ is Euclidean distance. Sets $\mathcal{W}_{ex}$ and $\mathcal{W}_{non\_ex}$ are updated by adding and subtracting point $C^{(k)}$ accordingly:

$$\begin{aligned} \mathcal{W}_{ex} &:= \mathcal{W}_{ex} \cup \{C^{(k)}\} \\ \mathcal{W}_{non\_ex} &:= \mathcal{W}_{non\_ex} \setminus \{C^{(k)}\} \end{aligned} \tag{4}$$

Consequently, the distance calculation defined in (3) can be significantly facilitated by saving in memory the distances calculated during the previous selection steps.

3.4. Local search with gradient approximation

While the diversity can be maintained during the visit of the hypersphere centers, the intensification is carried out via a local search which must be fast and take advantage from the visited positions. To do so, a line search based algorithm is used in this paper. Its principle is based on finding the most efficient direction from a set of those produced by varying one coordinate at a time. In order to speed up the search, this heuristic is augmented by a gradient approximation and a consequent pseudo-gradient descent. More specifically, once the coordinates axes directions have been explored, the produced set of evaluated points is used for gradient approximation. The goal is to skip some intermediate steps based on the assumption that previously explored points (in the line search) contain actionable information on a promising direction. Then, this efficient sampling strategy for a gradient approximation will improve the current point fitness score. This proposed hybrid method is called: Gradient Approximation based ILS (GrAILS).

More precisely, GrAILS consists of two strategies that are applied consequently:

1. vanilla ILS;

2. pseudo-gradient descent.

Vanilla ILS. Let $x_c \in \mathbb{R}^D$ be a starting point, and let γ be a step size. One iteration step is a loop over unit basis vectors $(\vec{e_1}, ..., \vec{e_D})$, where the current position x_c is updated via selection among $(x_c - \gamma\vec{e_i}, x_c + \gamma\vec{e_i}, x_c)$ according to the best fitness function values. As a result, a search trajectory at the end of this phase will consist of maximum $D+1$ points, with a total of $2D+1$ points explored.

Pseudo gradient descent. Let x_c be the current local search position, and $(x_1, ..., x_{2D})$ be $2D$ previously explored points stage. We would like to obtain gradient approximation in the current point x_c in order to perform a line search. To this aim, we write down the linear approximation of the fitness function at the point x_c corresponding to the first two terms of Taylor series:

$$\begin{cases} f(x_1) = f(x_c) + \nabla f(x_c)^T (x_1 - x_c) \\ \vdots \\ f(x_{2D}) = f(x_c) + \nabla f(x_c)^T (x_{2D} - x_c) \end{cases} \tag{5}$$

Equation (5) can be considered as a system of linear equations with respect to the gradient $\nabla f(x_c)$. In a more compact way,

$$X^T \nabla f(x_c) = F,$$

where $X = (x_1 - x_c; ...; x_{2D} - x_c)$ is a $D \times 2D$ matrix, and $F = (f(x_1) - f(x_c); ...; f(x_{2D}) - f(x_c))$ is a $2D$ vector. Remark that number of equations $2D$ is larger than the searched vector $\nabla f(x_c)$ length D. We approximately solve this system of equations with the least squares method. Its solution corresponds to Moore-Penrose matrix inverse[1] [34]:

$$\nabla f(x_c) = (X^T)^+ F.$$

The obtained gradient approximation provides a search direction:

$$v_{grad}^{unit} = -\frac{\nabla f(x_c)}{||\nabla f(x_c)||}.$$

We update current position x_c by:

$$x_c := x_c + \gamma v_{grad}^{unit}$$

The full description of the proposed algorithm is detailed in figure Algorithm 1.

It can be noticed that trackers can initiate an early stop of GrAILS if its trajectory is nearby one of the tracked solutions. As this functionality refers rather to the use of trackers, it is detailed in the corresponding subsection 3.6.

To illustrate the speedup of GrAILS, a convex quadratic equation (6) was used as an example to compare the performance of the vanilla ILS and the GrAILS methods. Fig. 3 presents a speedup of almost 2.5 times obtained by GrAILS *wrt* vanilla ILS. GrAILS reaches the minimum in 65 function evaluations while vanilla ILS gets to the optimum in 161 iterations. We notice that GrAILS reaches the optimum by repeating the previously mentioned strategy combination 4 times (ILS + pseudo-gradient descent).

$$\begin{aligned} \min \quad & \sum_{i=1}^{2} x_i^2 \\ \text{s.t.} \quad & -11 \le x_1 \le 7 \\ & 2 \le x_2 \le 10 \end{aligned} \tag{6}$$

Finally, we present here two strategies that GrAILS uses:

1. the number of points used for pseudo-gradient estimation could be reduced in order to decrease the amount of function evaluations used, or increased to improve the estimate. Originally it was proposed to use $2xD$ points explored through vanilla ILS stage. However, experiments have shown that using D points (obtained by randomly choosing a direction alongside each of the axes on ILS stage) is more efficient.
2. GrAILS implies moving in a single direction, the early stopping by trackers might be predicted before executing GrAILS by measuring the distance from the tracked solutions to the GrAILS direction line.

The positive effect of these two strategies is given in Section 5.2 where the ablation study of the proposed method was performed.

[1] We recall here that Moore-Penrose matrix inverse is a generalization of matrix inverse that exists and is unique for any real rectangular matrix. In case matrix is invertable its Moore-Penrose matrix inverse is equal to the matrix inverse. For a real full-rank matrix A its Moore-Penrose inverse can be calculated as $A^+ = (A^T A)^{-1} A^T$. Application of Moore-Penrose matrix inverse allows to obtain (one of) least squares problem solutions.

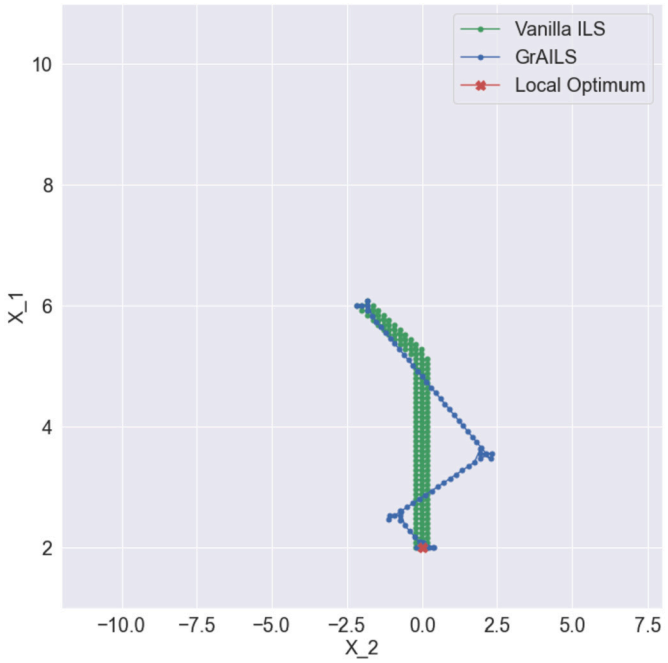


Fig. 3. Vanilla ILS vs GrAILS (proposed gradient approximation ILS) methods comparison. Both methods start from the center of the search space and converge to the local optimum (marked with a red cross). Vanilla ILS (in green) reaches the minimum in 161 FEs, while GrAILS (in blue) achieves the same result in only 65 FEs, demonstrating a 2.5× speedup compared to vanilla ILS.

3.5. Detection of landscape shift

FDS uses a detectors based approach to identify a shift in the landscape. This is a classical technique that consists in re-evaluating the knowledge acquired within the optimization procedure. The known samples are supposed to do not change (or just mildly in case that a function is noisy) within the same period. Particularly, FDS uses the optima identified during the search as samples to detect the landscape shift. The main advantage of such choice is that when the landscape changes, the newly obtained fitness for the optimum will be used for the tracker as a starting point. FDS triggers the detector based event every time the GrAILS (intensification) stage has finished. This ensures that a local or global optimum is always available to be used for comparison.

3.6. Tracking local optima

A tracking is used in FDS to ensure that information about the local optima found is passed into the next landscape. Once a shift of the landscape is detected, the best solutions are re-tracked using GrAILS. The stagnation criteria of the latter ($\gamma^{tr}_{min} = 1e^{-5}$) and the step size ($\gamma^{tr} = 1e^{-4}$) are reduced to ensure that the local search is performed in the neighborhood.

Let us now explain the tracking procedure in more detail. We note by $\mathcal{T}$ a set of tracked points that correspond to the best solutions found. A straightforward strategy would be to store in $\mathcal{T}$ the convergence points of GrAILS. When employing it, we have observed a problem of having multiple found solutions that in fact correspond to the same true local optima. Such excessive tracking leads to wasting evaluations during solutions re-tracking in the beginning of the landscape period, and would be beneficial to avoid. To solve this problem, one could try to merge points in $\mathcal{T}$ if the distance between them is less than a certain exclusion radius r_{excl}, thus introducing a form of a *density control mechanism*. A more proactive strategy would be to try to catch the trajectories of the local search early that are approaching one of the solutions already tracked, thus avoiding unnecessary feature evaluations during the operation phase. This strategy is used in FDS.

We start with an empty set of tracked points $\mathcal{T}$. At each step of GrAILS, we check if one the following conditions is met:

1. The current position of GrAILS trajectory x_c is within an exclusion radius $r_{excl} = 10\gamma_{min}$ of one of the currently tracked points $x_{tr} \in \mathcal{T}$. In this case, we verify if points x_c and x_{tr} can be merged into their mean $x_{mean} = 0.5(x_c + x_{tr})$, or should both of them be tracked as independent solutions [35]. To this aim we compare corresponding fitness values $f(x_c), f(x_{tr}), f(x_{mean})$ which can produce the following outcomes:
   (a) $f(x_{mean}) > \max\{f(x_c), f(x_{tr})\}$. In this case point x_{mean} is added to the set $\mathcal{T}$ instead of the point x_{tr};
   (b) $f(x_{tr}) \geq f(x_{mean}) \geq f(x_c)$. In such case the set $\mathcal{T}$ is not updated;
   (c) $f(x_c) \geq f(x_{mean}) \geq f(x_{tr})$. In this case point x_c is added to the set $\mathcal{T}$ instead of the point x_{tr};
   (d) $f(x_{mean}) < \min\{f(x_c), f(x_{tr})\}$. In such case point x_c is added to the set $\mathcal{T}$, and point x_{tr} remains in $\mathcal{T}$ as well.
   In all these cases GrAILS is stopped early.

Algorithm 1: Gradient Approximation based Intensive Local Search (GrAILS).

Input: Starting point x_c (array of floats of length D); step size γ (float); step size decrease factor $\gamma\_decrease$ (float);
Output: Best position so far x_best (array of floats of length D); corresponding fitness score f_best (float);

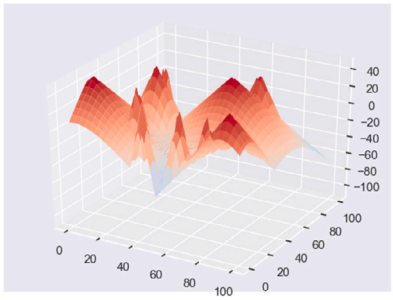
$current\_fitness \leftarrow evaluate\_performance(x\_c)$;
while *stopping_criteria is not reached* **do**
  $initial\_fitness \leftarrow current\_fitness$;
  /* Initialization of FIFO Stack with 2D+1 elements */
  $all\_fitness \leftarrow [current\_fitness]$; $all\_positions \leftarrow [x\_c]$;
  /* Vanilla ILS component */
  for idx *in range(D)* **do**
    /* Negative step size */
    $x\_c\_n \leftarrow copy(x\_c)$; $x\_c\_n[idx] \leftarrow x\_c\_n[idx] - \gamma$;
    $fitness\_n \leftarrow evaluate\_performance(x\_c\_n)$;
    /* Save information for pseudo-gradient step */
    $all\_fitness.append(fitness\_n)$;
    $all\_positions.append(x\_c\_n)$;
    /* Positive step size */
    $x\_c\_p \leftarrow copy(x\_c)$; $x\_c\_p[idx] \leftarrow x\_c\_p[idx] + \gamma$;
    $fitness\_p \leftarrow evaluate\_performance(x\_c\_p)$;
    /* Save information for pseudo-gradient step */
    $all\_fitness.append(fitness\_p)$; $all\_positions.append(x\_c\_p)$;
    /* Get best position and result */
    $fitness\_scores \leftarrow [fitness\_n, fitness\_p, current\_fitness]$; $positions \leftarrow [x\_c\_n, x\_c\_p, x\_c]$; $best\_idx \leftarrow get\_best\_fitness\_idx(fitness\_scores)$;
    /* Assign best result to current state */
    $current\_fitness \leftarrow fitness\_scores[best\_idx]$; $x\_c \leftarrow positions[best\_idx]$;
  end for
  /* Pseudo gradient descent component */
  $x\_1, \ldots, x\_2D \leftarrow remove\_current\_position(all\_positions)$;
  $f\_1, \ldots, f\_2D \leftarrow remove\_current\_fitness(all\_fitness)$;
  $X \leftarrow (x\_1 - x\_c; \ldots; x\_2D - x\_c)$;
  $X\_T \leftarrow transpose(X)$;
  $F \leftarrow (f\_1 - f\_c; \ldots; f\_2D - f\_c)$;
  /* Moore-Penrose inverse application */
  $grad\_approxim \leftarrow moore\_penrose\_inverse(X\_T) * F$;
  $grad\_unitary\_vector \leftarrow -grad\_approxim / \|grad\_approxim\|$;
  /* Descent in pseudo-gradient direction */
  repeat
    $previous\_fitness \leftarrow current\_fitness$; $previous\_x\_c \leftarrow x\_c$;
    /* Apply one step towards best direction */
    $x\_c \leftarrow x\_c + (grad\_unitary\_vector * \gamma)$; $current\_fitness \leftarrow evaluate\_performance(x\_c)$;
    /* Update FIFO stack */
    $all\_fitness.append(current\_fitness)$; $all\_positions.append(x\_c)$;
  until $current\_fitness < previous\_fitness$;
  /* Update with the best values */
  $current\_fitness \leftarrow previous\_fitness$; $x\_c \leftarrow previous\_x\_c$;
  if $current\_fitness == initial\_fitness$ **then** $\gamma \leftarrow \gamma * \gamma\_decrease$;
end while
$x_{best} \leftarrow x_c$;
$f_{best} \leftarrow current\_fitness$;
return x_best, f_best;

2. GrAILS has converged to a point x_{conv}, and condition (1) does not hold. Then point x_{conv} is added to the set $\mathcal{T}$. Once a landscape shift is detected, GrAILS with decreased parameters of stagnation criteria ($\gamma^{tr}_{min} = 1e^{-5}$) and the step size ($\gamma^{tr} = 1e^{-4}$) is performed to re-track solutions from $\mathcal{T}$. The local searches are performed sequentially according to the fitness values of points from $\mathcal{T}$ calculated in the current period.

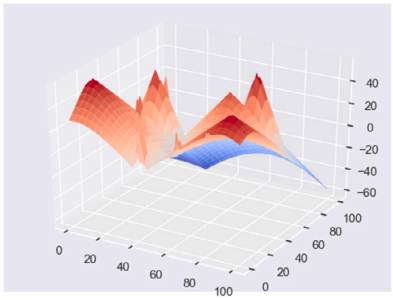
In summary, FDS algorithm integrates fractal decomposition, LS techniques, and memory mechanisms, offering a distinct approach compared to existing hybrid methods. While algorithms such as MMEO, MLSDO, and CHPSO combine EA, PSO, and multi-agent systems to enhance search efficiency and maintain diversity, FDS follows a more structured exploration strategy. It applies fractal decomposition with hyperspheres to ensure comprehensive and flexible search space coverage, particularly in dynamic environments. Unlike traditional hybrid methods that rely on population-based approaches, FDS utilizes a single-solution metaheuristic.

4. Experiment setting

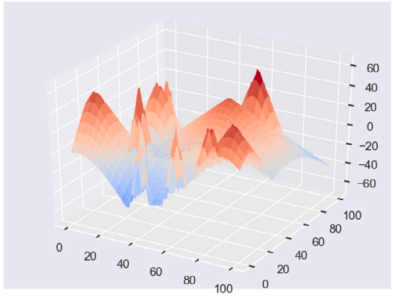
As mentioned in the introduction, in this work, we use the well-known Moving Peaks Benchmark (MPB).



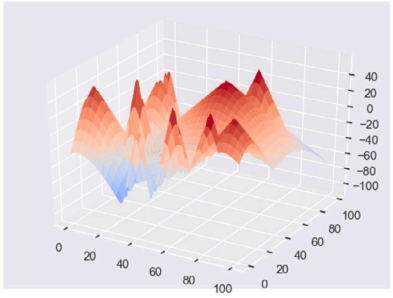
(a) 1st environment shift



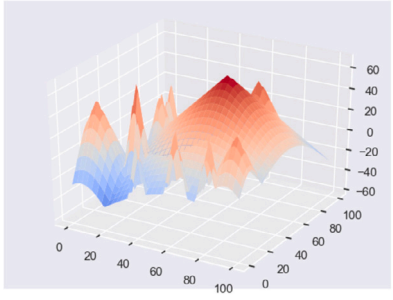
(b) 2nd environment shift



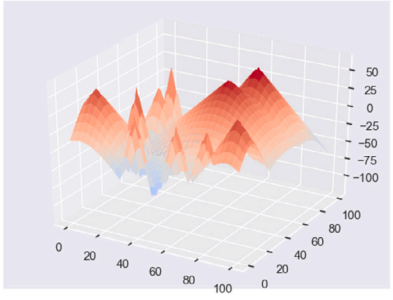
(c) 3rd environment shift



(d) 4th environment shift



(e) 5th environment shift



(f) 6th environment shift

Fig. 4. 2D MPB landscape in the scenario 2. The subfigures illustrate six consecutive landscape shifts, corresponding to 30,000 FEs. Each variable is constrained within the range from 0 to 100, demonstrating the dynamic nature of the optimization environment.

4.1. Recall of the moving peaks benchmark

The MPB [2] is a family of problems for benchmarking dynamic optimization algorithms. So far, MPB has been established as the most frequently used benchmark for testing dynamic single-objective optimization problems. It contains three different scenarios, with scenario 2 which is the most common scenario for comparing different algorithms (see Fig. 4).

In Table 1 the MPB parameters that define the scenario 2 are described. The landscape in this scenario is built upon a base function containing multiple peaks (by default 10) that are shaped after equation (7) with a defined width $[1, 12]$ and height $[30, 70]$ limit range.

This benchmark is expressed as a maximization problem where the different set of functions (peaks) vary randomly their shape (width and height) and position upon time. The primary task to solve the problem is to follow the topmost peak in the landscape while it moves in space over time. On any landscape shift, any of the sub-optimal peaks can develop into the new highest peak.

Table 1
Default parameters of MPB scenario 2.

Parameter	Scenario 2
Number of peaks	10
Dimension D	5
Peak height	[30,70]
Peak widths	[1,12]
Change cycle α	5000
Change severity s	1
Height severity	7
Width severity	1
Correlation coefficient	0
Number of changes N_c	100

Cone function used in scenario 2 is the following:

$$f(\vec{x},t) = H_i(t) - W_i(t)\sqrt{\sum_{i=1}^{D}(x_i - P_i(t))^2} \tag{7}$$

where $\vec{x}$ are the coordinates provided by the DOA at the function evaluation t. The problem dimension D, the cone peaks coordinates P, the height H, and the width W of each peak are initialized according to Table 1.

4.2. Performance measures

DOPs performance can be categorized into *fitness/error based* and *efficiency based* performance indicators. In the case of MPB, the metric used is fitness/error based type. This measure requires the information of the global optimum in each environment to be calculated. Although this value is not available in many real-world applications, this information is provided by most DOP benchmarks.

The concepts of the online and offline performances were proposed in [36] by De Jong. These performance indicators diverge in the fact that the online performance is based on all solutions, while the offline performance is based only on the best solution known so far. A drawback observed in [37] was that these measurements are only useful in the context where the value of the best solution remains constant over time (*i.e.* change of location scenario). Grefenstette tackled this problem by subtracting the current performance from the optimal solution that can be found in the DOP benchmark. These performance measures were called *online* and *offline errors*.

Current error evaluation metric is computed as the difference between the theoretical optimum of the current period and the best solutions found so far:

$$Current\ Error(t) = opt(t) - \vec{x}_{best}(t) \tag{8}$$

Offline error evaluation metric is defined as the average *Current error* over all evaluation steps:

$$Offline\ Error = \frac{\sum_{t=1}^{T} Current\ Error(t)}{T} \tag{9}$$

where T is the total number of FEs.

The offline error is the generally agreed upon metric to measure performance in dynamic environments, despite some of its defects, such as: a/ this metric is known to be easily affected by the average quality of all available solutions (the average height of the landscape); b/ it promotes algorithms that find decent solutions faster; and c/ it distorts the final result over the set of evaluations by cause of the averaging factor.

4.3. MPB scenarios

To study the FDS generalization and scalability, different MPB scenarios were considered through the modification of the benchmark parameters. These scenarios were taken from the literature and related work to maximize the comparative base.

The different scenarios considered are based on the variations of the following MPB parameters:

Severity of landscape shift: this parameter denotes the measure of the landscape change after the shift. Its value ranges from 1 to 6, lower values corresponding to milder change in the landscape. Severity of landscape shift parameter is commonly used in dynamic optimization algorithms studies to analyze the robustness of the methods.

Number of peaks: an increase in the number of peaks causes an increment in the complexity to find the optimal peak. Therefore, the results tend to be worse with the growth of the number of peaks.

Dimension of the problem: many real-world problems have a high number of dimensions, which provides a strong motivation to study this parameter. Despite the relevance to do so, few dynamic optimization algorithms were analyzed on it.

Frequency of the change: this parameter brings enough of the controversy because some optimization experts argument that in many real-world applications the system may notify when the landscape itself has changed. However, even in this conditions high frequency of change brings extra complexity to the problem as it reduces the time necessary to search for a peak.

Table 2
Default parameters used in FDS to benchmark across all the scenario 2 MPB configurations.

Parameter	Value	Short parameter description
γ	1e-2	Initial step size used in the GrAILS local search starting at a hypersphere center
γ_{min}	1e-3	Step size value for stopping criterion used in the GrAILS local search starting at a hypersphere center
γ^{tr}	1e-4	Initial step size used in the GrAILS local search for trackers
γ^{tr}_{min}	1e-5	Step size value for stopping criterion used in the GrAILS local search for trackers
$\gamma_{decrease}$	1e-2	Step size decrease factor

Table 3
Parameter sensitivity on γ and γ_{min}. Offline error (standard error) metric is used for the analysis.

FDS		γ				
		1e-1	5e-2	1e-2	5e-3	1e-3
γ_{min}	1e-2	0.41 (0.05)	0.40 (0.05)	-	-	-
	1e-3	0.39 (0.05)	0.36 (0.04)	**0.33 (0.04)**	0.36 (0.04)	-
	1e-4	0.44 (0.05)	0.41 (0.05)	0.48 (0.05)	0.64 (0.07)	1.05 (0.11)
	1e-5	0.49 (0.06)	0.50 (0.06)	0.81 (0.08)	0.89 (0.08)	1.67 (0.13)

Table 4
Parameter sensitivity on γ^{tr} and γ^{tr}_{min}. Offline error metric is used for the parameter sensitivity study.

FDS		γ^{tr}										
		1e-1	5e-2	1e-2	5e-3	1e-3	5e-4	1e-4	5e-5	1e-5	5e-6	1e-6
γ^{tr}_{min}	1e-5	2.27 (0.07)	2.40 (0.07)	2.57 (0.07)	2.59 (0.07)	0.60 (0.04)	0.45 (0.05)	0.38 (0.05)	0.37 (0.04)	-	-	-
	1e-6	1.35 (0.05)	9.36 (0.22)	13.53 (0.24)	2.58 (0.07)	0.58 (0.04)	0.44 (0.04)	**0.36 (0.04)**	0.39 (0.4)	0.80 (0.05)	12.55 (0.34)	-
	1e-7	6.92 (0.17)	9.54 (0.23)	13.51 (0.24)	10.07 (0.22)	10.21 (0.22)	0.48 (0.04)	0.39 (0.04)	**0.36 (0.04)**	0.81 (0.05)	12.41 (0.33)	26.64 (0.62)
	1e-8	10.02 (0.22)	12.45 (0.24)	14.88 (0.31)	10.13 (0.23)	10.26 (0.22)	9.91 (0.23)	10.00 (0.22)	0.38 (0.04)	0.81 (0.05)	12.71 (0.34)	26.92 (0.62)

5. Results and discussion

In this section, a detailed analysis of the proposed method is provided, which includes parameters sensitivity study (Section 5.1), ablation study (Section 5.2), and complexity analysis (Section 5.3). Further, we present the results of extensive comparison with competing methods in different MPB scenarios in Section 5.4.

5.1. FDS parameters sensitivity

Here, we study the parameters of FDS: the initial GrAILS step size γ and its stopping criterion γ_{min}; the initial step size used for trackers γ^{tr} and its stopping criterion γ^{tr}_{min}; the parameter $\gamma_{decrease}$.

The appropriate values of the parameters of FDS are in Table 2.

In Table 3 the offline error performance is represented along the variation of the γ and γ_{min} parameters. These parameters affect the GrAILS that starts at a hypersphere center. First, from this analysis it can be observed that with the change in the parameters, the obtained results remain quite competitive. If parameter values are chosen in the neighborhood of the default selected parameters, less sensitivity is observed. Overall, higher values of the parameters lead to good performance. This can be interpreted as a preference to cover the distance towards the peak as fast as possible, after GrAILS has been started, rather than move through the space with smaller and more precise steps. These results follow the MPB benchmark landscape where the peaks are located quite sparsely. Finally, the switch between hyperspheres occurs when γ attains γ_{min}, while $\gamma_{decrease}$ controls the frequency of this switch. This procedure allows having an efficient exploration strategy.

In Table 4 the offline error performance based on the γ^{tr} and γ^{tr}_{min} parameters is given. These parameters affect the GrAILS at the trackers level. The results show that there is small sensitivity to the parameters values, and within an augmentation or decrease factor of 10 applied to the default values, the offline error metric remains competitive. This shows the sensitivity of FDS *w.r.t.* this sets of parameters. The values selected in the trackers step are much smaller than for the GrAILS starting in the hypersphere centers. Indeed, when a shift in the landscape occurs, the algorithm seeks for the optimum in a nearby region.

In Table 5 the offline error performance based on the variation of $\gamma_{decrease}$ parameter is presented. This parameter affects any GrAILS starting from a hypersphere center or from a tracked point. Proposed method shows low sensitivity to this parameter variation. Once more, this indicates the robustness of FDS.

Table 5
Parameter sensitivity on $\gamma_{decrease}$. Offline error metric is used for the parameter sensitivity study.

FDS	$\gamma_{decrease}$								
	3e-1	2e-1	1e-1	9e-2	8e-2	5e-2	1e-2	5e-3	1e-3
Offline error (Standard Error)	0.34 (0.04)	0.39 (0.04)	0.38 (0.04)	**0.34 (0.04)**	0.37 (0.04)	0.37 (0.04)	0.41 (0.04)	0.36 (0.04)	0.37 (0.04)

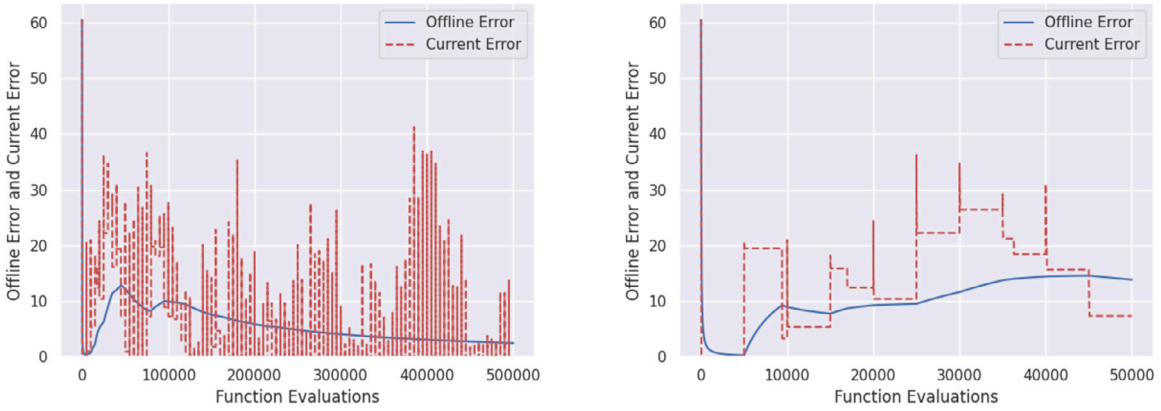


Fig. 5. (a) Single run using default parameters on MPB scenario 2. The plot illustrates the Current Error (dashed red line) and Offline Error (solid blue line). Every 5,000 FEs, the landscape changes, causing sharp spikes in the current error. Despite these shifts, the FDS algorithm efficiently adapts, consistently relocating the local/global optima. (b) Zoom-in on the first 50,000 FEs (compared to the full 500,000 in (a)). This detailed view highlights the primary challenge of MPB: during the initial phase, discovering all available peaks is difficult. However, once identified, the trackers ensure continuous detection and re-location of optimal solutions, leading to a gradual reduction in the offline error metric over time.

Fig. 5 presents an example of a convergence graph using the FDS method, illustrating the evolution of both Current and Offline errors over 500,000 function evaluations (FEs). The x-axis represents the number of function evaluations, while the y-axis denotes the error magnitude. The solid blue line indicates the offline error, showing a general decreasing trend, suggesting gradual convergence towards an optimal solution. The dashed red line represents the current error, exhibiting high variance with frequent spikes throughout the optimization process. This pattern suggests significant fluctuations in individual evaluations, likely due to an exploration-exploitation trade-off in the optimization algorithm. The decreasing offline error indicates overall progress, while the persistence of high current error values suggests ongoing search dynamics. The plot highlights the initial difficulty in identifying all peaks due to their sparse distribution across the landscape, as well as variations in their width and height.

In Fig. 6 a 2D analysis over the GrAILS exploration period is given for a particularly difficult random seed. This figure shows the performance of each GrAILS starting from a hypersphere center position, over the first period of the landscape (from 0 to 5000 evaluations). Each color represents a different attraction peak. Therefore, when employing the default MPB parameters, 10 colors are used in total. It can be observed that all the GrAILS executions are attracted consistently to the same peaks. This leads to a poor start in the next landscape.

Finally, in Fig. 7 an additional GrAILS intensification analysis is provided. These graphs start from a new random seed. Therefore, we observe a completely different scenario. We can identify 3 different parts. First, the initial GrAILS graphs (from subfigure line i = 0, column j = 0 to subfigure line i = 9, column j = 1) always reach a local optimum. Second, in subfigure line i = 9, column j = 3 the shift in the landscape has been detected in the middle of a GrAILS execution. Third, the tracking of the most promising areas (from subfigure line i = 9, column j = 4 to subfigure line i = 9, column j = 8) showing how FDS is able to recover back its local optimum after the shift. Finally, the last subfigure continues this chain of events over the next landscape.

5.2. Ablation study

In this subsection, we provide details on how FDS components influence the overall performance (see Table 6). These components were incrementally added to the base consisting of fractal decomposition and vanilla ILS (used for exploration and exploitation correspondingly) to achieve a steady increase in the performance. As described before, the main FDS components are the landscape shift detection, the tracking of best solutions, the GrAILS local search and the diversity control mechanism (implemented by selecting the furthest non-explored hypersphere *w.r.t.* the explored ones)

Overall GrAILS's speed of convergence depends on the number of steps until stopping criterium is reached. Experimental comparison with ILS which has a complexity of $O(D)$ shows that GrAILS requires $\approx$ 4.5 times less steps on MPB on average. In particular, for the search illustrated by Fig. 8 ILS required 264 evaluations, meanwhile GrAILS needed 59 evaluations.

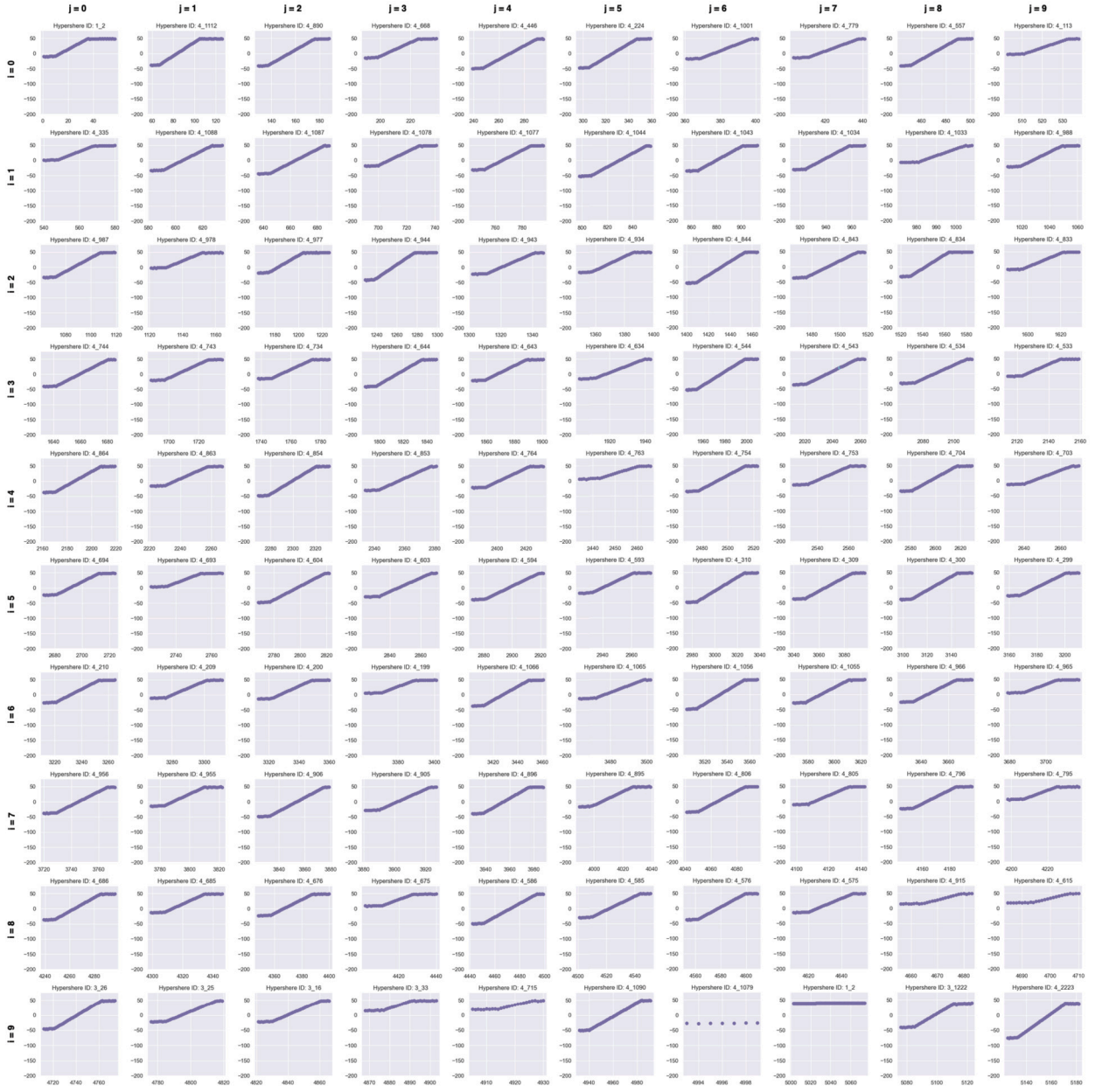


Fig. 6. GrAILS executions over the first landscape period (random seed 48), illustrating its three key phases. The intensification phase refines the search from multiple hypersphere centers, progressing from the initial subfigure (row i = 0, column j = 0) to (row i = 9, column j = 5). The landscape shift detection occurs mid-iteration, as seen in subfigure (row i = 9, column j = 6). Finally, the tracking and recovery phase ensures that local optima regain their performance, demonstrated in subfigure (row i = 9, column j = 7). This visualization highlights the robustness of GrAILS in adapting to dynamic landscapes.

5.3. FDS complexity analysis

Complexity analysis is an important step of studying optimization algorithm properties as it allows to predict the computational cost of a method when the problem parameters change, in particular, when the problem dimension D grows. In Table 7 the complexity of FDS operations with respect to D is given, for those of them that depend on this parameter. Specifically, a growth of problem dimension will affect *hypersphere selection* and *GrAILS*. Hypersphere selection implies comparison of a newly exploited hypersphere center to the unexploited ones. As centers of the hyperspheres correspond to the search tree of depth L with a branching factor $2xD$, the computation complexity of optimizing inside one hypersphere selection is $O(D^{L-1})$. We remark that parameter L is an FDS parameter. For MPB experiments we used $L = 4$. The number of hypersphere selection required will depend on problem characteristics, particularly on the landscape shift frequency. In the unlikely (computationally worst) case of all the hyperspheres centers being

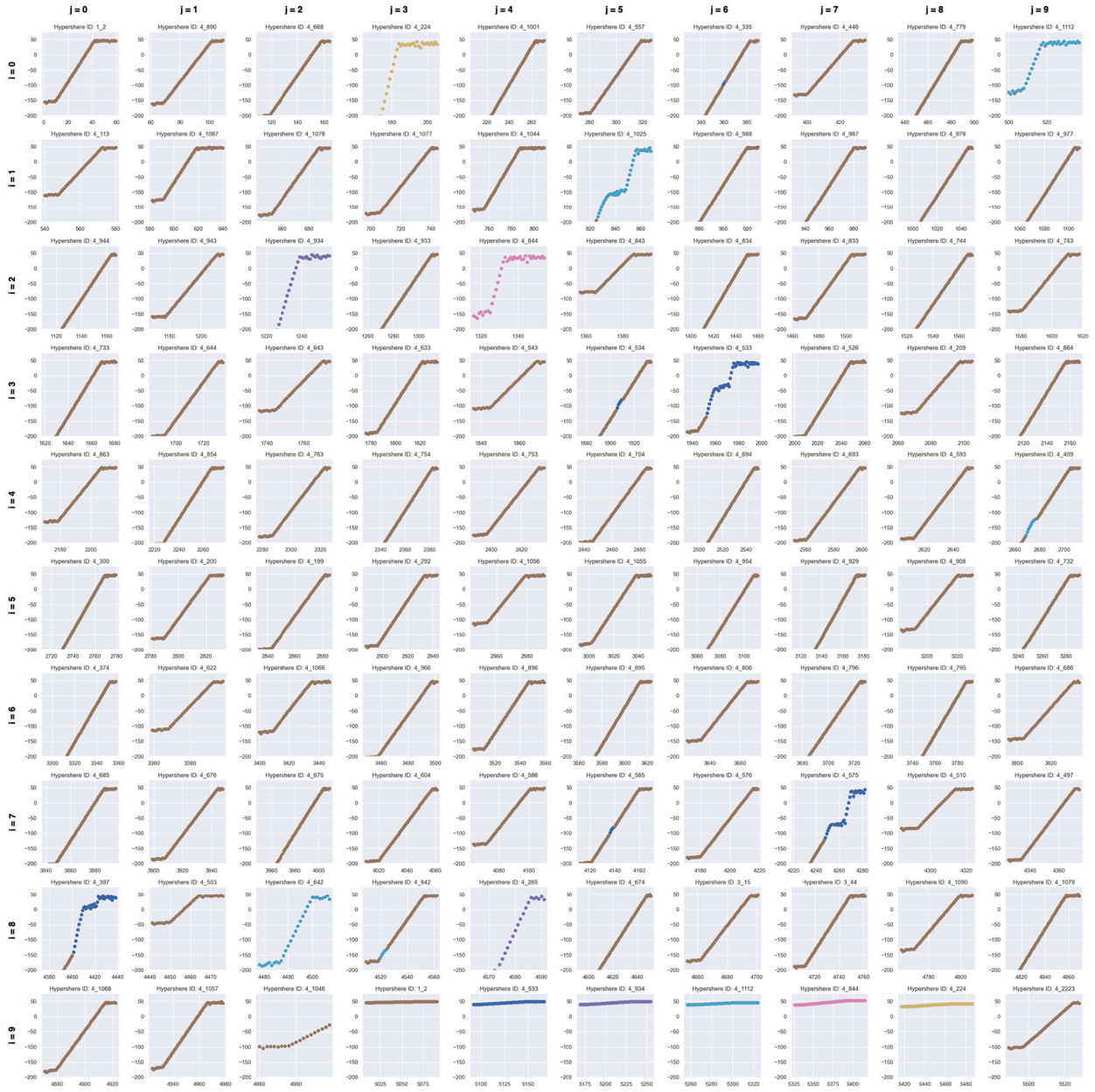


Fig. 7. GrAILS executions over the first landscape period (random seed 3), illustrating the three key phases of FDS GrAILS. The intensification phase refines the search from multiple hypersphere centers, progressing from subfigure (row i = 0, column j = 0) to (row i = 9, column j = 2). The landscape shift detection occurs mid-iteration, as seen in subfigure (row i = 9, column j = 3). Finally, the tracking and recovery phase ensures that local optima regain their performance, demonstrated from subfigure (row i = 9, column j = 4) to (row i = 9, column j = 8). This visualization highlights GrAILS' ability to adapt and recover in dynamic optimization landscapes.

explored theoretically the total complexity of all hyperspheres selections made together will reach $O(D^{2L-2})$. However, in practice the actual number of explored hyperspheres centers is expected to be much lower due to reasonable limitation on number of evaluations.

GrAILS assumes $2D$ steps in its ILS stage, and a calculation of Moore-Penrose inverse for $2D \times D$ matrix in the pseudo-gradient descent stage. The latter is commonly achieved via a Singular Value Decomposition (SVD) which has a complexity of $O((2D)^2 D) = O(D^3)$.

The experiments were performed on a machine equipped with a 2.3 GHz 8-core Intel Core i9 processor and 16 GB of 2667 MHz DDR4 RAM. Table 8 summarizes the runtime and memory consumption of the FDS method for different problem dimensions. The time per FE grows from 7.90×10^{-4} seconds at 5 dimensions to 1.29×10^{-3} seconds at 50 dimensions, indicating a nonlinear increase in computational cost. The memory usage per FE grows from 1.33×10^{-3} MB at 5 dimensions to and 1.75×10^{-3} MB at 50 dimensions. These results demonstrate how increasing the problem dimension affects both computational time and memory usage, emphasizing the trade-offs involved in scaling the optimization problem.

Table 6
Foundational blocks used to build FDS. Internal baseline is indicated in the first row. FDS is the result after the addition of each key-component. D is the dimension of the problem.

Foundational blocks	Offline Error
Fractal decomposition & vanilla ILS	11.89 (0.23)
Landscape shift detection	11.63 (0.23)
Trackers	4.25 (0.11)
GrAILS	1.61 (0.08)
GrAILS (ILS 1 step per D)	1.06 (0.87)
GrAILS & Line Search Projection	0.69 (0.06)
Rotation decomposition	0.37 (0.04)

Table 7
FDS complexity analysis. Here L denotes the maximum level of hyperspheres decomposition. In particular, for MPB experiments $L = 4$ was used.

Process description	Complexity
Hypersphere selection	$O(D^{L-1})$
GrAILS	$O(D^3)$

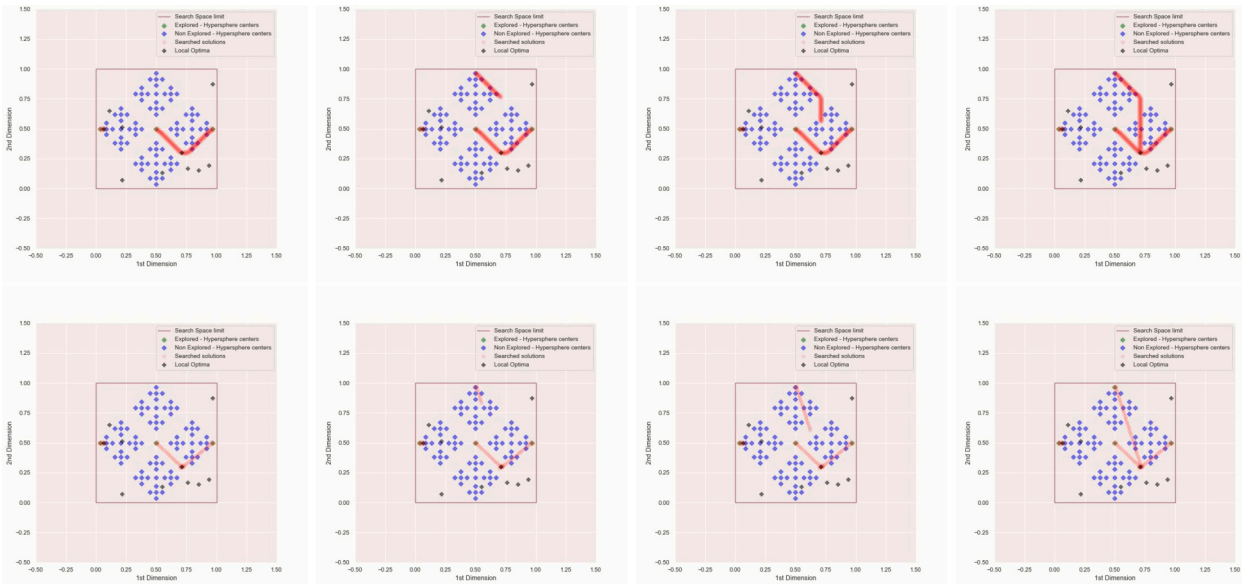


Fig. 8. 2D comparison between vanilla ILS (first row; 264 FEs) and GrAILS (second row; 59 FEs). The violet diamonds represent the centers of the hyperspheres (not inflated for clarity), while the black plus symbols denote the local optima. The red square outline marks the search space boundaries, and the red diamonds indicate the FEs performed. This comparison highlights the efficiency of GrAILS, achieving a significant reduction in FEs while maintaining effective search performance.

Table 8
Runtime and memory consumption of the FDS method for different problem dimensions. The table reports the time per FE (in seconds) and the memory consumption per FE (in MB).

Dimensions	5	10	20	50
time in seconds per FE (in seconds)	7.90×10^{-4}	6.12×10^{-4}	9.06×10^{-4}	1.29×10^{-3}
memory per FE (in MB)	1.33×10^{-3}	1.54×10^{-3}	1.34×10^{-3}	1.75×10^{-3}

5.4. Comparison to other competing methods

In this section, the results of FDS test on MPB benchmark are presented. In total, 23 benchmark configurations were considered. These configurations are obtained by varying different MPB's parameter (one at a time): Severity, Peaks, Dimension, and Frequency. As a recall, the default benchmark parameters were given in Table 1. To ensure that the results are statistically significant, 100 independent FDS runs were done for each of the 23 configurations, and a t-test at 5% significance level was performed on them. FDS parameters were the same for the different benchmark configurations.

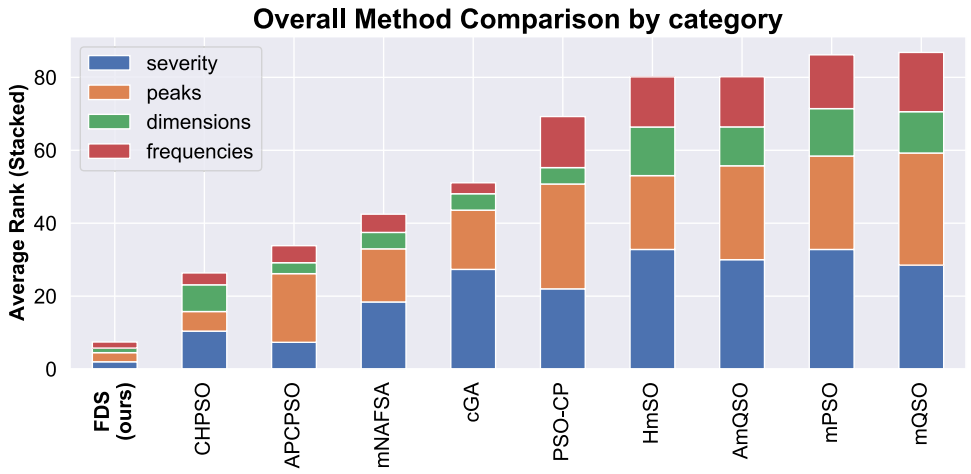


Fig. 9. Bar plot representing the sum of average ranks per category (lower values indicate better performance). The comparison includes methods that have scored at least once in each of the four categories: Severity, Peaks, Dimensions, and Frequencies. The results highlight that FDS achieves the best overall performance, excelling particularly in the Severity and Dimensions categories.

As described in Section 4, the evaluation metric used for MPB benchmark is the Offline Error (see Equation (9)). This metric is used per configuration, and thus to compare competing methods performance on a configuration set a metric aggregation is required.

Then, to evaluate the performance, we considered *the sum of average ranks*, where one average rank is calculated within a configuration group, and summed over four groups. Fig. 9 illustrates this aggregated metric with a stacked bar plot. Here only methods that reported results for at least one configuration in every group were considered. We observe that proposed method FDS achieves the best sum of average ranks.

More detailed results are presented in Table 9, Table 12, Table 10, and Table 11, that can be found in Appendix. We remark that FDS yields lower Offline Error than the competing methods in 19 configurations out of 23, including the default configuration of MPB scenario 2 (corresponding to the first column of the Table 9.

5.5. Limitations

All the experiments demonstrate the performance of FDS and highlights potential areas for refinement:

- FDS performs reliably at moderate to high change frequencies (1000–3000) and remains effective at lower (500) and higher (10,000) frequencies, though its adaptability could be further enhanced in extreme dynamic conditions. Even if in that case a restart from scratch of the optimization could be more suitable.
- FDS efficiently handles landscapes with a limited number of peaks (1 to 100) and continues to perform as the number of peaks increases (200), with a slight decrease in efficiency in highly complex multi-modal environments.

6. Conclusions

In this paper, a new algorithm, called FDS, is proposed to address hard continuous HDOPs. The proposed method is based on fractal decomposition of the variable decision space and uses a novel pseudo-gradient local search (GrAILS) for the exploitation.

FDS also uses both implicit and explicit archives, which are introduced to address landscape shifts. The implicit archive serves to maintain diversity, while the explicit archive is important for tracking optimal solutions across multiple time periods.

An extensive comparison of results is presented using the Moving Peaks benchmark. To illustrate the stability of the proposed method, multiple configurations of benchmark parameters were considered. The results obtained show the efficiency of the proposed method and its stability in different scenarios.

For future work, we plan to explore alternative decomposition patterns beyond hyperspheres, including adaptive decomposition strategies that dynamically adjust based on the characteristics of the problem. Furthermore, we are actively investigating the hybridization of FDS with other metaheuristics to enhance its search efficiency and robustness. Another promising direction is the integration of an adaptive step size within the GrAILS intensification technique, allowing FDS to adaptively adjust its settings based on problem-specific feedback.

Regarding the applications, FDS will be adapted to solve the Dynamic Traffic Flow Optimization with Variable Speed Limits problem and Dynamic Pricing for Parking Spaces.

CRediT authorship contribution statement

Arcadi Llanza: Writing – original draft, Visualization, Methodology, Investigation, Conceptualization. **Nadiya Shvai:** Writing – review & editing, Methodology. **Amir Nakib:** Writing – review & editing, Validation, Supervision, Project administration, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Detailed results of MPB experiments

Table 9
Offline error, standard error and ranking results in the MPB scenario 2. Per column, it includes the results for shift severities from 1 to 6 intensities. Per row, algorithms listed along the literature review are compared. Moreover, results are sorted from best to worst, top to bottom respectively at shift severity average rank. Finally, our method is highlighted in blue. FDS results have been obtained by running the algorithm over 100 independent runs. Symbol (*) means it is not statistically significant. (For interpretation of the references to color please refer to the web version of this article.)

Shift Severity	1.0	2.0	3.0	4.0	5.0	6.0	AVG RANK
FDS (2024)	**0.37 (0.04)**	**0.44 (0.04)**	**0.58 (0.04)**	**0.68 (0.04)**	**0.83 (0.05)**	**1.01 (0.05)**	1.00
DPCPSO [18]	0.77 (0.05)	0.71 (0.06)	0.94 (0.06)	1.11 (0.08)	1.07 (0.06)	1.09 (0.05)	3.00
APCPSO [17]	0.66 (0.04)	0.94 (0.07)	1.01 (0.07)	1.08 (0.05)	1.19 (0.07)	1.33 (0.09)	3.50
GRDE [8]	0.39 (0.07)	1.12 (0.07)	1.27 (0.08)	1.38 (0.08)	1.64 (0.10)	-	4.40
BfCS-wVN [26]	0.51 (0.11)	0.89 (0.16)	1.26 (0.13)	1.89 (0.18)	2.39 (0.20)	-	4.80
sslPSO [16]	0.75 (0.05)	1.18 (0.05)	1.32 (0.05)	1.59 (0.04)	1.59 (0.04)	1.92 (0.04)	5.33
AdaMPSO [30]	0.53 (0.11)	0.87 (0.16)	1.32 (0.20)	1.92 (0.18)	2.43 (0.19)	-	5.60
CCSA [29]	1.09 (0.03)	1.37 (0.04)	1.89 (0.05)	1.99 (0.04)	2.07 (0.04)	-	7.60
AdaMDE [30]	0.90 (0.30)	1.43 (0.29)	2.03 (0.25)	2.60 (0.27)	3.22 (0.30)	-	9.60
eFA-seq [38]	1.58 (0.05)	1.91 (0.03)	2.27 (0.06)	2.50 (0.06)	2.57 (0.05)	-	10.60
DynDE + LA [9]	1.32 (0.06)	-	2.23 (0.06)	-	3.33 (0.09)	-	11.00
DynDE + PI [9]	1.47 (0.08)	-	2.24 (0.08)	-	3.17 (0.10)	-	11.00
eFA-rw [38]	1.63 (0.05)	2.00 (0.06)	2.30 (0.06)	2.42 (0.07)	2.60 (0.06)	-	11.20
cGA [4]	1.15 (0.13)	2.19 (0.15)	3.31 (0.25)	5.35 (0.35)	6.45 (0.45)	8.15 (0.68)	11.33
$EFWA_{ce}$ [39]	1.63 (0.08)	2.11	2.61	-	-	-	13.33
Algorithm 4 [15]	1.72 (0.05)	-	-	-	-	-	16.00
Algorithm 3 [15]	1.81 (0.05)	-	-	-	-	-	17.00
CaAIS [32]	2.24 (0.02)	-	-	-	-	-	18.00
Algorithm 1 [15]	2.34 (0.06)	-	-	-	-	-	19.00
Algorithm 2 [15]	2.72 (0.06)	-	-	-	-	-	20.00

Table 10
Offline error, standard error and ranking results in the MPB scenario 2. Per column, it includes the results for the number of dimensions from 10 to 50. Per row, algorithms listed along the literature review are compared. Finally, our method is highlighted in blue. FDS results have been obtained by running the algorithm over 100 independent runs. Symbol (*) means it is not statistically significant.

Dimensions	10	20	50	AVG RANK
FDS (2024)	1.76 (0.13)	**2.72 (0.16)**	**3.40 (0.20)**	1.33
APCPSO [17]	**1.39 (0.06)**	4.33 (0.17)	-	2.50
AdaMPSO [30]	2.75 (0.41)	3.63 (0.46)	10.71 (1.26)	2.67
cGA [4]	2.45 (0.28)	4.25 (0.35)	-	3.00
AdaMDE [30]	3.03 (0.43)	4.63 (0.46)	19.48 (2.98)	4.33

Table 11
Offline error, standard error and ranking results in the MPB scenario 2. Per column, it includes the results for multiple frequencies from 500 to 10,000. Per row, algorithms listed along the literature review are compared. Finally, our method is highlighted in blue. FDS results have been obtained by running the algorithm over 100 independent runs. Symbol (*) means it is not statistically significant.

Frequency of change	500	1000	2000	2500	3000	10000	AVG RANK
FDS (2024)	**1.47 (0.06)**	**0.55 (0.06)**	**0.37 (0.05)**	**0.33 (0.04)**	**0.32 (0.04)**	0.41 (0.04)	1.50
GRDE [8]	-	-	-	-	0.99 (0.06)	**0.23 (0.05)**	1.50
cGA [4]	3.96 (0.21)	1.28 (0.13)	-	-	-	-	2.00
BfCS-wVN [26]	4.28 (0.42)	-	-	0.99 (0.24)	-	0.30 (0.06)	2.33
DynDE + LA [9]	-	2.83 (0.04)	1.96 (0.04)	-	1.65 (0.05)	-	2.67
DynDE + PI [9]	-	2.96 (0.05)	2.05 (0.05)	-	1.69 (0.06)	-	3.67
$EFWA_{ce}$ [39]	-	12.89 (0.41)	6.85 (0.27)	3.22 (0.20)	1.69 (0.06)	1.14 (0.09)	4.75
APCPSO [17]	-	-	5.72 (0.12)	-	2.12 (0.08)	0.32 (0.05)	5.00
DPCPSO [18]	-	-	4.50 (0.11)	-	2.01 (0.08)	0.43 (0.05)	5.00

Table 12
Offline error, standard error and ranking results in the MPB scenario 2. Per column, it includes the results for the number of peaks from 1 to 200. Per row, algorithms listed along the literature review are compared. Finally, own method is highlighted in blue. FDS results have been obtained by running the algorithm over 100 independent runs. Symbol (*) means it is not statistically significant.

# Peaks	1	5	20	30	40	50	100	200	AVG RANK
FDS (2024)	0.10 (0.02)	**0.25 (0.04)**	**0.42 (0.04)**	**0.46 (0.04)**	**0.49 (0.04)**	**0.45 (0.04)**	**0.50 (0.03)**	1.36 (0.04)	1.62
AdaMPSO [30]	-	-	1.20 (0.22)	0.66 (0.08)	-	0.99 (0.08)	1.03 (0.08)	**0.92 (0.08)**	2.80
BfCS-wVN [26]	0.30 (0.06)	0.38 (0.21)	0.74 (0.11)	1.00 (0.12)	0.97 (0.07)	0.84 (0.06)	1.11 (0.06)	1.18 (0.08)	3.00

(*continued on next page*)

Table 12 (*continued*)

# Peaks	1	5	20	30	40	50	100	200	AVG RANK
GRDE [8]	**0.00 (0.00)**	0.27 (0.08)	1.08 (0.09)	1.15 (0.06)	-	1.26 (0.08)	1.06 (0.10)	0.99 (0.12)	3.29
AdaMDE [30]	-	-	1.58 (0.20)	0.98 (0.12)	-	1.25 (0.11)	1.17 (0.09)	0.97 (0.06)	4.20
DPCPSO [18]	1.47 (0.49)	0.66 (0.07)	1.04 (0.05)	1.34 (0.05)	-	1.46 (0.05)	1.70 (0.05)	-	5.67
cGA [4]	0.92 (0.09)	1.06 (0.07)	1.18 (0.06)	1.35 (0.05)	1.53 (0.09)	1.65 (0.07)	1.80 (0.06)	1.71 (0.05)	6.25
APCPSO [17]	1.13 (0.03)	0.53 (0.06)	1.04 (0.05)	1.41 (0.04)	-	1.95 (0.05)	2.36 (0.05)	-	6.67
CCSA [29]	0.64 (0.01)	0.73 (0.04)	1.80 (0.03)	1.90 (0.03)	-	1.87 (0.02)	1.75 (0.02)	1.56 (0.02)	7.00
CaAIS [32]	2.24 (0.02)	2.28 (0.02)	2.51 (0.03)	2.63 (0.03)	2.28 (0.02)	2.32 (0.02)	1.67 (0.03)	2.64 (0.03)	9.00
eFA-rw [38]	0.64 (0.03)	1.21 (0.09)	2.16 (0.05)	2.50 (0.04)	-	2.70 (0.03)	3.03 (0.03)	3.14 (0.03)	9.29
eFA-seq [38]	0.64 (0.03)	1.06 (0.05)	2.22 (0.04)	2.64 (0.03)	-	2.91 (0.03)	3.27 (0.04)	3.44 (0.03)	10.00
DynDE + PI [9]	2.70 (0.11)	1.32 (0.09)	2.46 (0.08)	2.91 (0.11)	3.28 (0.09)	3.38 (0.10)	3.78 (0.11)	3.62 (0.09)	11.13
$EFWA_{ce}$ [39]	-	-	2.98	3.32	3.30	3.31	2.98	3.01	12.33
DynDE + LA [9]	3.07 (0.12)	1.41 (0.08)	2.60 (0.07)	3.05 (0.10)	3.34 (0.07)	3.56 (0.09)	3.88 (0.11)	3.71 (0.09)	12.38
Algorithm 2 [15]	5.61 (0.17)	2.69 (0.07)	3.09 (0.06)	3.32 (0.05)	3.35 (0.05)	3.29 (0.04)	-	2.92 (0.03)	13.43
Algorithm 4 [15]	4.62 (0.14)	1.74 (0.05)	2.77 (0.05)	3.25 (0.06)	3.50 (0.05)	3.66 (0.06)	-	3.91 (0.05)	13.57
Algorithm 3 [15]	3.74 (0.11)	1.80 (0.06)	2.82 (0.06)	3.27 (0.06)	3.53 (0.06)	3.57 (0.06)	-	3.87 (0.05)	13.71
Algorithm 1 [15]	5.11 (0.16)	2.35 (0.07)	3.19 (0.06)	3.49 (0.06)	3.73 (0.06)	3.71 (0.06)	-	3.62 (0.04)	15.29

Data availability

No data was used for the research described in the article.

References

[1] D. Yazdani, R. Cheng, D. Yazdani, J. Branke, Y. Jin, X. Yao, A survey of evolutionary continuous dynamic optimization over two decades—part a, IEEE Trans. Evol. Comput. 25 (2021) 609–629.

[2] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, in: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), vol. 3, IEEE, 1999, pp. 1875–1882.

[3] D. Yazdani, R. Cheng, D. Yazdani, J. Branke, Y. Jin, X. Yao, A survey of evolutionary continuous dynamic optimization over two decades—part b, IEEE Trans. Evol. Comput. 25 (2021) 630–650.

[4] M. Mohammadpour, H. Parvin, M. Sina, Chaotic genetic algorithm based on explicit memory with a new strategy for updating and retrieval of memory in dynamic environments, J. AI Data Min. 6 (2018) 191–205.

[5] B. Etaati, Z. Ghorrati, M.M. Ebadzadeh, A full-featured cooperative coevolutionary memory-based artificial immune system for dynamic optimization, Appl. Soft Comput. 117 (2022) 108389.

[6] R. Mendes, A.S. Mohais, Dynde: a differential evolution for dynamic optimization problems, in: 2005 IEEE Congress on Evolutionary Computation, vol. 3, IEEE, 2005, pp. 2808–2815.

[7] J. Brest, A. Zamuda, B. Boskovic, M.S. Maucec, V. Zumer, Dynamic optimization using self-adaptive differential evolution, in: 2009 IEEE Congress on Evolutionary Computation, IEEE, 2009, pp. 415–422.

[8] Z. Zhu, L. Chen, C. Yuan, C. Xia, Global replacement-based differential evolution with neighbor-based memory for dynamic optimization, Appl. Intell. 48 (2018) 3280–3294.

[9] J.K. Kordestani, A.E. Ranginkaman, M.R. Meybodi, P. Novoa-Hernández, A novel framework for improving multi-population algorithms for dynamic optimization problems: a scheduling approach, Swarm Evol. Comput. 44 (2019) 788–805.

[10] D. Parrott, X. Li, A particle swarm model for tracking multiple peaks in a dynamic environment using speciation, in: Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753), vol. 1, IEEE, 2004, pp. 98–103.

[11] S. Bird, X. Li, Using regression to improve local convergence, in: 2007 IEEE Congress on Evolutionary Computation, IEEE, Singapore, 2007, pp. 592–599.

[12] W. Du, B. Li, Multi-strategy ensemble particle swarm optimization for dynamic optimization, Inf. Sci. 178 (2008) 3096–3109.

[13] P. Novoa, D.A. Pelta, C. Cruz, I.G. del Amo, Controlling particle trajectories in a multi-swarm approach for dynamic optimization problems, in: Methods and Models in Artificial and Natural Computation. A Homage to Professor Mira's Scientific Legacy: Third International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2009, Santiago de Compostela, Proceedings, Part I 3, Spain, June 22–26, 2009, Springer, 2009, pp. 285–294.
[14] C. Li, S. Yang, M. Yang, An adaptive multi-swarm optimizer for dynamic optimization problems, Evol. Comput. 22 (2014) 559–594.
[15] J. Kazemi Kordestani, M.R. Meybodi, A.M. Rahmani, A note on the exclusion operator in multi-swarm pso algorithms for dynamic environments, Connect. Sci. 32 (2020) 239–263.
[16] D. Shen, B. Qian, M. Wang, A species conservation-based particle swarm optimization with local search for dynamic optimization problems, Comput. Intell. Neurosci. (2020).
[17] Y. Liu, J. Liu, Y. Jin, F. Li, T. Zheng, An affinity propagation clustering based particle swarm optimizer for dynamic optimization, Knowl.-Based Syst. 195 (2020).
[18] F. Li, Q. Yue, Y. Liu, H. Ouyang, F. Gu, A fast density peak clustering based particle swarm optimizer for dynamic optimization, Expert Syst. Appl. 236 (2024) 121254.
[19] J. Dréo, P. Siarry, An ant colony algorithm aimed at dynamic continuous optimization, Appl. Math. Comput. 181 (2006) 457–467.
[20] W. Tfaili, P. Siarry, A new charged ant colony algorithm for continuous dynamic optimization, Appl. Math. Comput. 197 (2008) 604–613.
[21] P. Korosec, J. Silc, The differential ant-stigmergy algorithm applied to dynamic optimization problems, in: Congress on Evolutionary Computation, IEEE, Trondheim, Norway, 2009, pp. 407–414.
[22] I. Moser, T. Hendtlass, A simple and efficient multi-component algorithm for solving dynamic function optimisation problems, in: 2007 IEEE Congress on Evolutionary Computation, IEEE, Singapore, 2007, pp. 252–259.
[23] S. Boettcher, A. Percus, Extremal optimization: methods derived from co-evolution, in: gecco-99: Proceedings of the Genetic and Evolutionary Computation Conference, 1999.
[24] D. Pelta, C. Cruz, J.R. González, A study on diversity and cooperation in a multiagent strategy for dynamic optimization problems, Int. J. Intell. Syst. 24 (2009) 844–861.
[25] J. Lepagnot, A. Nakib, H. Oulhadj, P. Siarry, A multiple local search algorithm for continuous dynamic optimization, J. Heuristics 19 (2013) 35–76.
[26] J.K. Kordestani, H.A. Firouzjaee, M.R. Meybodi, An adaptive bi-flight cuckoo search with variable nests for continuous dynamic optimization problems, Appl. Intell. 48 (2018) 97–117.
[27] R.I. Lung, D. Dumitrescu, A collaborative model for tracking optima in dynamic environments, in: Congress on Evolutionary Computation, IEEE, Singapore, 2007, pp. 564–567.
[28] R.I. Lung, D. Dumitrescu, Evolutionary swarm cooperative optimization in dynamic environments, Nat. Comput. 9 (2010) 83–94.
[29] W. Zhang, W. Zhang, G.G. Yen, H. Jing, A cluster-based clonal selection algorithm for optimization in dynamic environment, Swarm Evol. Comput. 50 (2019) 100454.
[30] J. Qin, C. Huang, Y. Luo, Adaptive multi-swarm in dynamic environments, Swarm Evol. Comput. 63 (2021) 100870.
[31] S. Raghul, G. Jeyakumar, A hybrid multi-population reinitialization strategy to tackle dynamic optimization problems, IEEE Access (2023).
[32] A. Rezvanian, S.M. Vahidipour, A.M. Saghiri, Caais: cellular automata-based artificial immune system for dynamic environments, Algorithms 17 (2023) 18.
[33] A. Nakib, S. Ouchraa, N. Shvai, L. Souquet, E.-G. Talbi, Deterministic metaheuristic based on fractal decomposition for large-scale optimization, Appl. Soft Comput. 61 (2017) 468–485.
[34] E.H. Moore, On the reciprocal of the general algebraic matrix, Bull. Am. Math. Soc. 26 (1920) 394–395.
[35] M.C. Du Plessis, A.P. Engelbrecht, Using competitive population evaluation in a differential evolution algorithm for dynamic environments, Eur. J. Oper. Res. 218 (2012) 7–20.
[36] K.A. De Jong, An Analysis of the Behavior of a Class of Genetic Adaptive Systems, University of Michigan, Ann Arbor, 1975.
[37] J.J. Grefenstette, Evolvability in Dynamic Fitness Landscapes: A Genetic Algorithm Approach, Proceedings of the Congress on Evolutionary Computation, vol. 3, IEEE, Washington, DC, USA, 1999, pp. 2031–2038.
[38] F.B. Ozsoydan, A. Baykasoğlu, Quantum firefly swarms for multimodal dynamic optimization problems, Expert Syst. Appl. 115 (2019) 189–199.
[39] H. Pekdemir, H.R. Topcuoglu, Efficient fireworks algorithms for dynamic optimisation problems in continuous space, J. Exp. Theor. Artif. Intell. 36 (2024) 389–414.