A fractal-based decomposition framework for continuous optimization

Thomas Firmin, El-Ghazali Talbi

▶ **To cite this version:**

Thomas Firmin, El-Ghazali Talbi. A fractal-based decomposition framework for continuous optimization. 2022. hal-04474444v2

HAL Id: hal-04474444
https://hal.science/hal-04474444v2

Preprint submitted on 13 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A fractal-based decomposition framework for continuous optimization

Thomas Firmin[1*] and El-Ghazali Talbi[1]

[1*]CRIStAL UMR CNRS 9189, University of Lille, Cité Scientifique, Villeneuve-d'Ascq, F-59650, France.

*Corresponding author(s). E-mail(s): thomas.firmin@univ-lille.fr;
Contributing authors: el-ghazali.talbi@univ-lille.fr;

Abstract

In this paper, we propose a generic algorithmic framework defining a unified view of fractal decomposition algorithms for continuous optimization. Fractals allow building a hierarchical decomposition of the decision space by using a self-similar geometrical object. The proposed generic framework is made of five distinct and independent search components: fractal geometrical object, tree search, scoring, exploration and exploitation. The genericity of the framework allowed the instantiation of popular algorithms from the optimization, machine learning and computational intelligence communities. Moreover, new optimization algorithms can be designed using various strategies of the search components. This shows the modularity of the proposed algorithmic framework. The computational experiments emphasize the behaviors of fractal-based approaches in terms of scalability, robustness, and the balance between exploitation and exploration in the search space. The obtained results illustrate the workability and the significance of certain search components. They influence the performances of fractal-based decomposition algorithms on different problems categories, such as sparable or multi-modal functions.

Keywords: Continuous optimization, Metaheuristic, High-dimensional optimization, Decomposition, Fractal, Tree search

1 Introduction

Optimizing a non-linear, non-convex, derivative free, or a black-box function in a high dimensional continuous search space is a complex task. Commonly we consider a minimization problem for an objective function $f : \Lambda \subset \mathbb{R}^n \to \mathbb{R}$:

$$\hat{x} \in \underset{x \in \Lambda}{\operatorname{argmin}} f(x) \tag{1}$$

where $\hat{x}$ is the global optima, f the objective function, and Λ a compact set made of inequalities (e.g. infima and suprema of the search space).

In this paper and following framework, we do not consider additional information as first or secondary derivatives, unlike some other frameworks [1].

Over the last twenty years, various optimization algorithms from multiple communities have been introduced to address continuous optimization challenges. These algorithms can be classified within a certain level of abstraction according to the problems they tackle, their similarities and

differences. These classifications allow identifying common search components, helping in their design and in understanding their behaviors [2, 3].

Metaheuristics is a family of partial search heuristic algorithms for which convergence toward the global optimum is not always guaranteed [2–4]. Among it, population-based algorithms generally rely on evolutionary algorithms (e.g. differential evolution, evolution strategies) and swarm intelligence (e.g. particle swarm optimization) [5]. Local search strategies are based on a single solution improvement (e.g. gradient-based algorithms, simulated annealing). These algorithms are stochastic and require a high number of evaluations to be efficient.

Concerning optimization problems with expensive objective functions, *surrogate-based* optimization (e.g. Bayesian optimization) can be an alternative to reduce the required budget. A major drawback of surrogate-based strategies is that they can suffer from the curse of dimensionality, and so, they poorly scale on high dimensional optimization problems [2, 3, 6].

In this paper, we investigate a particular class of optimization algorithms, which could be classified as *divide-and-conquer* strategies based on a hierarchical decomposition of the search space.

This work has been inspired by two distinct families of decomposition-based optimization algorithms. The first one concerns heuristics based on fractal decomposition, such as FRACTOP [7] and FDA [8]. The second one is based on algorithms derived from Lipschitzian global optimization, such as SOO [9], DIRECT [10] and its various extensions (e.g. eDIRECT, BIRECT, HD-DIRECT) [11].

Previous similar mathematical frameworks detailed some generalizations of partition-based algorithms [1, 12, 13]. In this work, we aim to include improper partition with overlapping sets allowing to model a broader category of algorithms including heuristics such as FDA.

In our framework, a *fractal* is a generalized concept describing a high dimensional *geometrical object* structuring the search space. It is also a *subset* of an initial decision space or of another fractal, and a *node* of a tree data structure.

A fractal is a never-ending pattern that can be used to generate any-scale fractal trees [14]. The self-reproducing characteristic of fractals suggests a new way for decomposing large-scale search spaces with low time and space complexity. Indeed, a fractal bears an unlimited number of levels that can be generated using simple and efficient analytical procedures with constant O(1) complexity. More exactly, a fractal has certain properties which can be exploited to better structure the search process, including recursion, scalability and self-similarity. By considering at least these three fundamental properties, it is possible to explore a search space indefinitely (recursion) in a structured manner and on all scales (scalability and self-similarity). The search space is then decomposed in a cascade of fractals organized by levels. Starting from a first fractal supposed to cover the whole search space, each fractal at a given level gives rise recursively to smaller fractals.

A *fractal-based decomposition algorithm* is organized around five search components: fractal, tree search, scoring, exploration and exploitation. It partitions the search space via a given *fractal geometrical object*. This partition of the space is then modeled via a tree-like data structure. Therefore, a *tree search* component is used to provide a dynamic and hierarchical fractal decomposition of the search space. A *scoring* search component allows modeling the probability of a subspace to contain the global optimum, by giving a fitness value to fractals.

These five search components are translated into software bricks within a generic Python framework named *Zellij*[1]. It offers a unified programming paradigm for the instantiation of fractal-based decomposition algorithms such as FDA (Fractal Decomposition Algorithm), SOO (Simultaneous Optimistic Optimization) or DIRECT (DIviding RECTangles). Moreover, Zellij allows the design of new optimization algorithms by combining search components, and also the implementation of alternative components thanks to a high level of abstraction.

[1]The Zellij software is available under GitHub https://github.com/ThomasFirmin/zellij.

1.1 Contributions

The main contribution of this paper is to unify fractal-based decomposition approaches into a generic and flexible algorithmic framework. The modularity of the framework allows the instantiation of popular optimization algorithms from the global optimization, machine learning and computational intelligence communities. This flexibility also allows the design of new optimization algorithms by developing new strategies of the framework's elementary building blocks, here called *search components*. These components are translated into flexible software bricks, helping the implementation of fractal-based decomposition algorithms. The workability of our framework was demonstrated on the BBOB benchmark from the Comparing Continuous Optimizer (COCO) framework. It illustrates that some search components have a great impact on the behaviors of fractal-based algorithms, in terms of scalability in dimensions, performances to certain types of problems or even on the tradeoff between exploration and exploitation.

1.2 Outline

The paper is organized as follows. In section 2, state-of-the art decomposition-based algorithms and other similar frameworks are presented. In section 3, the *Zellij* generic algorithmic framework based on fractal decomposition is presented. The following sections (sections 4 to 7) detail successively the search components of the algorithmic optimization framework: fractal geometrical object, tree search, scoring, exploration, and exploitation. Section 8, instantiate some fractal-based algorithms within Zellij. These algorithms are then used in Section 9, which depicts the experimental setup to compare fractal-based decomposition algorithms. Section 10 analyzes the obtained computational results on the BBOB benchmark from the COCO framework. Finally, we summed up in section 11, the framework, future works and new research opportunities for tackling high dimensional optimization problem.

2 Background and related works

This section introduces some popular fractal-based optimization algorithms from the metaheuristics family and from Lipschitzian optimization. Various frameworks generalizing some of these algorithms are also presented.

2.1 FRACTOP, FDA and PolyFRAC: Fractal-based metaheuristics

FRACTOP is one of the first metaheuristic based on fractal decomposition [7]. It uses hypercubes to decompose the search space, a genetic algorithm to explore each fractal, and simulated annealing to exploit a promising fractal. The algorithm implements a fuzzy measure, called *belief*, as a scoring method to determine the fitness of a fractal. One major drawback of this algorithm is its poor scalability in high dimension, due to an exponential complexity (2^n) to build a n-dimensional partition of equal size hypercubes.

The FDA metaheuristic partly solves the *curse of dimensionality* problem of FRACTOP [8]. Instead of a hypercube-based decomposition, it uses hyperspheres. By using such fractals, the decomposition has a lower complexity, but at the cost of overlapping fractals due to an inflation ratio. This ratio partially reduces the lack of space coverage implied by hyperspheres decomposition. The exploration component, called *Promising Hypersphere Search* (PHS), computes three points: the center of the hypersphere and two opposite points equidistant to the center. To score an explored fractal, FDA uses the *distance-to-the-best* solution found so far. To exploit a promising fractal, FDA applies to the smallest subspaces a heuristic named *Intensive Local Search*(ILS). The ILS is similar to a coordinate descent algorithm and is not bounded to a hypersphere, allowing to reach some uncovered areas of the search space.

The polyFRAC algorithm is a modification of FDA [15]. Rather than using hyperspheres, polyFRAC takes advantage of H-polytope fractals. The algorithm computes an approximation of these fractals, since, finding the vertices (i.e. n-faces) of a H-polytope is a complex procedure. The

exploration, exploitation, and scoring strategies are similar to those of FDA.

2.2 DIRECT: Dividing Rectangles

The DIRECT algorithm is initially a modification of the Schubert's algorithm [16]. It assumes that the objective function is Lipschitz-continuous, with a positive constant K:

$$|f(x) - f(y)| \leq K\|x - y\|, \quad \forall x, y \in *$$

Thus, there are several advantages of such an assumption. Indeed, it allows to easily prove convergence towards a global optimum. Additionally, the algorithm is deterministic and few hyperparameters have to be set.

Before partitioning, DIRECT samples the center of all sub-hyperrectangles. Then, at each iteration, the algorithm uses a series of trisections on the longest sides, depending on previously sampled centers, to subdivide the search space into smaller hyperrectangles. DIRECT introduces the concept of *Potentially Optimal Hyperrectangle* (POH), which is a strategy that selects the most promising fractal for further partitioning [10]. POH can be seen as the computation of a Pareto front between the size of hyperrectangles and their fitness values, preventing a lack of exploration due to over-dividing small subspaces.

However, DIRECT has several drawbacks. There is a poor balance between exploration and exploitation, and the computation complexity is subject to the *curse of dimensionality*. Many extended versions of DIRECT have been proposed in the literature to counterbalance some of these drawbacks. Some of them can be considered within our search components:

- Partitioning/Fractal: BIRECT (bisection) [17], eDIRECT (Voronoï cell) [18], DISIMPL (simplices) [19].
- Selection/Tree search: Pareto-Lipschitzian optimization [20], DIRECT Restart [21].
- Exploitation: DIRMIN [22].

These modifications try to overcome DIRECT's lower performances in high dimensions, low convergence rate when trapped by local optima, or lack of a local optimizer.

2.3 DOO, SOO, NMSO: Optimistic Optimization

DOO and SOO claim to be a generalization of the DIRECT algorithm. These algorithms make a strong assumption on the existence of a semi-metric l. This assumption simplifies the Lipschitz-continuous property by assuming a *local smoothness* around the global optimum $\hat{x}$ [9] :

$$f(\hat{x}) - f(x) \leq l(\hat{x}, x), \quad \forall x \in \Lambda$$

DOO is used when l is known; otherwise, SOO is more adapted. The strength of these algorithms, is their low number of parameters and the proof of a convergence bound. Both algorithms are deterministic. At each iteration and at each level of the partition tree, the best fractal is selected according to the evaluation of a representative solution inside it (e.g. center). In SOO, the balance between exploration and exploitation relies on the tree search algorithm. It consists in selecting, within the tree and in descending order, the best fractal at each level only if no other fractals from previous levels are better. In addition, a stochastic version called Sto-SOO has been designed for noisy loss function, where each fractal has to be evaluated multiple times [23].

Another optimistic optimization algorithm, based on a SOO scheme and named Naive Multi-scale Search Optimization (NMSO) [24], performs well on black-box optimization with expensive functions and a low budget. It uses a trisection, and more generally, k-section, to partition the space into hyperrectangular subspaces. NMSO computes the center of each hyperrectangle as its representative point and a tree data-structure. NMSO, compared to SOO, tends to be more exploitive by favoring deep trees. However, compared to DIRECT and SOO, NMSO has more parameters; a total of four, impacting its sensitivity, the exploration-exploitation tradeoff and the partition size.

2.4 Frameworks for partition-based algorithms

One of the first frameworks proposed in 1987 by Horst and Tuy [13, 25], precursors of branch-and-bound [3], describes a common structure and a proof of convergence for different partitioning

algorithms such as the Shubert algorithm [16] or ones within the Pintér's class [26]. They proposed a *conceptual algorithm*, including a definition of a *partition* of a subset $S \subseteq \Lambda$ based on boundaries of S; ∂S. They proposed an indirect definition of a *refinement*, i.e. a nested partition, using two sets, $\mathscr{P}$ containing selected subsets to be refined, and $\mathscr{R}$ the remaining ones. This approach was later improved with the *Divide-the-Best Algorithm* (DBA) framework [1]. DBA proposes a higher level of abstraction by including parameters of subsets, additional information about evaluations (derivatives), and an improved abstraction of the selection strategy, allowing to refine multiple subsets at the same iteration. Later, in [12], a k-ary tree structure is proposed to model a hierarchical partition of the search space. The authors proposed the *Multi-Scale Optimization* (MSO) framework and an analysis of its convergence using the Hölder condition, boundedness and sphericity of subspaces. Conversely to previously described works, we cannot propose a theoretical convergence within our framework, as we cannot ensure, for all cases described in section 4, that the global optima $\hat{x}$ is always reachable. Modeling heuristics comes at the expense of ensuring a proof of global convergence.

Other more specific studies [27, 28], proposed to break-down DIRECT-based algorithms into a *partitioning strategy*, i.e. the partition and evaluation of a subset, and *selection scheme*, i.e. our tree search component. The authors proposed a different combination of three selection and partitioning strategies. The three investigated selection strategies are an improvement of the POH from DIRECT, an aggressive one, and a two-step-based Pareto selection. Concerning, the partition, the selection can be combined with bisections, 1-dimensional trisection (e.g. SOO) and n-dimensional trisections (e.g. DIRECT). The proposed algorithms can sample the center of the hyperrectangles, two diagonal points or two vertices. By combining, these components, they designed 12 different DIRECT-type algorithms and showed that proper combination of algorithmic components result in different behaviors and performances on specific problems and situations.

These, works and frameworks pave the way toward hyperheuristics and automated design of optimization algorithms based on the partition of the search space. Decomposing this family of algorithms into search components, might result in similar works about hyperheuristics applied to population-based metaheuristics [29].

3 Zellij: A fractal-based decomposition algorithmic framework

This section introduces the basic concepts and search components of the *Zellij* framework.

Definition 1 (Search space)**.** *Let us define a continuous search space Λ of dimension n as a bounded subset of a metric space:*

$$\Lambda = L \times U = \prod_{i=1}^{n} [l_i, u_i] \ , \tag{2}$$

with $L, U \subset \mathbb{R}^n$, the infima and suprema, such that $\forall i \in [1, ..., n]$, $l_i < u_i$ and $\forall x \in \Lambda$, $\forall i \in [1, ..., n]$, $l_i \leq x_i \leq u_i$.

For convenience, we consider a measure set (Λ, Σ, μ), where Σ is a σ-algebra on the power set of Λ and μ is a measure.

Definition 2 (Fractal)**.** *A fractal S is :*

1. *a subset of a search space : $S \subseteq \Lambda$*
2. *a non-empty set : $S \neq \emptyset$*
3. *mesurable : $S \in \Sigma$*
4. *a child of an ancestor fractal (parent) denoted $\mathcal{A}_S$.*
5. *a node of a rooted tree $\mathcal{T}$ structure : $S \in \mathcal{T}$*
6. *the root of $\mathcal{T}$ if $S = \Lambda$*
7. *characterized by a set of m properties computed using inheritance of $\mathcal{A}_S$: $P(S, \mathcal{A}_S) := \{p_1(S, p_1(\mathcal{A}_S, \dots)), \dots, p_m(S, p_m(\mathcal{A}_S, \dots))\}$.*

In our framework, a *fractal* S, is a self-similar geometrical object which does not depend on any information besides the nature of its ancestor. As a simple example, decomposing a hypercube (i.e. n-cube) into smaller hypercubes of equal size, relies solely on the boundaries of the parent. The inheritance from the parent $\mathcal{A}_S$ during the creation of S is the only acceptable transfer of information between fractals. This is modeled by the

properties $P(S, \mathcal{A}_S)$. However, a child S can only compute its properties by considering $\mathcal{A}_S$ and cannot append $P(\mathcal{A}_S, \mathcal{A}_{\mathcal{A}_S})$, so to prevent combinatorial explosion. The number of properties m is globally fixed for all fractals. The inheritance prevents communication overhead during parallelization of any fractal-based algorithm. This inheritance-only mechanism has some drawbacks, as we cannot model algorithms based on shared information between fractals. For example, the Multilevel Coordinate Search algorithm [30] cannot be modeled. Indeed, points are sampled after the creation of a fractal and at the borders of hyperrectangles, so they can belong to different fractals, involving transfer of information between already created fractals.

Five properties characterize a fractal partitioning: partition size, building complexity, coverage, overlap, and memory complexity (see Table 1 and Figure 5). Coverage and overlap are measures defined in Section 4 and bounded to *improper partitions*. These two properties are the consequences of modeling heuristic approaches. Indeed, unlike existing mathematical frameworks [1, 12, 13], some parts of fractals can be outside their parents, and can overlap. Two fractals with overlapping boundaries can be considered as a special case of negligible overlapping. Therefore, we do not define the boundaries of a fractal S, often written as a function of S: ∂S [13], $\delta(S)$ [1], $\beta(S)$ [12]...

The decomposition of the search space can be defined by a *k-ary rooted partition tree* [12]. The root represents the initial complete search space, Λ, and nodes correspond to the generated fractals. The design of a tree search algorithm is crucial to efficiently explore and exploit the fractals. By introducing pruning strategies, one can reduce the search space and tackle memory issues, by eliminating some fractals. Popular tree search algorithms are *Best First Search* (BFS) [31], *Beam Search* [32] or *Epsilon Greedy Search* [33].

In the design of an optimization algorithm, high performance requires a trade-off between the exploration of the search space and the exploitation of the acquired knowledge. One has to find the best strategy for this dilemma [2–4, 34]. Exploration (i.e. diversification) of the search space allows one to obtain new knowledge. Non-explored fractals must be visited to ensure that all fractals are evenly explored and that search is not confined to a reduced number of fractals. Exploitation (i.e. intensification) into a reduced region of the search space uses that knowledge (e.g. best found fractals) to improve it. The promising fractals are searched more thoroughly in the hope to find better solutions.

Sampling in high dimensional search spaces is a critical task. Moreover, one does not sample in the same way in a hypercube or in a hypersphere. In DIRECT [10] and SOO [9], only the centers of the hyperrectangles are used in sampling. In our framework, we consider both deterministic and stochastic sampling. We also consider unique or multiple points methods to sample inside a fractal. The sampling process is essential for both exploration and exploitation search components.

The *exploration* can be done in a passive way (e.g. Markov Chain Monte Carlo (MCMC) sampling, low discrepancy sequences [35]) or in an active way (e.g. metaheuristics [2, 3, 36], surrogate-based optimization [6]). The challenge here is to find efficient sampling algorithms for diverse and complex fractals such as polytopes.

When a fractal reaches the maximum depth of the partition tree, an *exploitation* algorithm can be applied to it. The exploitation phase has not to be constrained within a fractal. The only bounds will be ones from the initial search space so that the exploitation can move freely toward a local or global optimum. One can use local search strategies such as gradient-based algorithms or simulated annealing [2, 3, 36].

The role of the *scoring* search component is to assign a quality, a fitness value, for a given fractal, by using acquired knowledge following an exploration phase. It can be seen as the quality value obtained by an acquisition function in Bayesian optimization. Many scoring methods can be used, such as *minimum*, *mean*, *median*, *distance-to-the-best* [8, 15], or *belief* [7].

The *Zellij* workflow is described in Figure 1. One can identify the five search components, their interactions and their algorithmic behaviors. The two *For each* instructions correspond to `line 14` and `line 16` of Algorithm 1. Two tests are made

at each iteration, the stopping criterion corresponds to the `while` loop at `line 13`, and the maximum depth test of the tree $\mathcal{T}$ to the `line 17` in Algorithm 1.

Because each search component is independent of another, it allows instantiating various strategies for fractals, tree search, exploration, exploitation and scoring search components. One can for instance reproduce FRACTOP by using *Hypercubes*, *Best First Search*, *Belief*, a *Genetic Algorithm* (GA) for the exploration and a *Simulated annealing* (SA) for the exploitation. Some search components can be very basic. Indeed, in DIRECT and SOO, there is no explicit exploitation strategy. The exploration and scoring methods consist in computing the center of each fractal and taking its objective value as its fitness. Other versions of fractal-based algorithms implementing different search components are discussed in Section 8.

To sum up, three main properties characterize *Zellij*:

- **Generalization**: our goal is to build a framework which unifies and generalizes various popular fractal-based decomposition algorithms from different communities (e.g. global optimization, reinforcement learning, computational intelligence).
- **Modularity**: the framework must be as modular as possible, so one can easily develop new optimization algorithms using the fractal-based decomposition approach.
- **Massively parallel**: a transparent and efficient parallel implementation of the algorithms on various architectures (e.g. multicores, GPUs). By decomposing the search space, these algorithms create smaller subproblems. This is an interesting property for their parallelization within a distributed environment. Previous works have investigated such parallel approaches and have demonstrated their workability [37–40]. Thus, by defining a common framework, our long-term goal is to propose common parallelization solutions to decomposition-based algorithms for Peta- and Exa-scale distributed architectures.

In the following sections, we will further describe the five search components, and introduce some theoretical background to fractal-based decomposition algorithms.

4 Geometrical fractal object

The fractal search component within Zellij framework allows structuring high dimensional search spaces to better explore and exploit it. Several types of fractals can be used in the decomposition of the search space, such as hyperspheres, hypercubes or Voronoï cells (see Figure 5). This geometrical object has a great impact on the behaviors of fractal-based decomposition algorithms. We start by describing a usual *partition* of the search space Λ, for which some principles can be found in [1, 12, 13]. To model heuristic approaches such as FDA, we extend these previous frameworks to *improper partitions*.

Definition 3 (k-partition)**.** *A k-partition of a fractal $\alpha \subseteq \Lambda$, can be defined by the union of k non-empty and disjoint subspaces. These subsets can be obtained by a partition operator $F : \alpha, P(\alpha, \mathcal{A}_\alpha) \rightarrow \{S_i\}_{i\in[1,...k]}$, with $S_i \subset \alpha$, and:*

$$\alpha = \bigcup F(\alpha, P(\alpha, \mathcal{A}_\alpha)) = \bigcup_{i=1}^{k} S_i \ , \qquad (3)$$

where $P(\alpha, \mathcal{A}_\alpha)$ are properties of α inherited from its parent $\mathcal{A}_\alpha$.

Here $\forall i \in [1, .., k], S_i \neq \emptyset$, and $\bigcap_{i=1}^{k} S_i = \emptyset$. This definition does not consider overlapping subspaces, nor improper partitions; $\alpha \neq \bigcup_{i=1}^{k} S_i$ or $\bigcap_{i=1}^{k} S_i \neq \emptyset$. For concision, we now write $F(\alpha, P(\alpha, \mathcal{A}_\alpha))$ as $F(\alpha)$, properties of a fractal are always accessible by any functions taking a fractal as an argument.

Additionally, all five components of our framework can use additional information about a fractal α, such as a measure of the size of a fractal. For example, the σ function in DIRECT [41], with σ_2 or σ_∞, measures the size of hyperrectangles. These properties are handled by $P(\alpha, \mathcal{A}_\alpha)$.

Using a given fractal, one can build a recursive decomposition of Λ or a decomposition of a subset of Λ. Building a k-partition of an existing subset is called a *refinement*. Thus, when refining a fractal, the definition of S does not change. The

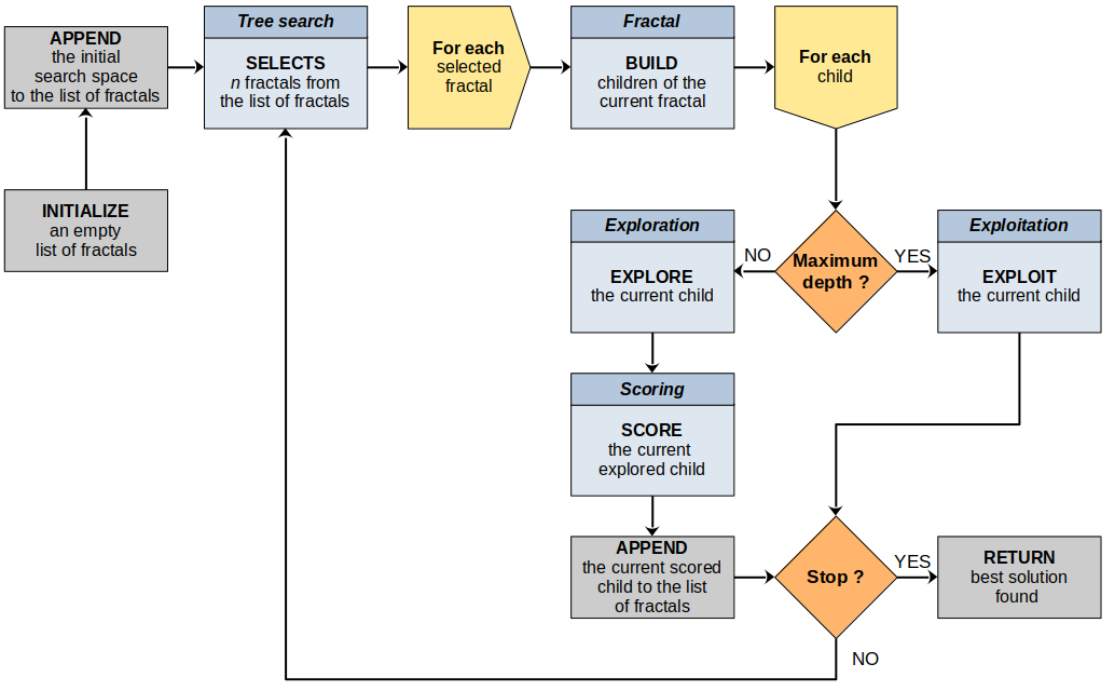


Fig. 1: The workflow of the Zellij framework. In blue, the five search components. In orange, the tests made at each iteration.

refinement is a self-similar object, it uses the same definition as S to build children of a fractal; all S_i are of the same nature.

Definition 4 (Hierarchical k-refinement)**.** *Let D be the number of successive refinements of a subset $S_{l,j,i} \subseteq \Lambda$. We write $j \in [1, ..., E_l]$, with E_l the number of sets that have been refined l times, and $i \in [1, ..., k]$ the ith set of a k-partition of a superset j. A hierarchical k-refinement of $S_{l,j,i}$ of maximum level D (i.e. depth) is written as:*

$$\begin{aligned} &\forall l \in [2, ..., D-1], \\ &\exists (x, y, j) \in ([1, ..., E_{l-1}], [1, ..., k], [1, ..., E_l]) : \\ &S_{l,x,y} = \bigcup F(S_{l,x,y}) \\ &\quad = \bigcup_{i=1}^{k} S_{l+1,j,i} \end{aligned} \tag{4}$$

Here, a subset identified by (l, j, i) corresponds to the fractal at level l, child number i of the fractal j at level $l-1$. An appropriate data structure used to model Definition 4 is a k-ary rooted tree. Hence, one can rewrite the initial search space Λ as the root of this tree: $S_{(1,0,1)} = \Lambda$, where $E_0 = \emptyset$. A fractal is now considered as a node of a k-ary rooted tree $\mathcal{T}$. Figure 2 shows an example of a 2-refinement of depth 4, of a 2D square using a bisection, drawn as a red dotted line, along the longest side.

For a fixed $l \in [1, ..., D-1]$, one can write the set $_l$ of supersets (ancestors) at level l that have children at level $l+1$. Let us denote a leaf at the level l of a tree $\mathcal{T}$ as $\circ[l, j, i]$. An interesting property of using a k-ary tree on a k-refinement that fully covers Λ is that the initial search space will be equal to the union of all the leaves.

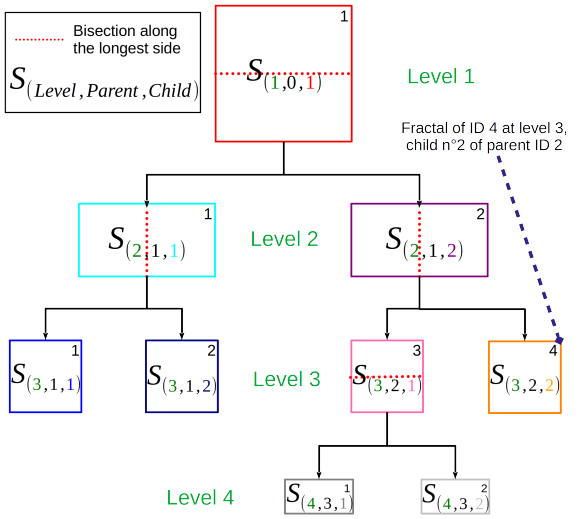


Fig. 2: Example of a 2-refinement of depth 4, with a bisection along the longest side.

Theorem 1. *Let Λ be a search space and the root of a k-ary partition tree $\mathcal{T}$ of maximum depth D. For all $1 < l \leq D$, the union of all leaves $\circ[l, j, i]$, children of nodes numbered by $[1, ..., E_{l-1}]$ as defined in Definition 4, is equal to Λ:*

$$\Lambda = S_{(1,0,1)} = \bigcup_{l=2}^{D} \bigcup_{j=1}^{E_{l-1}} \bigcup_{i=1}^{k} \circ[l, j, i] \tag{5}$$

Proof. We consider a k-ary rooted tree $\mathcal{T}^{(d)}$ of depth d, with $1 < d \leq D$. Considering $_l$ the set of fractals at level l having children at level $l+1$:

$\bigcup_{d-1} = \bigcup_{j=1}^{E_{d-1}} \bigcup_{i=1}^{k} \circ[d, j, i]$

If we remove all $\circ[d, j, i]$ from $\mathcal{T}^d$, we obtain a tree $\mathcal{T}^{(d-1)}$. So, the leaves of $\mathcal{T}^{(d-1)}$ at level $d-1$ can be written as $\{\circ[d-1, j, i]_{,d-1}\}$.

Thus, $\bigcup_{d-2} = \bigcup_{j=1}^{E_{d-2}} \bigcup_{i=1}^{k} \circ[d-1, j, i] \bigcup_{d-1}$

and so on, until $\mathcal{T}^{(1)}$:

$\Lambda =_1 = \bigcup_{i=1}^{k} \circ[2, 1, i] \bigcup_2$ □

For simplicity, one can write all leaves $\circ[l, j, i]$ of $\mathcal{T}^{(d)}$, a tree $\mathcal{T}$ of depth d, as $\mathcal{L}_{\mathcal{T}^{(d)}}$.

As mentioned before, Theorem 1 is valid for an exhaustive and mutually exclusive partition. This is not the case for some fractal-based algorithms such as FDA, which uses hyperspheres [8]. Indeed, fractals can overlap and do not necessarily cover Λ. Additionally, it is important to mention that, the higher the dimension of the problem, the smaller the covering of the hyperspheres. In FDA, the covering is increased according to a parameter, called the inflation ratio, no matter the overlap between fractals. However, the inflation ratio can make a part of the hypersphere going outside Λ. In our algorithmic framework, we consider the intersection between fractals and Λ. A fractal, or a part of a fractal, cannot be outside the initial search space Λ. If so, then the fractal is trimmed.

We previously described the basic principles of fully covering fractal-based decomposition algorithms. To extend our framework to improper partitions, we have to measure what the overlap and coverage are for fractal-based decomposition algorithms. By considering a tree $\mathcal{T}^{(d)}$ and (Λ, Σ) a measurable set, we can write $\mathcal{L}_{\mathcal{T}^{(d)}} \subseteq \Sigma$ and $\mathcal{L}_{\mathcal{T}^{(d)}} \in \Sigma$. We define a measure, $\mu : \Sigma \to [0, +\infty]$. A stricter condition is applied to μ, $\mu(A) = 0 \iff A = \emptyset$. Hence, the result of a k-partition cannot be made of null sets. Because F cannot produce null sets, and because D, the maximum tree depth, is finite, we can define a measure of the coverage of Λ by $\mathcal{L}_{\mathcal{T}^{(d)}}$ and their overlap.

Definition 5 (Coverage)**.** *Let (Λ, Σ, μ) a measure space, and $A, B \subseteq \Lambda$, $A, B \in \Sigma$ and $B \subseteq A$. A measure of the coverage of A by B can be written as:*

$$C(A, B) = \mu\left(\complement_A B\right) \tag{6}$$

However if $B \nsubseteq A$ such as in FDA:

$$C(A, B) = \mu\left(A \setminus \left(A \bigcap B\right)\right) \tag{7}$$

From Equation 7 we can infer $B \setminus A$, the part of B outside A.

One can distinguish two types of coverage: $C(\Lambda, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}})$ measures the coverage of the initial fractal by $\mathcal{L}_{\mathcal{T}^{(d)}}$, and $C(S_{l,j,i}, \bigcup F(S_{l,j,i}))$ measures the coverage of a fractal by its children.

Definition 6 (Overlap)**.** *Let (Λ, Σ, μ) a measure space, and $A \neq \emptyset$ a countable set such that $\forall n \in \mathbb{N}, \forall a_n \in A : a_n \in \Sigma$ and $\bigcup_{n \in \mathbb{N}} a_n \in \Sigma$. A measure of the overlap between all a_n can be written*

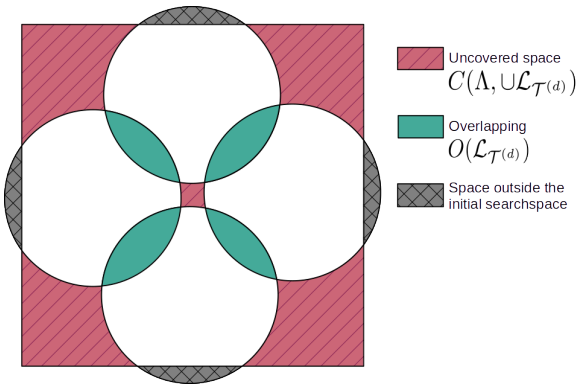


Fig. 3: 2-dimensional illustration of covering and overlap applied on a FDA scheme.

as:

$$O(A) = \mu\left(\bigcup_{i=1}^{|A|-1}\bigcup_{j=i+1}^{|A|}\left(a_i \bigcap a_j\right)\right) \tag{8}$$

Coverage and overlap properties are illustrated in Figure 3 as a 2-dimensional FDA scheme. One can see, in hatched-magenta, the uncovered space and in solid-green the overlap between fractals. Moreover, because of the inflation of hyperspheres, the gridded-gray space represents parts of the fractals outside Λ.

As the monotonic property of measures is not strict, $A \subseteq B \implies \mu(A) \leq \mu(B)$, one cannot say that $\mathcal{L}_{\mathcal{T}^{(d)}}$ covers Λ by only looking at $C(\Lambda, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}})$. Thus, several assumptions are made. For continuous dimensions, we consider that $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}$ covers Λ if $C(\Lambda, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) = 0$, even if $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}$ does not include Λ boundaries. To describe necessary assumptions of an improper partition, we have to redefine the *partition operator* F described in Definition 2.

Definition 7 (Partition operator)**.** *Considering a search space Λ and its measure space (Λ, Σ, μ). A partition operator F is a function, $F : \alpha,\ P(\alpha, \mathcal{A}_\alpha) \to \{S_i\}_{i\in[1,\dots k]}$, describing how to create children S_i of a given fractal $\alpha \subseteq \Lambda$ such that :*

1. *Children cannot be empty nor null sets:*

$$\forall i \in [1, \dots, k]\,,\ (S_i \neq \emptyset) \wedge (\mu(S_i) > 0)$$

2. *The union of the children cannot be empty, and its measure is significant:*

$$\left(\bigcup S_i \neq \emptyset\right) \wedge \left(\mu\left(\bigcup S_i\right) > 0\right)$$

3. *The intersection between children must be part of the search space or can be empty:*

$$\left(\bigcap S_i \subseteq \Lambda\right) \vee \left(\bigcap S_i = \emptyset\right)$$

4. *Two children are strictly different subsets, and a child cannot be a subset of another:*

$$\forall i, j \in [1, \dots, k]\,,\ i \neq j, (S_i \neq S_j) \wedge (S_i \not\subseteq S_j)$$

5. *A subset of a child must be part of its ancestor and the measure of their intersection must be significant:*

$$\forall i \in [1, \dots, k]\,,$$
$$\left(S_i \bigcap \alpha \neq \emptyset\right) \wedge \left(\mu\left(S_i \bigcap \alpha\right) > 0\right)$$

6. *A child must be significantly smaller than its ancestor:*

$$\forall i \in [1, \dots, k]\,,\ \mu(S_i) < \mu(\alpha)$$

Even if, Definition 7 allows modeling many different *improper partitions*, open questions remain about stricter conditions that could be applied to the definition of the *partition operator*. Notably, about enforcing the part of a child outside its parent to be significantly smaller than the part inside it; $\mu\left(S_i \bigcap \alpha\right) \gg \mu\left(S_i \setminus \alpha\right)$. Which could also be applied to all children; $\mu\left(\bigcup S_i \bigcap \alpha\right) \gg \mu\left(\bigcup S_i \setminus \alpha\right)$.

By using Definitions 2, 5, 6 and 7, we can now modify Theorem 1 for an improper k-partition.

Theorem 2. *Let Λ be a search space, $\Sigma = \mathcal{A}(\Lambda)$ a σ-algebra on Λ, and (Λ, Σ, μ) a measure space. Λ is the root of a k-ary partition tree $\mathcal{T}^{(d)}$ of depth d and $\mathcal{L}_{\mathcal{T}^{(d)}}$ its leaves. $C(.,.)$ and $O(.,.)$ are measures of the coverage and overlapping between fractals. We suppose that $\mathcal{L}_{\mathcal{T}^{(d)}}$ is not a null set and $\mu(A) = 0 \iff (A = \emptyset) \vee (\bigcup A = \emptyset)$. Then, a given decomposition-based algorithm is said to be:*

1. ***Preservative**: If the union of the leaves covers the search space;*
$\forall d > 1,\ C(\Lambda, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) = 0$ *and:*

   (a) *Proper: If the fractals never significantly overlap;*
   $O(\mathcal{L}_{\mathcal{T}^{(d)}}) = 0$

   (b) *Improper: If the fractals significantly overlap;*
   $O(\mathcal{L}_{\mathcal{T}^{(d)}}) > 0$

   Then,
$$\mu(\Lambda) = \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) \tag{9}$$

2. ***Sacrificial**: If the union of the leaves does not cover the search space;*
$\forall d > 1,\ C(\Lambda, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > 0$ *and:*

   (a) *Lowly: If children do not cover their ancestor for the first decomposition of the search space;*
   $\forall l > 1,\ C(S_{l,i,j}, \bigcup F(S_{l,i,j})) = 0$ *and* $C(\Lambda, \bigcup F(\Lambda)) > 0$. *Then,*
$$\mu(\Lambda) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) \tag{10}$$
   with $\forall d > 1,\ \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) \leq \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}\right)$

   (b) *Highly: If after every refinement children do not cover their ancestor;*
   $\forall l > 1,\ C(S_{l,i,j}, \bigcup F(S_{l,i,j})) > 0$ *and* $\bigcup F(S_{l,i,j}) \setminus S_{l,i,j} = \emptyset$. *Then,* $\forall d > 1$,
$$\mu(\Lambda) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}\right) \tag{11}$$

   (c) *Unbounded: If after every refinement a part of the children are outside their ancestor;*
   $\forall l > 1,\ C(S_{l,i,j}, \bigcup F(S_{l,i,j})) > 0$ *and* $\bigcup F(S_{l,i,j}) \setminus S_{l,i,j} \neq \emptyset$. *Then, we cannot infer any relations between measures of the leaves:*
$$\mu(\Lambda) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) \lesseqgtr \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}\right) \tag{12}$$

Proof. We consider a k-ary rooted tree $\mathcal{T}$ of depth d, with $1 < d \leq D$. This tree is denoted as $\mathcal{T}^{(d)}$, and the leaves of $\mathcal{T}^{(d)}$ as $\mathcal{L}_{\mathcal{T}^{(d)}}$. One can consider a partitioning function F defined by Equation 3.

Preservative:
If $\forall d > 1,\ C(\Lambda, \cup\mathcal{L}_{\mathcal{T}^{(d)}}) = 0$.
Then $\Lambda = \cup\mathcal{L}_{\mathcal{T}^{(d)}}$, because by construction $\cup\mathcal{L}_{\mathcal{T}^{(d)}} \subseteq \Lambda$ and

$$\begin{aligned} C(A, B) = 0 &\iff \mu\left(A \setminus \left(A \bigcap B\right)\right) = 0 \\ &\iff A = B \end{aligned}$$

So, we have:

$$\Lambda = \bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \implies \mu(\Lambda) = \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right)$$

Lowly sacrificial:
If $\forall d > 1,\ C(\Lambda, \cup\mathcal{L}_{\mathcal{T}^{(d)}}) > 0$ and $C(S_{d,j,i}, \cup F(S_{d,j,i})) = 0$. Then, $\Lambda \supset \cup\mathcal{L}_{\mathcal{T}^{(d)}} \implies \mu(\Lambda) > \mu\left(\cup\mathcal{L}_{\mathcal{T}^{(d)}}\right)$
and $\cup\mathcal{L}_{\mathcal{T}^{(d)}} \subseteq \cup\mathcal{L}_{\mathcal{T}^{(d+1)}}$.
So we have,

$$\begin{aligned} &\forall a \in \mathcal{L}_{\mathcal{T}^{(d)}},\ F(a) \in \mathcal{L}_{\mathcal{T}^{(d+1)}} \\ &\quad \implies \mu(a) \leq \mu\left(\bigcup F(a)\right) \\ &\quad \implies \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) \leq \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}\right) \end{aligned}$$

Highly sacrificial:
If $\forall d > 1,\ C(\Lambda, \cup\mathcal{L}_{\mathcal{T}^{(d)}}) > 0,\ C(S_{d,j,i}, F(S_{d,j,i})) > 0$ and $\cup F(A) \setminus A = \emptyset$ (Subsets cannot have solutions outside their parents).
Then, $\Lambda \supset \cup\mathcal{L}_{\mathcal{T}^{(d)}} \implies \mu(\Lambda) > \mu\left(\cup\mathcal{L}_{\mathcal{T}^{(d)}}\right)$
and $\cup\mathcal{L}_{\mathcal{T}^{(d)}} \supset \cup\mathcal{L}_{\mathcal{T}^{(d+1)}}$.
So we have,

$$\begin{aligned} &\forall a \in \mathcal{L}_{\mathcal{T}^{(d)}},\ F(a) \in \mathcal{L}_{\mathcal{T}^{(d+1)}} \\ &\quad \implies \mu(a) > \mu\left(\bigcup F(a)\right) \\ &\quad \implies \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}\right) \end{aligned}$$

Unbounded sacrificial:
If $\forall d > 1,\ C(\Lambda, \cup\mathcal{L}_{\mathcal{T}^{(d)}}) > 0,\ C(S_{d,j,i}, F(S_{d,j,i})) > 0$ and $\cup F(A) \setminus A \neq \emptyset$.
Then, $\Lambda \supset \cup\mathcal{L}_{\mathcal{T}^{(d)}} \implies \mu(\Lambda) > \mu\left(\cup\mathcal{L}_{\mathcal{T}^{(d)}}\right)$
and $\cup\mathcal{L}_{\mathcal{T}^{(d)}} \supset \cup\mathcal{L}_{\mathcal{T}^{(d+1)}}$. If $\mu\left(\cup F(A) \setminus A\right) = 0$, then the part of $F(A)$ outside A is negligible, so

behaviors are similar to highly sacrificial:

$$\begin{aligned}
&\forall a \in \mathcal{L}_{\mathcal{T}^{(d)}},\ F(a) \in \mathcal{L}_{\mathcal{T}^{(d+1)}} \\
&\quad \implies \mu(a) > \mu\left(F(a) \bigcap a\right) + \mu\left(\bigcup F(A) \setminus A\right) \\
&\quad \implies \mu(a) > \mu\left(\bigcup F(a)\right) \\
&\quad \implies \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}\right)
\end{aligned}$$

Otherwise, if $\mu\left(\cup F(A) \setminus A\right) > 0$, then we need additional information or constraints to determine the inequalities. □

This theorem describes behaviors of an improper, k-refinement where leaves might overlap or not fully cover the initial search space. Preservative algorithms describe a partition that covers Λ no matter whether there is overlap between leaves or not, as we consider $\cup\mathcal{L}_{\mathcal{T}^{(d)}}$. The proper preservative situation, is comparable to previous frameworks [1, 12, 13], where two fractals, A and B, are disjoint if and only if: $A \cap B = \beta(A) \cap \beta(B)$, where β are the boundaries of a fractal. So $A \cap B \neq \emptyset$ but the overlap is negligible, $\mu(\beta(A) \cap \beta(B)) = 0$.

However, low-sacrifice based algorithms describe leaves of the tree that do not fully cover Λ, but where children of the fractal cover its parent. In this case, one accepts to lose a part of the search space at the first refinement of Λ. Then future refinements should not sacrifice solutions anymore.

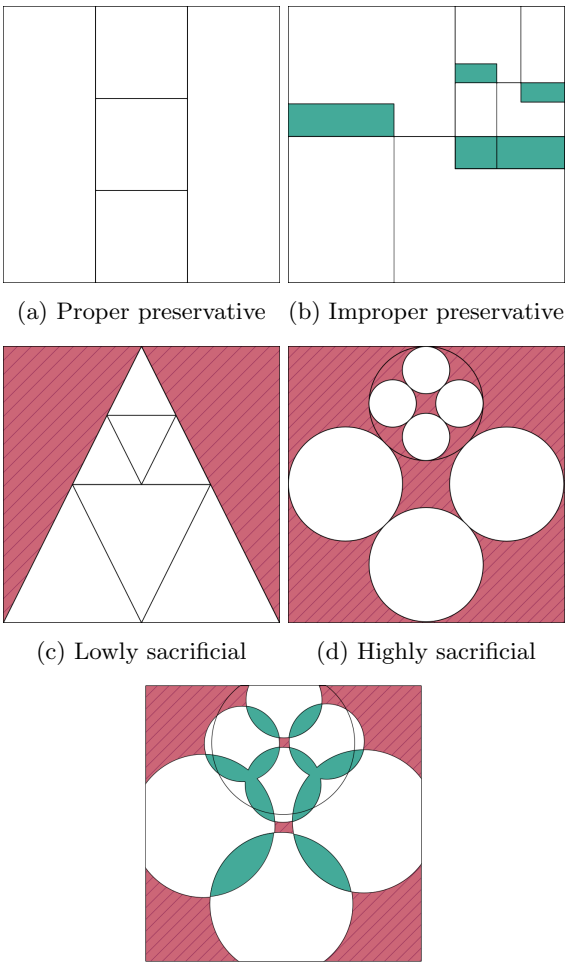
For high-sacrifice based algorithms, one only considers fractals that cannot expand outside their parents. If so, it becomes hard to define the notion of sacrifice, as we can imagine a decomposition where successive children do not fully cover their parents and shift outside an ancestor fractal. This can be the case in unbounded situations, where fractals can cover solutions outside the parent space. One should be aware of these properties, and determine if it is acceptable to lose some solutions to carefully choose the right decomposition function F.

According to Theorem 2, let us consider an initial search space Λ defined by a hypercube. In FDA [8], the algorithm considers the inscribed hypersphere as the initial search space. The algorithm accepts to "sacrifice" points between the hypercube and the inscribed hypersphere to perform an improper k-refinement using hyperspheres. To overcome this, the exploitation phase of FDA can ignore fractals' borders, so it can reach uncovered solutions.

These different situations are described in Figure 4. Proper preservative in Figure 4a based on SOO, preserve the initial search space Λ, no space is lost. Improper preservative algorithms in Figure 4b illustrate a case where overlapping fractal fully cover Λ. Low sacrifice fractals, on Figure 4c, are here based on Sierpinski triangles. Such fractals sacrifice space of Λ to use a geometry based on triangles. And finally, the examples on Figure 4d and 4e, based on an FDA scheme, sacrifice space after each refinement. The unbounded sacrificial example shows how difficult it can be to describe overlapping and coverage when children can expand outside their ancestors.

The main challenge when selecting a type of fractal is its scalability in dimension. Scalability can be measured by three properties defined in Table 1: the partition size k, the building complexity $\mathcal{O}(F)$ and the data structure. According to the partition size property, the hypercube [7] and simplices [19] are not scalable for high dimensional search space. This non-scalability impacts the complexity of the partitioning function F.

According to the building complexity property $\mathcal{O}(F)$, the Voronoï cell needs a particular close attention. Indeed, most of the algorithms used for computing the Voronoï diagram are well studied for 2D and 3D (e.g. QuickHull [42], Fortune's algorithm [43], Bowyer-Watson algorithm [44]). However, they suffer from the curse of dimensionality. Hence, one has to use heuristics to create such fractals. Indeed, one cannot build the exact diagram in high dimensions [18, 45–48], and then one has to use stochastic methods to build an approximation, such as SpokeDart with hyperplane sampling [46]. We distinguish two approaches, the fixed (Figure 5f) and dynamic (Figure 5e) Voronoï diagrams. In the fixed Voronoï diagram, a child cell will inherit its parent's boundaries, whereas in the dynamic diagram, when building a child, the whole diagram is re-computed to consider children's boundaries. This approach implies variations of the borders of fractals. In a dynamic approach, a Voronoï cell is a very complex type



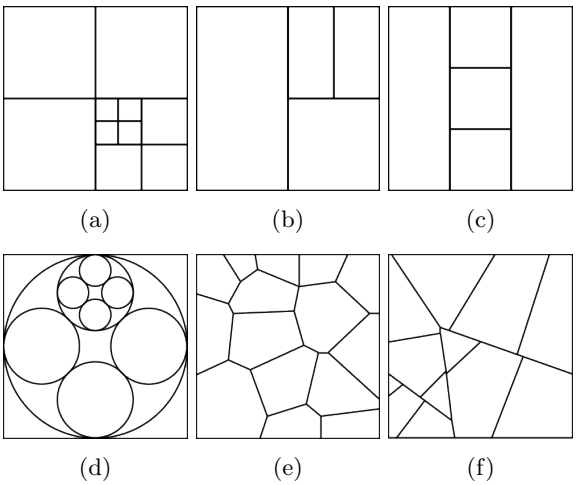
(a) Proper preservative (b) Improper preservative

(c) Lowly sacrificial (d) Highly sacrificial

(e) Unbounded sacrificial

Fig. 4: 2-dimensional illustrations of Theorem 2. The hatched-magenta area corresponds to uncovered space resulting from successive decomposition: $C(\Lambda, \cup\mathcal{L}_{\mathcal{T}^{(d)}})$. Solid-green corresponds to overlapping: $O(\mathcal{L}_{\mathcal{T}^{(d)}})$.

of fractal, as its definition depends on varying information on other fractals. For example, when recomputing the diagram, points sampled in a dynamic Voronoï cell, can be relocated to other ones. Whereas in the fixed approach, once a fractal, i.e., a Voronoï cell is built, then its borders cannot be changed anymore. Other hypervolumes showed in Figure 5 are easier to compute, and further details can be found in [7, 8, 10].



(a) (b) (c)

(d) (e) (f)

Fig. 5: Various examples of fractals in 2 dimensions: (a) hypercubes, (b) bisections, (c) trisections, (d) hyperspheres, (e) dynamic Voronoï, (f) fixed Voronoï.

Furthermore, if we look at the coverage and overlap properties, one can clearly see the major drawback of hyperspheres compared to other hypervolumes. $C(\Lambda, \mathcal{L}_{\mathcal{T}^{(d)}})$ and $C(S_{l,j,i}, \bigcup F(S_{l,j,i}))$ measure the uncovered space. If these measures are strictly superior to 0, then the partition does not fully cover the search space or a parent fractal.

According to [8], the inflation ratio applied on n-spheres can reduce the substantial lack of coverage by increasing the overlap. However, we have to mention the impact of the curse of dimensionality on such objects. Indeed, if we look at the Hausdorff measure of a unit n-ball, the n-volume and the n-surface tend to zero as the dimension tends to infinity. This can explain the empirical results obtained in this paper, indicating that the deeper is the decomposition tree, the lower FDA performs.

Figure 5 illustrates that 2D representations are misleading. Even if they are intuitive and allow to better understand basic principles, we cannot infer what will happen in n-dimension by only looking at 2D or 3D drawings. Thus, the choice of a fractal can be informed by carefully examining the six properties defined in Table 1.

Table 1: Example of fractals and their properties

	$S_{l,j,i}$	n-cube	Trisection	Bisection	n-sphere	Voronoï	Simplex
Partition size	k	2^n	3	2	$2n$	c [a]	$n!$
Partition building complexity	$\mathcal{O}(F)$	$\mathcal{O}(2^n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(2^n)$ [b]	$\mathcal{O}(n!)$
Coverage of Λ	$C(\Lambda, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}})$	0	0	0	> 0	0	0
Children coverage	$C(A, \bigcup F(A))$	0	0	0	> 0	0	0
Overlap	$O(\mathcal{L}_{\mathcal{T}^{(d)}})$	0	0	0	> 0	0 or > 0 [c]	0
Data structure	$\emptyset$	center and side length	2 points of size n	2 points of size n	center and radius	See [c]	$n+1$ points of size n
Examples	$\emptyset$	[7]	[9]	[17]	[8]	[15, 18]	[19]

[a] Number of centroids defined by the user.
[b] Valid for usual algorithms, we can reduce this complexity by approximating the Voronoï diagram in high dimensions. This complexity, also depends on c, the number of centroids. But here we consider the complexity depending on the dimension n.
[c] It depends on the algorithm used to compute the diagram. It can be a set of vertices for the QuickHull algorithm or a set of hyperplanes for sampling methods. An exact algorithm or a heuristic approach impacts the properties.

5 Tree search

One can define a tree search algorithm as a function τ which selects Q unique fractals among all $\mathcal{L}_{\mathcal{T}^{(d)}}$ according to a vector of size $s = |\mathcal{L}_{\mathcal{T}^{(d)}}|$:

$$\tau : \mathcal{L}_{\mathcal{T}^{(d)}}, P(\mathcal{L}_{\mathcal{T}^{(d)}}), \mathbb{R}^s \rightarrow \{e1, ..., e_Q\} \ , \quad (13)$$

where $\forall q \in [1, ..., Q] : e_q \in \mathcal{L}_{\mathcal{T}^{(d)}}$, $1 \leq Q \leq s$. The vector $\mathbb{R}^s$ resulting from the scoring search component describes the quality value of each leaf. The properties of the leaves are denoted $P(\mathcal{L}_{\mathcal{T}^{(d)}}) = \{P(a, \mathcal{A}_a) \,|\, \forall a \in \mathcal{L}_{\mathcal{T}^{(d)}}\}$

One can instantiate tree search algorithms using an `OPEN` and `CLOSED` lists [31]. The `OPEN` list contains non-explored or non-exploited fractals. The `CLOSED` list contains expanded fractals. Thus, the tree search component is a rule defining how to append non-expanded fractals to these lists, and how to select them. The `CLOSED` list is generally used to prevent the algorithm from selecting expanded fractals.

When building a k-refinement, tree search algorithms are crucial and have an impact on the exploration and exploitation tradeoff. In the DIRECT algorithm, this issue is tackled by the POH strategy, and many variations of DIRECT are based on this selection strategy [21, 41]. The FDA algorithm uses a sorted *depth first search*, called Move-up, which favors the selection of deep fractals. The FDA algorithm focuses on the exploitation phase applied at the last level D. Such strategy impacts the capacity of the algorithm to efficiently explore the search space.

Moreover, *Breadth First Search* (BrFS) is totally useless in our case. Indeed, all fractals at a certain level will be decomposed before selecting fractals of the next level. Thus, *Depth First Search* (DFS) can be considered a greedy *exploitation only*, whereas BrFS can be qualified as a greedy *exploration only* algorithm [49]. Both strategies are ineffective, as the scores given to fractals do not impact the selection. We lose the notion of hierarchy between fractals.

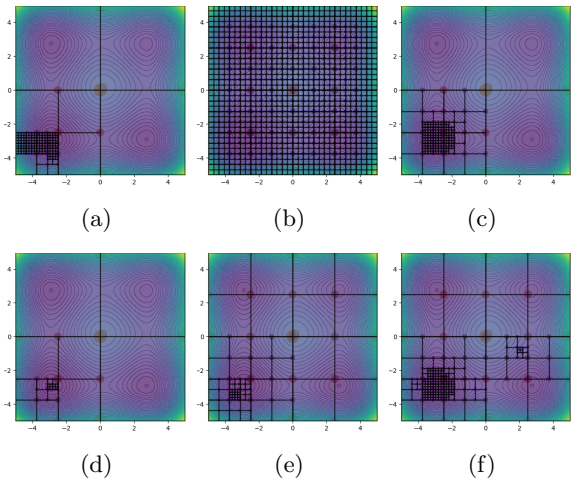


Fig. 6: Fractal decomposition with hypercubes applied on 2D Styblinski-Tang function with various tree search: (a) Depth First Search, (b) Breadth First Search, (c) Best First Search, (d) Beam Search, (e) Cyclic Best First Search, (f) Epsilon Greedy Search.

A tree search algorithm can be characterized by its balance between exploration and exploitation of the tree. Thus, one efficient and easy to use algorithm, which can replace DFS or BrFS, is the Best First Search [31] algorithm. Some of these tree search are stochastic (e.g. Epsilon Greedy Search, Diverse Best First Search [50]) or, others, allow having the hand on the exploration-exploitation tradeoff (e.g. Cyclic best First Search [51]). The notion of sacrifice defined in Theorem 2 also concerns tree search algorithms and pruning techniques. For time and memory complexity reasons, it can be necessary to prune some leaves. For instance, in the *Beam Search* algorithm [32], only a given number of leaves are stored.

Figure 6 illustrates how the tree search algorithm impacts the behaviors of *fractal decomposition-based* algorithms. One can clearly see the difference between depth and breadth first search. BrFS explored the entirety of each level before tackling the next one, no matter the hierarchy. Whereas DFS is focused on the deepest fractal.

6 Scoring fractals

The tree search component needs a heuristic value determining how promising a fractal is. We define a scoring method γ which takes a set of solutions restricted to a fractal and assigns a quality value. Such as in [1, 13], γ is similar to the *characteristic value* defining the probability that a fractal contains the global optimum:

$$\gamma : \alpha,\ P(\alpha, \mathcal{A}_\alpha),\ R,\ f|_\alpha(R) \to \mathbb{R} \qquad (14)$$

The scoring method γ takes a fractal α, a sample of solutions R restricted to the region of α, and their corresponding objective values $f|_\alpha(R)$. The properties of the leaves, denoted $P(\alpha, \mathcal{A}_\alpha)$, can also be considered. The function returns a score within $\mathbb{R}$ which defines the quality of α.

For DIRECT and SOO, γ returns $f(x)$, with x the center of the fractal. In FRACTOP, the *Belief* is used. It depends on a fuzzy measure computed with the best evaluation found so far, and sampled points within the fractal. This score has the particularity to inherit a part of the parent's score. In FDA, the algorithm maximizes the *distance-to-the-best* solution found so far among all sampled solutions.

The combination between τ and γ is essential, and has different purposes concerning the exploration and exploitation tradeoff. Finally, τ and γ can use additional information, such as a measure of the size of a fractal. For example, the σ function in DIRECT, with σ_2 or σ_∞, measures the size of hyperrectangles according to their level and the length of their longest side [41]. We can notice that for a fractal for which its children are smaller and of identical size (regularity), the level of a fractal can be considered as a measure of its size. This is the case for FDA, but not in DIRECT and for Voronoï fractals.

7 Exploration and exploitation strategies

The exploration *Explore* and exploitation *Exploit* search components can be defined by the following functions applied on a fractal α:

$$\begin{aligned} &\mathit{Explore} : \alpha,\ P(\alpha, \mathcal{A}_\alpha) \to R,\ f|_\alpha(R) \\ &\mathit{Exploit} : \alpha,\ P(\alpha, \mathcal{A}_\alpha) \to R,\ f|_\alpha(R)\ , \end{aligned} \qquad (15)$$

where R is a set of sampled point restricted to the region of the fractal α.

To assess the potential of a fractal, one has to sample relevant solutions from it. This is the purpose of the *Explore* function. For instance, FRACTOP uses a genetic algorithm, whereas in FDA, three fixed solutions are computed.

Sampling in hyperrectangles or hypercubes is a simple procedure. One can apply all sampling methods or metaheuristics using infima and suprema. For example, low discrepancy sequences such as Sobol, Halton, and Kronecker methods can be used [35]. Sampling in a hypersphere requires a few tricks to satisfy the equation of a n-ball. The Box-Muller method can be a solution [52–54]. However, sampling inside a Voronoï cell (i.e. polytope) is a complex procedure. One could use methods to approximate the Lebesgue measure [55], hit-and-run sampling [56], hyperplanes sampling [46], or MCMC sampling [57]. For active algorithms, such as metaheuristics (e.g. local search, evolutionary algorithms, swarm optimization), one must adapt the search operators (e.g. neighborhood, mutation, crossover, velocity update) to the type of fractal.

Finally, if the algorithm lacks of exploitation, one could apply an exploitation algorithm within leaves. In our framework, this algorithm is named *Exploit*. Such a function starts from a solution within a leaf fractal and if needed can ignore its boundaries to converge towards an optimum. This is the case for FRACTOP and FDA algorithms.

It is a thorough task to select an exploration and an exploitation search components. In our framework, the exploitation search component is exclusively applied to fractals of maximum depth to emphasize the search around promising areas. Consequently, it is appropriate to allocate larger budget to the exploitation compared to the exploration strategy within each fractal. However, according to the cost of the objective function, the maximum depth of the tree or the partition size (i.e. number of children per fractal), one should carefully choose the budget of the exploration and exploitation. Indeed, a budget that is too high for the exploration search component, can result in a low exploitation phase. Whereas, a budget that is too low for the exploration phase, can result in expensive exploitation in low confidence areas.

Algorithm 1 Fractal-based decomposition algorithm

Inputs:
Λ — *Initial search space*
D — *Maximum depth*
F — *Fractal decomposition function*
Explore — *Exploration strategy*
Exploit — *Exploitation strategy*
τ — *Tree search*
γ — *Scoring*
Outputs: $\hat{x}$ — *Best solution found*

$\hat{x} \leftarrow \infty$
`OPEN` $\leftarrow \{\Lambda\}$ — *List of non-expanded fractals*
`CLOSED` $\leftarrow \{\cdot\}$ — *List of expanded fractals*
`current` $\leftarrow \{\Lambda\}$
`scores` $\leftarrow \{+\infty\}$
while stopping criterion not reached **do**
  for each `leaf` $\in$ `current` **do**
    `children` $\leftarrow F($`leaf`$)$ — *Decomposition*
    for each `child` $\in$ `children` **do**
      if `level(child)` $< D$ **then**
        R, `values` $\leftarrow$ *Explore*(`child`)
        `score` $\leftarrow \gamma($`child`$, R,$ `values`$)$
        Append `child` **to** `OPEN`
        Append `score` **to** `scores`
        if min(`values`) $< \hat{x}$ **then**
          $\hat{x} \leftarrow$ min(`values`)
      else
        `values` $\leftarrow$ *Exploit*(`child`)
        if min(`values`) $< \hat{x}$ **then**
          $\hat{x} \leftarrow$ min(`values`)
    Append `leaf` to `CLOSED`
    `index` $\leftarrow$ Index of `leaf` in `OPEN`
    Remove element at `index` **from** `OPEN`
    Remove element at `index` **from** `scores`
  `current` $\leftarrow \tau($`OPEN`$,$ `scores`$)$
return $\hat{x}$

8 Instantiating algorithms within Zellij

This section introduces how to instantiate some popular fractal-based optimization algorithms within Zellij. Moreover, it shows how one can extend these algorithms using various search strategies for the different components. These algorithms are presented, per search components, in Table 2

8.1 FDA

FDA decomposes the search space Λ by using $2n$ hyperspheres. The initial center C_1 of Λ is computed by $C_1 = L + \frac{U-L}{2}$. The radius of the hypersphere of center C_1 is $r_1 = \frac{U-L}{2}$. So the region of $S_{l,i,j}$ is defined by the hypersphere of center $C_{l,i,j}$ and its radius r_l at level l. To simplify notation, only $C_{.,.,.}$ are considered instead of the region $S_{.,.,.}$. Centers are points of size n from the search space, $C_{.,.,.} \in \Lambda$. The partition operator F is defined by:

$$F(C_{l,.,.}) = \{C_{l+1,.,j} \,|\, \forall j \in [1,\ldots,n], \\ C_{l+1,.,j} = C_{l,.,.} + (-1)^j (r_l - r_{l+1}) \vec{e_j}\} \ ,$$

where $\vec{e_j}$ is the unit vector at dimension j and $r_{l+1} = \frac{r_l}{1+\sqrt{2}}$.

The PHS, i.e. the exploration component, consists in computing the center of the current hypersphere and two symmetrical points. So, $Explore : S_{l,i,j},\ P(S_{l,i,j}, \mathcal{A}_{S_{l,i,j}}) \rightarrow R,\ f|_{S_{l,i,j}}(R)$, where

$$R = \left\{ C_{l,i,j} - \lambda \frac{r_l}{\sqrt{n}},\ C_{l,i,j},\ C_{l,i,j} + \lambda \frac{r_l}{\sqrt{n}} \right\} \ ,$$

where $\lambda > 1$ is the inflation ratio.

From these 3 sampled points, the scoring method is defined by taking the maximum of a slope, such that,

$$\gamma : S_{l,i,j},\ R \rightarrow \max_{r \in R} \left(\frac{f(r)}{\|r - \texttt{BSF}\|} \right) \ ,$$

where `BSF` is the best solution found so far.

Concerning the tree search component, i.e. Move-Up, it is comparable to a sorted Depth First Search algorithm. Here, the best fractal of maximum current depth d is selected at each iteration, only if it is not a fractal of maximum depth D. The Move-Up algorithm is described in Algorithm 2

The exploitation component, i.e. ILS, which is similar to a coordinate search is defined in [8, 40].

Algorithm 2 Move-Up

Inputs:
- $\mathcal{L}_{\mathcal{T}^{(d)}}$ — *Leaves*
- d — *Current depth*
- D — *Maximum depth*
- γ — *Scoring component*

Outputs: α — *Selected fractal*

```
if d = D then
    depth ← d − 1
else
    depth ← d
α ← ∅
while (α = ∅) ∧ (depth>1) do
    leaf = {S_{depth,i,j} | ∀i, j, S_{depth,i,j} ∈ L_{T^(d)}}
    α ← argmin_{a∈leaf}(γ(a))
    depth ← depth − 1
return α
```

8.2 SOO and NMSO

SOO and NMSO divide the search space Λ by using trisections ($K = 3$) along the current longest side. Similarly to the Definition 1, here the region of a fractal $S_{l,i,j}$ is defined by a hyperrectangle of bounds $L_{l,i,j}$ and $U_{l,i,j}$. So the simplified partition operator can be written,

$$F((L_{l,i,j},\ U_{l,i,j})) = \{(L_{l,i,j} + k \cdot \lambda \cdot \vec{e_\lambda}, \\ U_{l,i,j} - (K - k - 1) \cdot \lambda \cdot \vec{e_\lambda})\}_{k \in [0,\cdots,K-1]} \ ,$$

where λ is the index of the longest dimension of $S_{l,i,j}$ $\vec{e_\lambda}$ is the unit vector at λ.

The exploration component of SOO and NMSO, consists in computing the center of each fractal. So, $Explore : S_{l,i,j},\ P(S_{l,i,j}, \mathcal{A}_{S_{l,i,j}}) \rightarrow R,\ f|_{S_{l,i,j}}(R)$, where

$$R = \left\{ \frac{U_{l,i,j} - L_{l,i,j}}{2} \right\} \ ,$$

Then, the scoring component is the objective value of the center $\gamma : S_{l,i,j},\ R \rightarrow f(R)$. The tree search of SOO consists in selecting, within the tree $\mathcal{T}^(d)$ in a top-down manner, the best fractal at each level only if it is better than all fractals of previous levels. This component is described in Algorithm 3.

Algorithm 3 SOO tree search

Inputs:

$\mathcal{L}_{\mathcal{T}^{(d)}}$ *Leaves*
d *Current depth*
D *Maximum depth*
γ *Scoring component*

Outputs: A *Selected fractals*

$\texttt{depth} \leftarrow 1$
$v_{\texttt{max}} \leftarrow +\infty$
$A \leftarrow \{\cdot\}$
while $\texttt{depth} < \min(d, D)$ **do**
  $\texttt{leaf} = \{S_{\texttt{depth},i,j} \,|\, \forall i, j, S_{\texttt{depth},i,j} \in \mathcal{L}_{\mathcal{T}^{(d)}}\}$
  $\alpha \leftarrow \underset{a \in \texttt{leaf}}{\operatorname{argmin}}(\gamma(a))$
  if $(\alpha \neq \emptyset) \wedge (\gamma(\alpha) \leq v_{\texttt{max}})$ **then**
    $A \leftarrow A \cup \{\alpha\}$
    $v_{\texttt{max}} \leftarrow \gamma(\alpha)$
  $\texttt{depth} \leftarrow \texttt{depth} + 1$
return A

The tree search of NMSO is more complex, we give here an overview of the algorithm, more details about the implementation are found in [24] and their source code [2]. This component balances in width and depth exploration of $\mathcal{T}^{(d)}$, by building sequences of depth first search selection. Then according to criteria based on the slopes between centers of the children of a fractal, and their sizes, NMSO restarts a new sequence from one of the highest (low level), non-expanded fractal. The fractals from stopped sequences are put in a *basket*. If after a certain number of iterations these leaves from the basket are visited a certain number of times according to their quality, then they are selected to resume the stopped sequences.

SOO and NMSO do not have any exploitation component. The balance between exploration and exploitation is mainly guided by the tree search component.

8.3 DIRECT

The instantiation of DIRECT within *Zellij* is based on the following user guide [58]. Like SOO and NMSO, DIRECT is based on trisections applied to hyperrectangle of bounds $L_{l,.,.}$ and $U_{l,.,.}$ for a level l. The partition operator requires the exploration of the fractal to sort the dimensions of maximal sizes of a given fractal. Then, a series of trisections along all dimensions of maximal sizes, is iteratively applied on the resulting central hyperrectangles. The exploration of a fractal $S_{l,.,.}$ can be written has: *Explore* : $S_{l,.,.}, P(S_{l,.,.}, \mathcal{A}_{S_{l,.,.}}) \rightarrow R,\ f|_{S_{l,.,.}}(R)$, where

$$R = \{C_{l,.,.} - \lambda \vec{e_j},\ C_{l,.,.} + \lambda \vec{e_j}\}_{j \in I} \quad ,$$

with the center of a fractal $C_{l,.,.} = \frac{U_{l,.,.} - L_{l,.,.}}{2}$, the set I containing the indices of the dimensions of maximum size for a given fractal, $I = \text{argmax}_{i \in [1,\cdots,n]}(U_{l,.,.}[i] - L_{l,.,.}[i])$, and λ is equal to one third of the longest side of a given fractal, $\lambda = \max(U_{l,.,.} - L_{l,.,.})/3$. Once, a hyperrectangle has been explored, the resulting points along all dimensions of maximum sizes are used to iteratively trisects the fractal, refer to [58] for more details.

The tree search algorithm of DIRECT is complex and bounds the Lipschitz constant of leaves $\mathcal{L}_{\mathcal{T}^{(d)}}$ by building the set of POHs, i.e. a tradeoff between the score of a fractal and its size. The selection of POHs is described in Algorithm 4 and is based on [58]. For each leaf, the algorithm builds three sets of fractals according to their size. One set is made of leaves that are of equal size and two others are made of leaves that are bigger, respectively smaller, than the current leaf. Then, by computing different inequalities, the algorithm determines if the Lipschitz constant can be bounded or not. So, here we have to include the size of fractals to their properties, $P(S_{l,.,.}, \mathcal{A}_{S_{l,.,.}}) = \sigma(S_{l,.,.})$, where σ is a measure of the size of a hyperrectangle [41]. In the pseudocode the best fractal found so far (the best center) is denoted $\texttt{BSF}$, $\epsilon > 0$ is a small value defining how a score of a fractal should exceed $\texttt{BSF}$ to be considered better.

As SOO and NMSO, the scoring of a fractal is the objective value of its center, and there is no exploitation component.

Two extensions of DIRECT, known as *Locally biased DIRECT*(DIRECT-L) [41], and *DIRECT restart*(DIRECT-R) [21], slightly modify Algorithm 4 and the measure σ, to obtain different behaviors.

[2] https://github.com/ash-aldujaili/NMSO/tree/master

Algorithm 4 POHs

Inputs:

$\mathcal{L}_{\mathcal{T}^{(d)}}$ *Leaves*
d *Current depth*
D *Maximum depth*
γ *Scoring component*
ϵ

Outputs: A *POHs*

$A \leftarrow \{\cdot\}$
for each $a \in \mathcal{L}_{\mathcal{T}^{(d)}}$ **do**
  $I_1 = \{b \,|\, \forall b \in \mathcal{L}_{\mathcal{T}^{(d)}}, \sigma(b) < \sigma(a)\}$
  $I_2 = \{b \,|\, \forall b \in \mathcal{L}_{\mathcal{T}^{(d)}}, \sigma(b) > \sigma(a)\}$
  $I_3 = \{b \,|\, \forall b \in \mathcal{L}_{\mathcal{T}^{(d)}}, \sigma(b) = \sigma(a)\}$
  if $\gamma(a) \leq \gamma(b), \forall b \in I_3$ **then**
    $\texttt{max}_{I_1} = \max_{b \in I_1} \left(\frac{\gamma(a)-\gamma(b)}{\sigma(a)-\sigma(b)} \right)$
    $\texttt{min}_{I_2} = \min_{b \in I_2} \left(\frac{\gamma(b)-\gamma(a)}{\sigma(b)-\sigma(a)} \right)$
    if $\texttt{BSF} = 0$ **then**
      $\texttt{err} = -\epsilon + \frac{\gamma(\texttt{BSF})-\gamma(a)}{|\gamma(\texttt{BSF})|} + \frac{\sigma(a) \cdot \texttt{min}_{I_2}}{|\gamma(\texttt{BSF})|}$
    else
      $\texttt{err} = -\gamma(a) + \sigma(a) \cdot \texttt{min}_{I_2}$
    if $(\texttt{min}_{I_2} - \texttt{max}_{I_1} > 0) \wedge (\texttt{err} \geq 0)$ **then**
      $A \leftarrow A \cup \{a\}$ *a is a POH*
return A

8.4 Modifications of popular fractal-based algorithms

Thanks to our generic framework, we have modified some previous algorithms to illustrate the flexibility of *Zellij*. Concerning FDA, we decided to replace the Move-Up algorithm, by a *Q-Best-First-Search* (Q-BFS), which is a greedy algorithm consisting in selecting the Q best nodes from $\mathcal{L}_{\mathcal{T}^{(d)}}$ at each iteration (FDA-BFS, FDA-DBFS). The same replacement was applied to SOO (SOO-BFS). The advantage of using Q-BFS, instead of *Beam search* for example, is that all fractals are kept. So for SOO, the entirety of the search space is still accessible, Q-BFS might modify its rate of convergence.

We also tested FDA and FDA-DBFS using deep trees, i.e. the maximum depth D was set to 5 for FDA and FDA-BFS, and to 10 for FDA-D and FDA-DBs. Additionally, we replaced the distance-to-the-best scoring (γ) of FDA by a centered version of it (FDA-C). The measure is centered on the best solution found so far,

$$\gamma : S_{l,i,j},\ R \rightarrow \min_{r \in R} \left(\frac{f(r) - f(\texttt{BSF})}{\|r - \texttt{BSF}\|} \right) \ .$$

In the original version of FDA, as the algorithm selects the highest ratio, the one associated to a better solution can be considered as non-promising if the point is far from BSF, whereas a bad solution can be considered as good if it is close to BSF.

9 Experimental setup

The objective of these comparisons is to illustrate the workability of our software framework by instantiating several popular algorithms within the *Zellij* framework. One can also evaluate their scalability and their sensitivity according to the different search components: fractal, tree search, scoring, exploration and exploitation components. We summed-up all 11 implemented algorithms and their different search components in Table 2.

Thus, the optimization algorithms were evaluated on the BBOB benchmark from the Comparing Continuous Optimizer framework [60]. This benchmark is made of 24 functions with peculiar properties such as, separability, ill-conditioning, multi-modality and weak structured multi-modality. Each function has 15 different instances, and are available for 6 dimensions, $n \in \{2, 3, 5, 10, 20, 40\}$. COCO compares algorithms on the number of solved problems, under a given tolerance $\Delta f = 10^{-8}$, such that,

$$f(\hat{x}) \leq f(x^\star) + \Delta f \ , \tag{16}$$

where $x^\star$ is the known global optimum and $f(\hat{x})$ is the acceptable optimum to consider a problem as solved. Then COCO computes a metric called the *Expecting Running Time* (ERT), based on the number of problems solved and the budget of the optimization. Here, the budget, budget, is the number of function evaluations and depends on the dimensionality of the problem, $\texttt{budget} = 10^4 \cdot n$, as in [24].

COCO is based on the number of problems solved. Hence, to extend the comparison, we also analyze the best solutions found for all problems, which might not follow the equation 16. To do so, we

Table 2: Instantiated algorithms using Zellij

Algorithm	Fractal	Tree Search	Explor.	Exploit.	Scoring	Depth	Source
FRACTOP	n-cube	Best First Search	GA	SA	Belief	4	[7]
FDA	n-sphere	Move-Up	PHS	ILS	DTTB	5	[8]
FDA-BFS	n-sphere	Q-BFS [a]	PHS	ILS	DTTB	5	This work
FDA-C	n-sphere	Move-Up	PHS	ILS	C-DTTB	5	This work
FDA-D	n-sphere	Move-Up	PHS	ILS	DTTB	10	[8]
FDA-DBFS	n-sphere	Q-BFS [a]	PHS	ILS	DTTB	10	This work
SOO	Trisection	Best at each level [b]	Center	$\emptyset$	$\emptyset$	$h_{\max}$ [c]	[9]
SOO-BFS	Trisection	Q-BFS [a]	Center	$\emptyset$	$\emptyset$	$h_{\max}$ [c]	This work
NMSO	Trisection	See [24]	Center	$\emptyset$	$\emptyset$	600	[9]
DIRECT	Trisection	All POH	Center	$\emptyset$	$\emptyset$	600 [d]	[10]
DIRECT-L	Trisection	1 POH per level	Center	$\emptyset$	$\emptyset$	600 [d]	[41]
DIRECT-R	Trisection	Adaptive POH	Center	$\emptyset$	$\emptyset$	600 [d]	[21]

[a] At each iteration $Q = n$(dimension) nodes are returned.
[b] If the best fractal at a given level is worse than one from previous levels, then it is not selected [9].
[c] Here $h_{\max} = 10\sqrt{\log(n10^4)^3}$ [59].
[d] The maximum depth is set to 600, as the `maxdeep` variable in the original FORTRAN implementation. In the original code, a `maxdiv` variable, set to 3000, limits the number of successive decomposition of a fractal. In *Zellij*, when the difference between an infimum and a supremum is inferior to a value ϵ set to $1e-16$, the fractal cannot be decomposed anymore.

use the two-sided Wilcoxon signed-rank test and corresponding mean ranks for each of the 11 algorithms. We perform this test for each subclass of functions and dimension. Thus, we were able to see their reliability. We define an error rate, p-value, of 5% for the statistical test. The two hypotheses are:

- **H0:** The two samples come from the same distribution.
- **H1:** The two samples come from different distributions.

The two-sided Wilcoxon signed-rank test is based on the mean of the ranking of all 11 algorithms for each subclass of functions and for each dimension. Summing these means and taking the lowest results is not enough to determine if an algorithm is better than another. Indeed, giving a rank to an algorithm for a function is arbitrary. An algorithm ranked second is not necessarily and significantly worse than the first one. We resumed the comparison between ranks and statistical test in Figures 10, Figure 11 and Figure 12. These figures can be read column by column. Each column represents comparisons of an algorithm (column label) with all other 10 algorithms (row labels). As an example, in Figure 12, for dimensions 3, if we focus on DIRECT-L (last column), we can compare it with SOO (sixth row). The color indicates

that there is a statistical evidence that DIRECT-L is better than SOO. In Figure 10, Figure 11 and Figure 12, there are three color codes:

- **Solid-Grey**: $\alpha > 0.05$, we cannot reject the null hypothesis.
- **Gridded-Green**: $\alpha \leq 0.05$ and $rank(column_i) < rank(row_j)$. We can reject the null hypothesis, and the algorithm with the label of the column i has a lower rank (is better) than the algorithm of the row j.
- **Dotted-Red**: $\alpha \leq 0.05$ and $rank(column_i) > rank(row_j)$. We can reject the null hypothesis, and the algorithm with the label of the column i has a higher rank (is worse) than the algorithm of the row j.

All experiments were carried-out on Grid'5000 [61], a large-scale and flexible testbed for experiment-driven research. We used a multi-CPUs cluster containing Intel Xeon Gold 5220 of 18 cores, each CPU has 96 GiB of RAM.

10 Results analysis and discussion

10.1 Sensitivity to the dimensionality

The initial observations reveal that for dimensions from 2 to 40, while FDA-based algorithms perform poorly on low dimensional problems, it can maintain certain performances when dimension grows. It is illustrated in Figures 8, 9, 11 and 12. For dimension 40, performances of FDA-based algorithms are comparable to SOO, SOO-BFS and NMSO. We can mention that FDA-D and FDA-DBFS always perform worse than FDA algorithms with a shallow tree. This confirms results obtained in [8], illustrating that the deeper the tree, the lower the performances of FDA. DIRECT-based algorithms are one of the best performing ones in low dimensionalities, with performances comparable to SOO and NMSO. However, it scales poorly, except for multi-modal functions, where all DIRECT algorithms perform better than FDA ones, but are worse than SOO and NMSO. It is important to notice the low number of successes in solving high dimensional problems in Figures 8 and 9, 11. Therefore, the two-sided Wilcoxon signed-rank tests illustrated in Figures 10, 12, and 12, are interesting to get a more in-depth analysis.

So, the algorithms that scale the best up-to dimension 40, appear to be SOO, NMSO and FDA. However, because FDA and SOO-BFS are greedy algorithms, they perform poorly in lower dimensions has they seem to be easily trapped into local optima.

10.2 Sensitivity to the tree search

In the following lines, we focus on FDA-BFS, SOO-BFS and FDA-BFS. Concerning, SOO-BFS, it is clear that the original tree search algorithm performs better than the Q-BFS. So, replacing this search component from SOO by one that is too greedy has a significant impact on the performances of the algorithms. The same observations can be made for FDA, FDA-BFS and FDA-DBFS.

Concerning DIRECT, DIRECT-R and DIRECT-L, it appears that biasing DIRECT toward exploitation (DIRECT-L), helps to scale the algorithm in higher dimensions. While in Figure 9, DIRECT-L solves almost always more problems than DIRECT, their performances (in ERT according to `budget`) for dimension 40 are comparable. To break the tie, the two-sided Wilcoxon signed-rank tests, in Figures 10, 12 shows that DIRECT-L is more likely to find a better solution than DIRECT, even if it cannot always solve the problem. However, DIRECT-R seems to be worse than DIRECT and DIRECT-L.

This section, illustrates how fractal-based decomposition algorithms are sensitive to the tree search algorithm and its parametrization. So, a particular attention is needed to the design of this search component.

10.3 Sensitivity to function properties

Concerning *separable* functions, NMSO and SOO are ones of the top algorithms, no matter the dimensions. When comparing the ERT in Figures 7 and 8, their performances in solving low dimensional problems are comparable to DIRECT-L and DIRECT. However, when dimensions grow, FDA-based algorithms, except FDA-D and FDA-DBFS, benefit from their exploitation

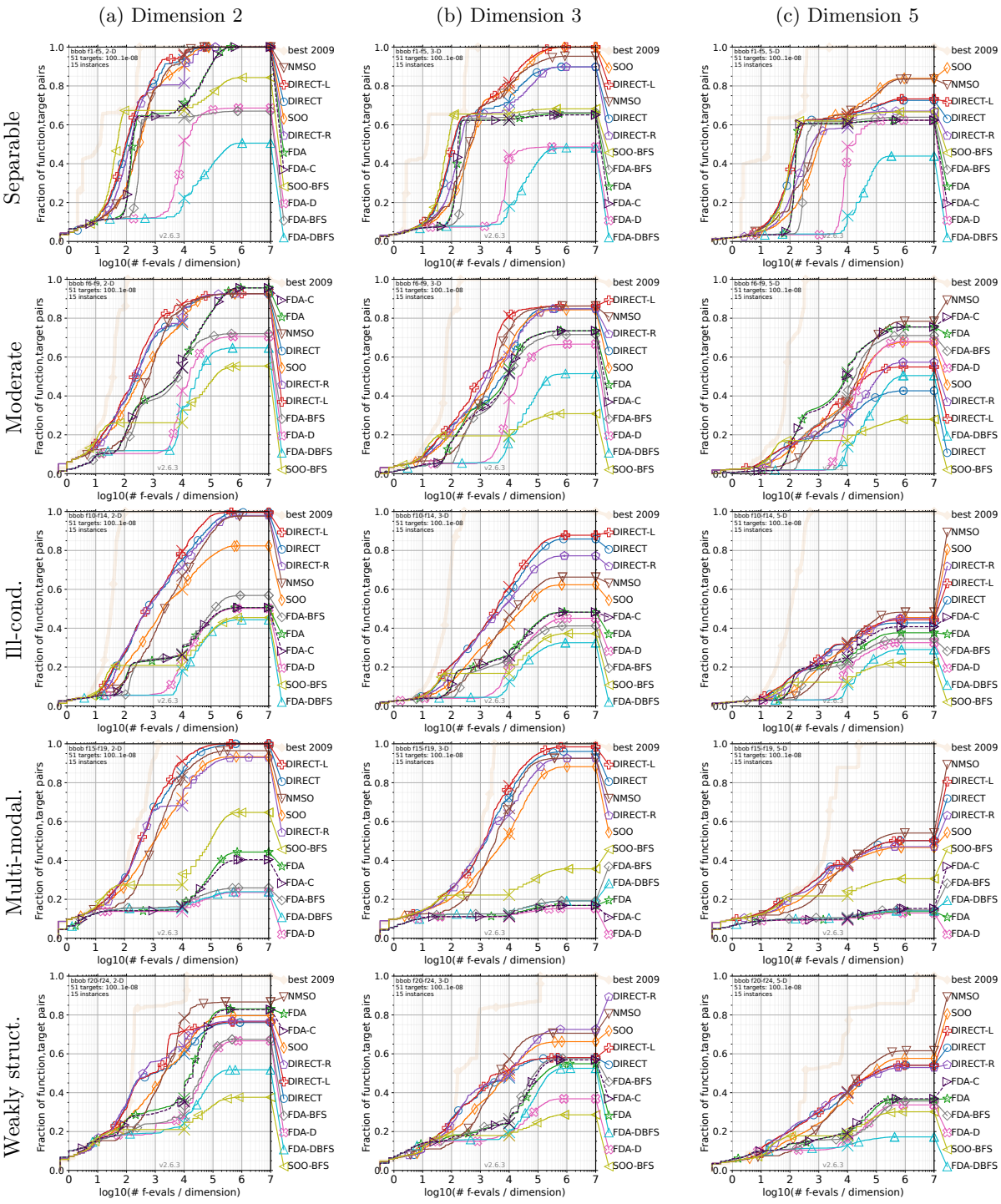


Fig. 7: Empirical cumulative distribution according to the `budget` divided by the dimensions for each functions' subclass and dimension 2, 3 and 5.

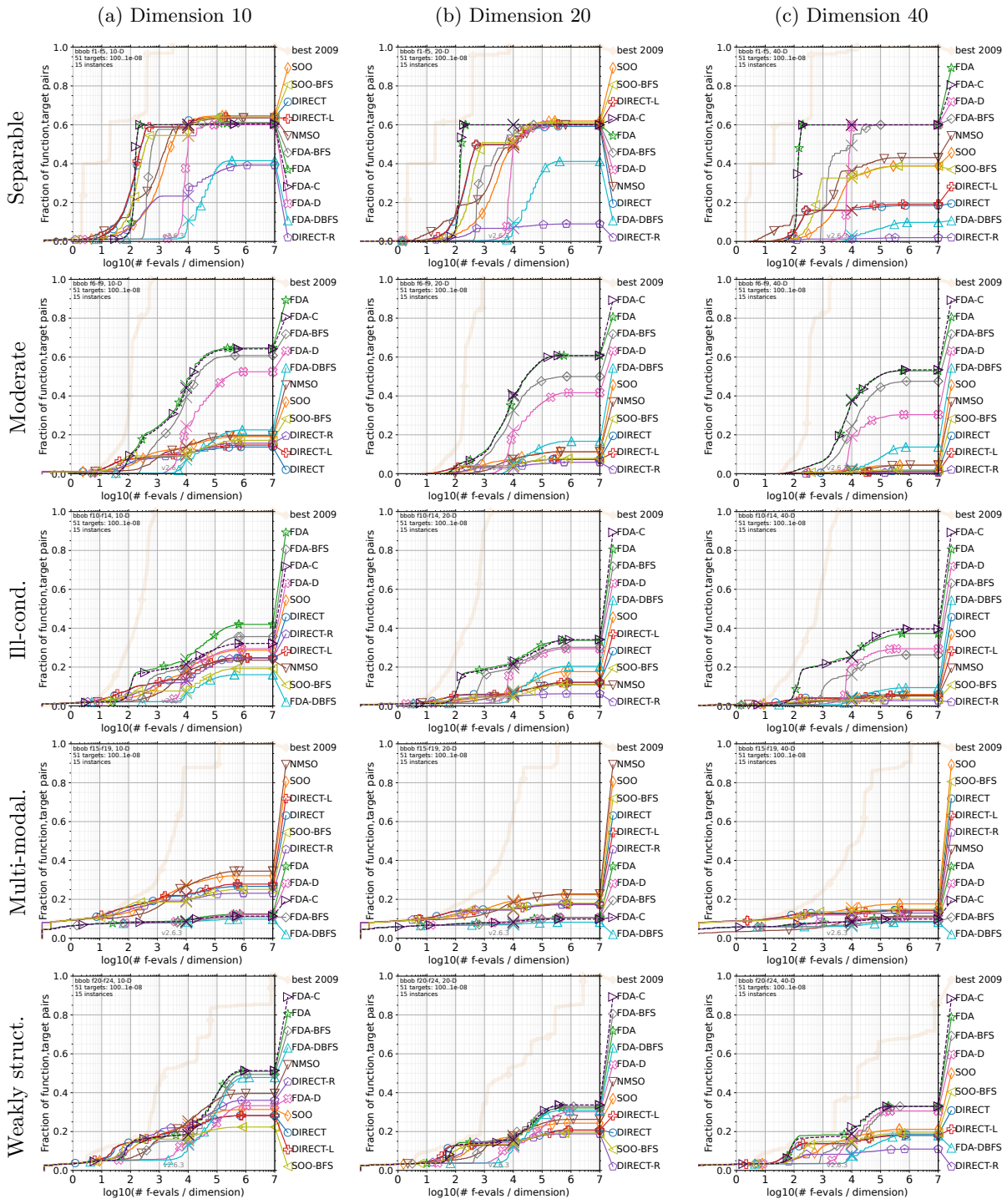
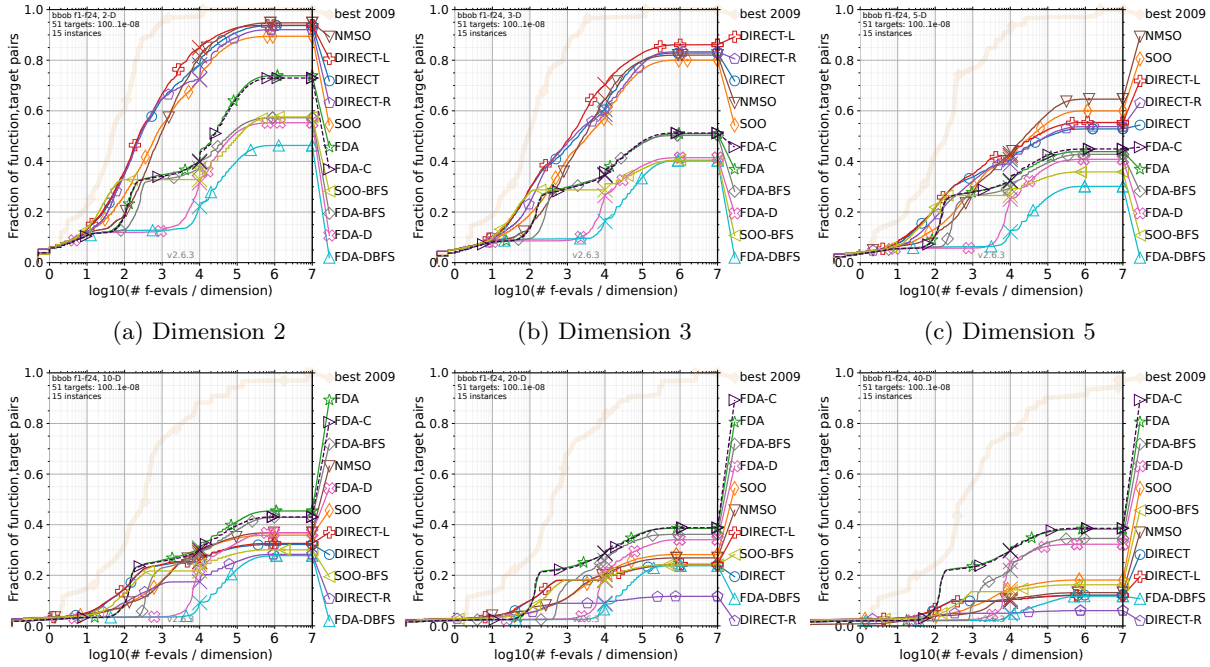


Fig. 8: Empirical cumulative distribution according to the `budget` divided by the dimensions for each functions' subclass and dimension 10, 20 and 40.



(a) Dimension 2 (b) Dimension 3 (c) Dimension 5

(d) Dimension 10 (e) Dimension 20 (f) Dimension 40

Fig. 9: Empirical cumulative distribution according to the `budget` divided by the dimensions for all functions and dimensions.

components to scale. When ERT and statistical tests are compared together, the exploitation component makes FDA competitive with NMSO and SOO, yet NMSO seems slightly superior.

For low dimensional (2, 3 and 5) and *moderate* problems, SOO, DIRECT, NMSO and FDA seems to perform similarly with a dominance of NMSO, SOO and DIRECT-L, in Figure 7. However, when the dimension grows, FDA becomes better, except for dimension 40, for which SOO, FDA and NMSO have similar behaviors.

Ill-condtioned problems are harder, and algorithms have difficulties scaling when the dimension grows. NMSO, DIRECT-L and SOO seem to dominate low dimensional problems. And, once again, for higher dimensions, FDA scales better. For dimension 40, FDA, SOO and NMSO have similar performances, even if they solve fewer problems than FDA. Which can be explained by the exploitation component of FDA, allowing the algorithm a faster convergence.

As mentioned in [8], the ILS of FDA seems to take advantage of separable functions, and can be easily dragged toward local optima. The *multi-modal* problems illustrate these behaviors, while SOO and NMSO are the best performing algorithms in high dimensions (10, 20 and 40), followed by DIRECT-based algorithms. The same behaviors are observable for *weakly structured multi-modal problems*, except that FDA performs better than DIRECT-based algorithms for dimension 40, and has similar performances compared to SOO and NMSO.

Finally, it appears that to design an efficient fractal-based algorithm, an exploitation component allows to significantly improve performances and convergence on certain problems. However, a greedy exploitation component, unable to escape from local optima, can decrease the performance of an algorithm. Thus, the balance between exploration and exploitation seems to rely on combining an exploitation component, an efficient partition of the search space and a not too greedy tree search.

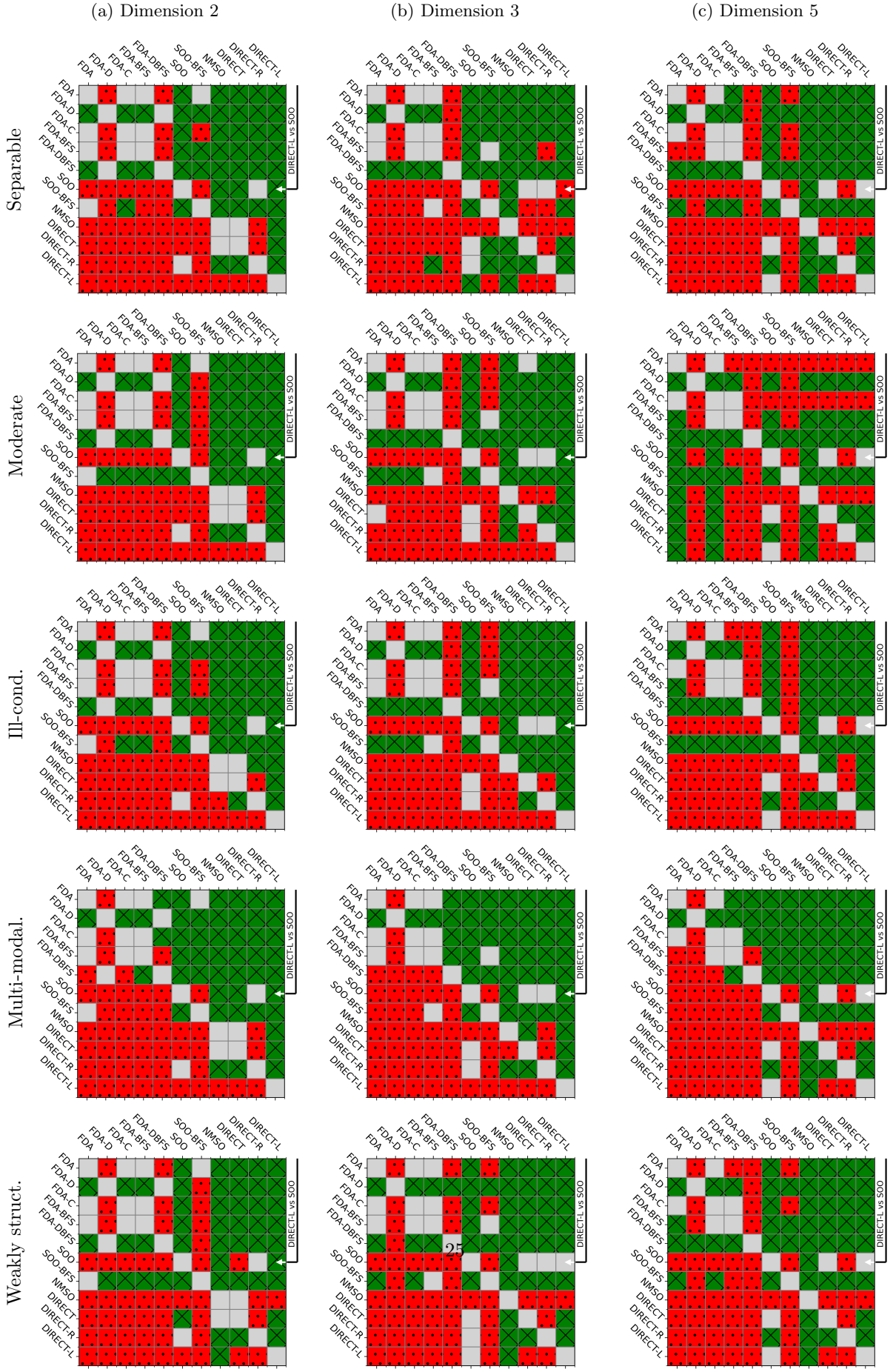


Fig. 10: Ranks and Pair Wise Wilcoxon test comparisons for each functions' subclass and dimensions 2, 3 and 5. **Solid-Grey**: Statistically unsignificative ($\alpha > 0.05$). **Gridded-Green**: Better. **Dotted-Red**: Worse.

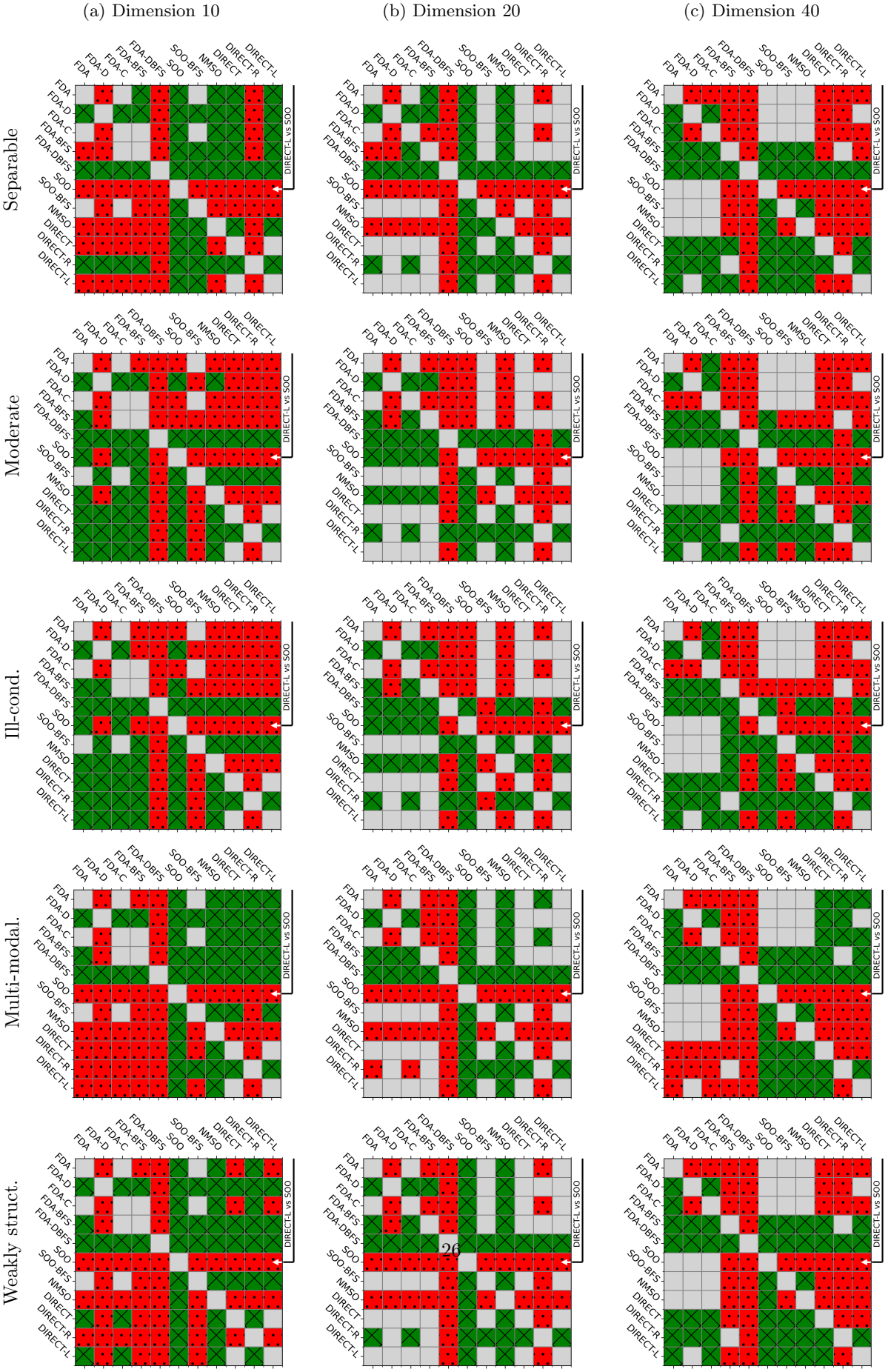
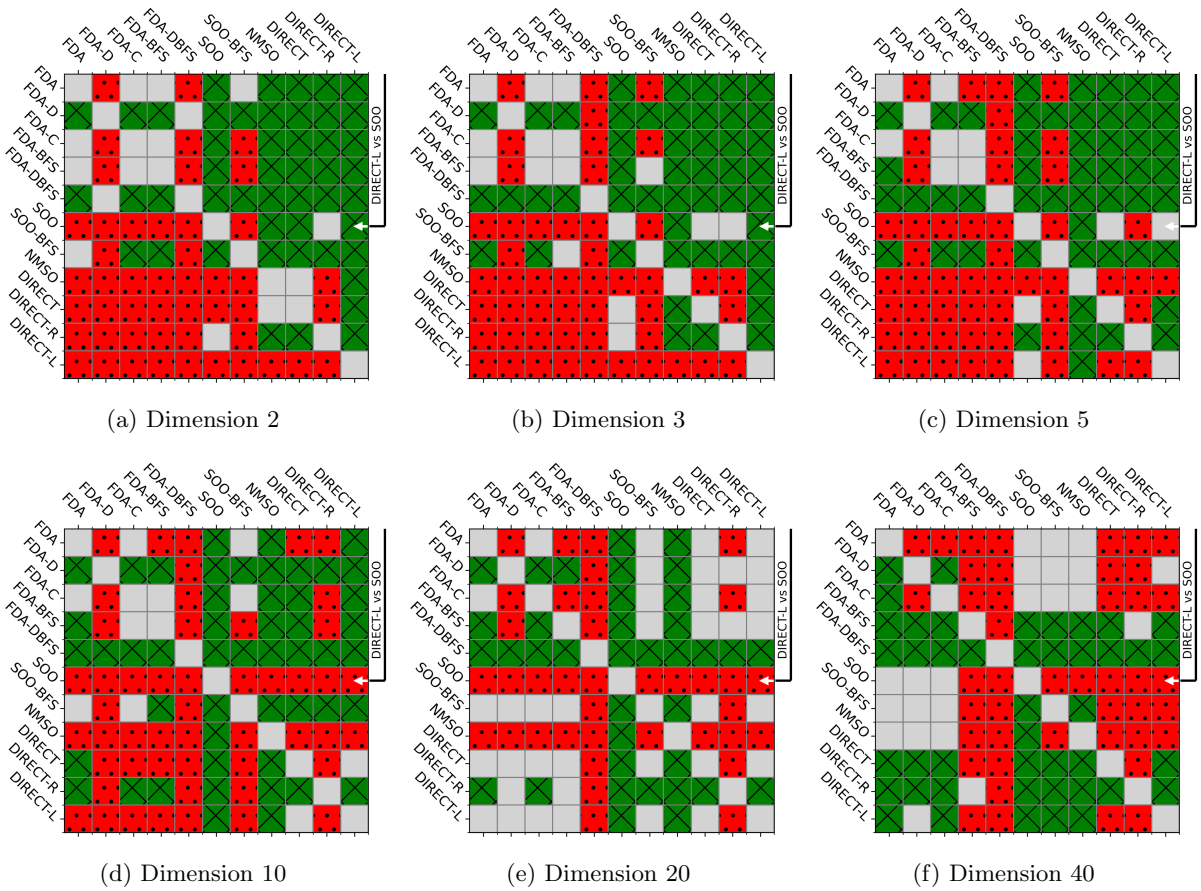


Fig. 11: Ranks and Pair Wise Wilcoxon test comparisons for each functions' subclass and dimensions 10, 20 and 40. **Solid-Grey**: Statistically unsignificative ($\alpha > 0.05$). **Gridded-Green**: Better. **Dotted-Red**: Worse.



(a) Dimension 2 (b) Dimension 3 (c) Dimension 5

(d) Dimension 10 (e) Dimension 20 (f) Dimension 40

Fig. 12: Ranks and Pair Wise Wilcoxon test comparison for all functions and dimensions. **Solid-Grey**: Statistically unsignificative ($\alpha > 0.05$). **Gridded-Green**: Better. **Dotted-Red**: Worse.

11 Conclusion

In this paper, we propose a unified and flexible algorithmic framework for fractal-based decomposition algorithms. This algorithmic framework allows modeling decomposition-based algorithms according to five search components: geometrical fractal, tree search, scoring, exploration, and exploitation. A Python package named *Zellij* has been developed and is available on GitHub[3]. Thanks to this framework, we can model various decomposition-based algorithms such as DIRECT, SOO, FRACTOP, FDA and much more. Furthermore, the modular programming standard used in *Zellij*, allows to prototype new algorithms quickly.

Computational results have been obtained comparing popular and extended deterministic algorithms according to the search components and their parameters. Their sensitivities to the dimension of the problem and to some of their components have also been analyzed. fractal-based algorithms are seemingly very sensitive to certain of their components, directly impacting the exploration-exploitation tradeoff. The proposed framework opens a door for developing and analyzing various search components to improve existing algorithms and designing new ones.

Future works will focus on the extension of the fractal-based decomposition algorithmic framework for combinatorial and mixed optimization problems, containing various types of variables (e.g. discrete, continuous, categorical). We will also investigate other geometrical fractal objects

[3] https://github.com/ThomasFirmin/zellij

such as Voronoï fractals and Latin hypercubes, which implies stochasticity.

In long-term works, we plan to tackle the massive parallelization of fractal-based decomposition algorithms on large scale heterogeneous architectures including multi-cores and accelerators such as GPU, towards Exascale architectures.

Because our algorithmic framework is made of five distinct and independent components, one could imagine an autonomous search for the best combinations of them. As it as already been done for population-based algorithm [29]. Thus, we could figure out autonomous *fractal decomposition-based* algorithms dedicated to specific problems [28]. In terms of applications, some computational experiments are under development on solving high-impact optimization problems such as hyper-parameter optimization and automated design of deep neural networks.

Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

This work has been supported by the University of Lille, the ANR-20-THIA-0014 program AI_PhDLille and the ANR PEPR Numpex.

Data availability statement

The datasets generated during the current study, containing raw data, summaries, and statistics of all experiments, are available free of charge on GitHub at https://github.com/ThomasFirmin/fdb_zellij_exp.

References

[1] Sergeyev, Y.D.: On convergence of "divide the best" global optimization algorithms. Optimization **44**(3), 303–325 (1998) https://doi.org/10.1080/02331939808844414 . Accessed 2024-04-03

[2] Stork, J., Eiben, A.E., Bartz-Beielstein, T.: A new taxonomy of global optimization algorithms. Natural Computing **21**(2), 219–242 (2022) https://doi.org/10.1007/s11047-020-09820-4 . Accessed 2024-05-05

[3] Locatelli, M., Schoen, F.: (Global) Optimization: Historical notes and recent developments. EURO Journal on Computational Optimization **9**, 100012 (2021) https://doi.org/10.1016/j.ejco.2021.100012 . Accessed 2024-05-05

[4] Talbi, E.-G.: Metaheuristics: from Design to Implementation. John Wiley & Sons, Hoboken, N.J (2009)

[5] Bansal, J.C.: In: Bansal, J.C., Singh, P.K., Pal, N.R. (eds.) Particle Swarm Optimization, pp. 11–23. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91341-4_2 . https://doi.org/10.1007/978-3-319-91341-4_2

[6] Garnett, R.: Bayesian Optimization. Cambridge University Press, Cambridge (2023). https://doi.org/10.1017/9781108348973

[7] Demirhan, M.e.a.: FRACTOP: A Geometric Partitioning Metaheuristic for Global Optimization. Journal of Global Optimization **14**(4), 415–436 (1999) https://doi.org/10.1023/A:1008384329041 . Accessed 2022-03-01

[8] Nakib, A., Ouchraa, S., Shvai, N., Souquet, L., Talbi, E.-G.: Deterministic metaheuristic based on fractal decomposition for large-scale optimization. Applied Soft Computing **61**, 468–485 (2017) https://doi.org/10.1016/j.asoc.2017.07.042 . Accessed 2022-03-01

[9] Munos, R.: Optimistic Optimization of a Deterministic Function without the Knowledge of its Smoothness, 9 https://doi.org/10.5555/2986459.2986547

[10] Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. Journal of Optimization Theory and Applications **79**(1), 157–181

(1993) https://doi.org/10.1007/BF00941892 . Accessed 2022-01-27

[11] Jones, D.R., Martins, J.R.R.A.: The DIRECT algorithm: 25 years Later. Journal of Global Optimization **79**(3), 521–566 (2021) https://doi.org/10.1007/s10898-020-00952-6 . Accessed 2022-03-01

[12] Al-Dujaili, A., Suresh, S., Sundararajan, N.: MSO: a framework for bound-constrained black-box global optimization algorithms. Journal of Global Optimization **66**(4), 811–845 (2016) https://doi.org/10.1007/s10898-016-0441-5 . Accessed 2024-03-22

[13] Horst, R., Tuy, H.: On the convergence of global methods in multiextremal optimization. Journal of Optimization Theory and Applications **54**(2), 253–271 (1987) https://doi.org/10.1007/BF00939434 . Accessed 2024-04-03

[14] Mandelbrot, B.B., Wheeler, J.A.: The fractal geometry of nature. American Journal of Physics **51**(3), 286–287 (1983) https://doi.org/10.1119/1.13295

[15] Khodabandelou, G., Nakib, A.: H -polytope decomposition-based algorithm for continuous optimization. Information Sciences **558**, 50–75 (2021) https://doi.org/10.1016/j.ins.2020.12.090 . Accessed 2022-03-01

[16] Shubert, B.O.: A Sequential Method Seeking the Global Maximum of a Function. SIAM Journal on Numerical Analysis **9**(3), 379–388 (1972) https://doi.org/10.1137/0709036 . Accessed 2022-09-29

[17] Paulavičius, R., Chiter, L., Žilinskas, J.: Global optimization based on bisection of rectangles, function values at diagonals, and a set of lipschitz constants. Journal of Global Optimization **71**(1), 5–20 (2016) https://doi.org/10.1007/s10898-016-0485-6

[18] Liu, H., Xu, S., Wang, X., Wu, J., Song, Y.: A global optimization algorithm for simulation-based problems via the extended DIRECT scheme. Engineering Optimization **47**(11), 1441–1458 (2015) https://doi.org/10.1080/0305215X.2014.971777 . Accessed 2022-01-27

[19] Paulavičius, R., Žilinskas, J.: Simplicial Lipschitz optimization without the Lipschitz constant. Journal of Global Optimization **59**(1), 23–40 (2014) https://doi.org/10.1007/s10898-013-0089-3 . Accessed 2022-07-22

[20] Mockus, J.: On the Pareto Optimality in the Context of Lipschitzian Optimization. Informatica **22**(4), 521–536 (2011) https://doi.org/10.15388/Informatica.2011.340 . Accessed 2022-11-25

[21] Finkel, D.E., Kelley, C.T.: An Adaptive Restart Implementation of DIRECT, 16

[22] Liuzzi, G., Lucidi, S., Piccialli, V.: A DIRECT-based approach exploiting local minimizations for the solution of large-scale global optimization problems. Computational Optimization and Applications **45**(2), 353–375 (2010) https://doi.org/10.1007/s10589-008-9217-2 . Accessed 2023-09-19

[23] Valko, M., Carpentier, A., Munos, R.: Stochastic Simultaneous Optimistic Optimization, 9 https://doi.org/10.5555/3042817.3042896

[24] Al-Dujaili, A., Suresh, S.: A Naive multi-scale search algorithm for global optimization problems. Information Sciences **372**, 294–312 (2016) https://doi.org/10.1016/j.ins.2016.07.054 . Citaaaation Key: nmso. Accessed 2024-05-05

[25] Horst, R., Tuy, H.: Global Optimization. Springer, Berlin, Heidelberg (1996). https://doi.org/10.1007/978-3-662-03199-5 . http://link.springer.com/10.1007/978-3-662-03199-5 Accessed 2024-05-06

[26] Pintér, J.: Globally convergent methods for n -dimensional multiextremal optimization. Optimization **17**(2), 187–202 (1986) https://doi.org/10.1080/02331938608843118 . Accessed 2024-05-06

[27] Stripinis, L., Paulavičius, R.: Lipschitz-inspired HALRECT algorithm for derivative-free global optimization. Journal of Global Optimization **88**(1), 139–169 (2024) https://doi.org/10.1007/s10898-023-01296-7 . Accessed 2024-05-05

[28] Stripinis, L., Paulavičius, R.: An empirical study of various candidate selection and partitioning techniques in the DIRECT framework. arXiv. arXiv:2109.14912 [cs, math] (2022). http://arxiv.org/abs/2109.14912 Accessed 2024-03-22

[29] Zhao, Q., Yan, B., Hu, T., Chen, X., Duan, Q., Yang, J., Shi, Y.: AutoOptLib: Tailoring Metaheuristic Optimizers via Automated Algorithm Design. arXiv. arXiv:2303.06536 [cs] (2023). http://arxiv.org/abs/2303.06536 Accessed 2024-05-23

[30] Huyer, W., Neumaier, A.: Global Optimization by Multilevel Coordinate Search

[31] Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of a*. J. ACM **32**(3), 505–536 (1985) https://doi.org/10.1145/3828.3830

[32] Frohner, N., Gmys, J., Melab, N., Raidl, G.R., Talbi, E.-g.: Parallel Beam Search for Combinatorial Optimization (Extended Abstract). Proceedings of the International Symposium on Combinatorial Search **15**(1), 273–275 (2022) https://doi.org/10.1609/socs.v15i1.21783 . Accessed 2023-01-31

[33] Valenzano, R., Xie, F.: On the completeness of best-first search variants that use random exploration. Proceedings of the AAAI Conference on Artificial Intelligence **30**(1) (2016) https://doi.org/10.1609/aaai.v30i1.10081

[34] Morales-Castañeda, B., Zaldívar, D., Cuevas, E., Fausto, F., Rodríguez, A.: A better balance in metaheuristic algorithms: Does it exist? Swarm and Evolutionary Computation **54**, 100671 (2020) https://doi.org/10.1016/j.swevo.2020.100671

[35] Drmota, M., Tichy, R.F.: Sequences, Discrepancies and Applications. Springer, Berlin Heidelberg (1997). https://doi.org/10.1007/bfb0093404

[36] Talbi, E.-G.: Metaheuristics. Wiley, Hoboken (2009). https://doi.org/10.1002/9780470496916

[37] Strongin, R.G., Sergeyev, Y.D.: Global Optimization: Fractal Approach and Non-redundant Parallelism. GLOBAL OPTIMIZATION

[38] Evtushenko, Y., Posypkin, M., Sigal, I.: A framework for parallel large-scale global optimization. Computer Science - Research and Development **23**(3-4), 211–215 (2009) https://doi.org/10.1007/s00450-009-0083-7 . Accessed 2024-05-05

[39] Jian He, Verstak, A., Sosonkina, M., Watson, L.T.: Performance Modeling and Analysis of a Massively Parallel Direct—Part 2. The International Journal of High Performance Computing Applications **23**(1), 29–41 (2009) https://doi.org/10.1177/1094342008098463 . Accessed 2024-03-22

[40] Nakib, A., Souquet, L., Talbi, E.-G.: Parallel fractal decomposition based algorithm for big continuous optimization problems. Journal of Parallel and Distributed Computing **133**, 297–306 (2019) https://doi.org/10.1016/j.jpdc.2018.06.002 . Accessed 2022-03-01

[41] Gablonsky, J.M., Kelley, C.T.: A Locally-Biased form of the DIRECT Algorithm, 12 https://doi.org/10.1023/A:1017930332101

[42] Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The quickhull algorithm for convex hulls. ACM Trans. Math. Softw. **22**(4), 469–483 (1996) https://doi.org/10.1145/235815.235821

[43] Fortune, S.: A sweepline algorithm for Voronoi diagrams, 22 https://doi.org/10.1007/BF01840357

[44] Watson, D.F.: Computing the n-dimensional

Delaunay tessellation with application to Voronoi polytopes*. The Computer Journal **24**(2), 167–172 (1981) https://doi.org/10.1093/comjnl/24.2.167 https://academic.oup.com/comjnl/article-pdf/24/2/167/967258/240167.pdf

[45] Rushdi, A.A., Swiler, L.P., Phipps, E.T., D'Elia, M., Ebeida, M.S.: VPS: VORONOI PIECEWISE SURROGATE MODELS FOR HIGH-DIMENSIONAL DATA FITTING. International Journal for Uncertainty Quantification **7**(1), 1–21 (2017) https://doi.org/10.1615/Int.J.UncertaintyQuantification.2016018697 . Accessed 2022-01-27

[46] Mitchell, S.A., Ebeida, M.S., Awad, M.A., Park, C., Patney, A., Rushdi, A.A., Swiler, L.P., Manocha, D., Wei, L.-Y.: Spoke-Darts for High-Dimensional Blue-Noise Sampling. ACM Transactions on Graphics **37**(2), 1–20 (2018) https://doi.org/10.1145/3194657 . Accessed 2022-01-28

[47] Villagran, A., Huerta, G., Vannucci, M., Jackson, C.S., Nosedal, A.: Non-parametric Sampling Approximation via Voronoi Tessellations. Communications in Statistics - Simulation and Computation **45**(2), 717–736 (2016) https://doi.org/10.1080/03610918.2013.870798 . Accessed 2022-01-28

[48] Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V.: Generating All Vertices of a Polyhedron Is Hard. Discrete & Computational Geometry **39**(1-3), 174–190 (2008) https://doi.org/10.1007/s00454-008-9050-5 . Accessed 2022-03-01

[49] Even, S.: Graph Algorithms, 2nd edn. Cambridge University Press, Cambridge (2011). https://doi.org/10.1017/CBO9781139015165

[50] Imai, T., Kishimoto, A.: A novel technique for avoiding plateaus of greedy best-first search in satisficing planning., vol. 2 (2011). https://doi.org/10.1609/socs.v2i1.18208

[51] Morrison, D., Sauppe, J., Zhang, W., Jacobson, S., Sewell, E.: Cyclic best first search: Using contours to guide branch-and-bound algorithms. Naval Research Logistics Quarterly **64**(1), 64–82 (2017) https://doi.org/10.1002/nav.21732

[52] Voelker, A., Gosmann, J., Stewart, T.: Efficiently sampling vectors and coordinates from the n-sphere and n-ball (2017) https://doi.org/10.13140/RG.2.2.15829.01767/1

[53] Muller, M.E.: A note on a method for generating points uniformly on n-dimensional spheres. Commun. ACM **2**(4), 19–20 (1959) https://doi.org/10.1145/377939.377946

[54] Harman, R., Lacko, V.: On decompositional algorithms for uniform sampling from n-spheres and n-balls. Journal of Multivariate Analysis **101**(10), 2297–2304 (2010) https://doi.org/10.1016/j.jmva.2010.06.002

[55] Ge, C., Ma, F.: A fast and practical method to estimate volumes of convex polytopes. In: Wang, J., Yap, C. (eds.) Frontiers in Algorithmics, pp. 52–65. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19647-3_6

[56] Corte, M.V., Montiel, L.V.: Novel matrix hit and run for sampling polytopes and its GPU implementation. Computational Statistics (2023) https://doi.org/10.1007/s00180-023-01411-y

[57] Chen, Y., Dwivedi, R., Wainwright, M.J., Yu, B.: Fast mcmc sampling algorithms on polytopes. J. Mach. Learn. Res. **19**(1), 2146–2231 (2018) https://doi.org/10.5555/3291125.3309617

[58] Finkel, D.E.: Direct optimization algorithm user guide. (2003). https://api.semanticscholar.org/CorpusID:6899005

[59] Derbel, B., Preux, P.: Simultaneous optimistic optimization on the noiseless BBOB testbed. In: 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 2010–2017. IEEE, Sendai, Japan (2015). https://doi.org/10.1109/CEC.2015.7257132 . http:

//ieeexplore.ieee.org/document/7257132/ Accessed 2024-05-22

[60] Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., Brockhoff, D.: COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. Optimization Methods and Software **36**(1), 114–144 (2021) https://doi.org/10.1080/10556788.2020.1808977 . arXiv:1603.08785 [cs, math, stat]. Accessed 2024-04-09

[61] Balouek, D., Carpen-Amarie, A., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lebre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Pérez, C., Quesnel, F., Rohr, C., Sarzyniec, L.: Adding Virtualization Capabilities to Grid'5000