A fractal-based decomposition framework for continuous optimization

Thomas Firmin, El-Ghazali Talbi

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 which defines 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 show the significance of each search component of the fractal framework, and the necessity to build harder and well-defined benchmarks which can assess the performance of deterministic, axis-aligned and symmetrical decomposition-based algorithms.

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 $f : \mathcal{S} \subset \mathbb{R}^n \rightarrow \mathbb{R}$:

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

where $\hat{x}$ is the global optima, f the objective function, and $\mathcal{S}$ a compact set made of inequalities (e.g. upper and lower bounds of the search space).

In the past two decades, several optimization algorithms have been proposed to tackle continuous optimization problems. Population-based algorithms generally rely on evolutionary algorithms (e.g. differential evolution, evolution strategies) and swarm intelligence (e.g. particle swarm optimization) [1]. Local search strategies are based on a single solution improvement (e.g. gradient-based algorithms, simulated annealing). Most of the state-of-the-art optimization algorithms are stochastic and require a high number of evaluations to be efficient. For optimization problems

with expensive objective functions, surrogate-based optimization (e.g. Bayesian optimization) can be an alternative. 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].

In this paper, we investigate a particular class of optimization algorithms, which could be classified as *divide-and-conquer* strategies based on hierarchical decomposition of the search space. This class of algorithms can overcome the dimensionality problem by creating flexible, scalable and massively parallel decomposition-based algorithms for high dimensional optimization problems.

This work has been inspired by two distinct families of decomposition-based optimization algorithms. The first one is based on algorithms derived from Lipschitzian global optimization [3], such as DIRECT and its various extensions (e.g. eDIRECT, BIRECT, HD-DIRECT) [4]. The second one concerns metaheuristics based on fractal decomposition, such as FRACTOP [5] and FDA [6].

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*.

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.

A fractal is a never-ending pattern that can be used to generate any-scale fractal trees [7]. 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 decomposes the search space via a given *fractal geometrical object*. *Tree search* provides a dynamic and hierarchical fractal decomposition of the search space. A *scoring* search component allows to balance the exploration and exploitation of the search space by giving a fitness value to fractals. A generic software framework named *Zellij*[1] has been developed. It offers a unified programming paradigm for the instantiation of state-of-the-art fractal-based decomposition algorithms such as DIRECT (DIviding RECTangles), SOO (Simultaneous Optimistic Optimization) and FDA (Fractal Decomposition Algorithm). Moreover, one can also design new optimization algorithms according to the definition of new strategies for the search components.

The paper is organized as follows. In section 2, the *Zellij* generic algorithmic framework based on fractal decomposition is presented. The following sections (section 3 to 6) detail successively the search components of the algorithmic optimization framework: fractal geometrical object, tree search, scoring, exploration, and exploitation. In section 7, state-of-the art decomposition-based algorithms have been instantiated to the framework and extended by defining new strategies for their search components. Section 8 depicts the experimental setup for comparing five popular fractal-based decomposition algorithms and six new designed ones. Section 9 analyzes the obtained computational results on CEC2020 and SOCO2011 benchmarks. Finally, we summed up in

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

section 10, the framework, future works and new research opportunities for tackling high dimensional optimization problem.

2 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 $\mathcal{S}$ of dimension n as a bounded subset of a metric space:*

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

with $L, U \in \mathbb{R}^n$, the lower and upper bounds, such that $\forall i \in [1, ..., n]$, $l_i < u_i$ and $\forall x \in \mathcal{S}$, $\forall i \in [1, ..., n]$, $l_i \leq x_i \leq u_i$

In our framework, a *fractal* is a self-similar geometrical object which does not depend on any information besides the nature of its parent. 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. To generalize our approach to other geometrical objects than fractals, we can also consider geometrical objects that depend on additional knowledge. For instance, in the DIRECT algorithm, the decomposition process in smaller hyperrectangles depends on the sampling and the evaluation of their centers. Selecting a particular geometrical object (e.g. hypercube, hypersphere) has a major impact on the decomposition process. Five properties characterize a fractal: partition size, building complexity, coverage, overlap, and memory complexity (see Table 1 and Figure 5). These properties are discussed in Section 3.

The decomposition of the search space can be defined by a *k-ary rooted partition tree*. The root represents the initial complete search space, 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* [8], *Beam Search* [9] or *Epsilon Greedy Search* [10].

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 [11]. 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 [3] and SOO [12], only the center of the hyperrectangle is 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* could be done in a passive way (e.g. Markov chain Monte Carlo (MCMC) sampling, low discrepancy sequences [13]) or in an active way (e.g. metaheuristics [14], surrogate-based optimization [2]). 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 [14].

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* [6][15], or *belief* [5].

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 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 DIRECT implementing different search components are discussed in Section 7.

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

- **Generalization**: our goal is to build a framework which unifies and generalizes various popular fractal decomposition-based 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 decomposition-based approach.
- **Massively parallel**: a transparent and efficient parallel implementation of the algorithms on various architectures (e.g. multicores, GPUs) is carried out. The main challenge is the parallelization of the tree search component of the framework. Many parallel tree search algorithms can be considered.

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

3 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.

Definition 2 (k-partition)**.** *A k-partition of a subset $\alpha \subseteq \mathcal{S}$ can be defined by the union of k non-empty and disjoint subspaces. These subsets can be obtained by a fractal building function $F : \alpha \rightarrow \{S_i\}_{i\in[1,...k]}$, where $S_i \subset \alpha$, and:*

$$\alpha = \bigcup F(\alpha) = \bigcup_{i=1}^{k} S_i \tag{3}$$

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 partial partitions.

Using a given fractal, one can build a recursive decomposition of $\mathcal{S}$ or a decomposition of a subset of $\mathcal{S}$. 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 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 3 (Hierarchical k-refinement)**.** *Let D be the number of successive refinements of a subset $S_{l,j,i} \subseteq \mathcal{S}$. We write $j \in [1, ..., E_{l-1}]$, 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:*

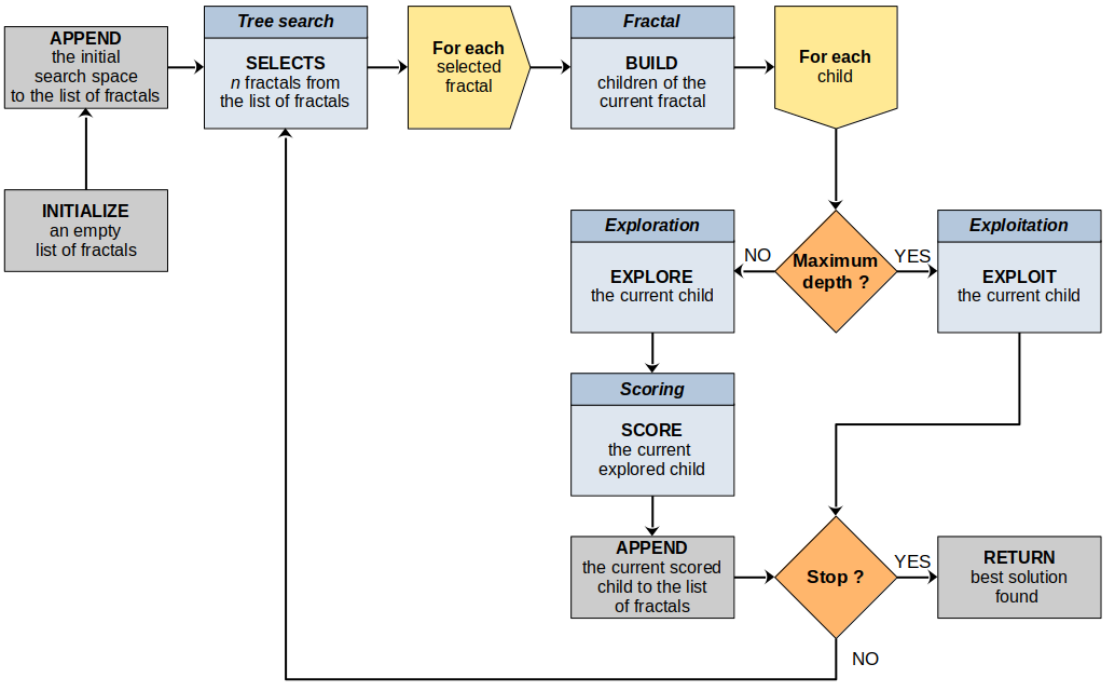


Fig. 1: The workflow of the Zellij framework. In blue, the five search components. In orange, the tests made at each iteration.

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

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 3 is a k-ary rooted tree. Hence, one can rewrite the initial search space $\mathcal{S}$ as the root of this tree: $S_{(1,0,1)} = \mathcal{S}$, 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 P_l of supersets (parents) at level l that have children

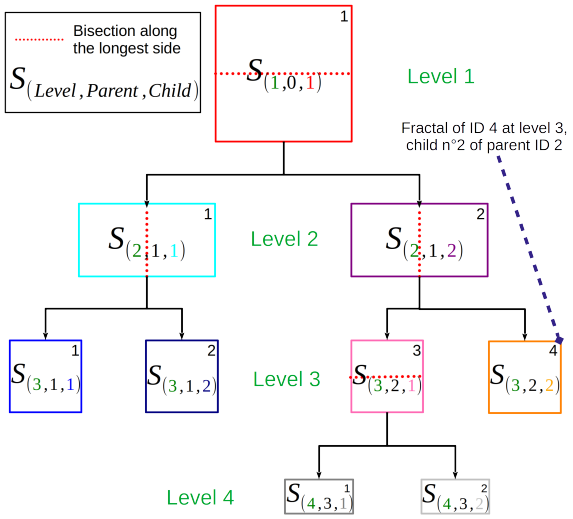


Fig. 2: Example of a 2-refinement of depth 4, with a bisection along the longest side.

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 $\mathcal{S}$ is that the initial search space will be equal to the union of all the leaves.

Theorem 1. *Let $\mathcal{S}$ 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 3, is equal to $\mathcal{S}$:*

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

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

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

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 $\{\circ[d-1,j,i], P_{d-1}\}$.

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

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

$\mathcal{S} = P_1 = \bigcup_{i=1}^{k} \circ[2,1,i] \bigcup P_2$ □

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 [6]. Indeed, fractals can overlap and do not necessarily cover $\mathcal{S}$. 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 maximized 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 $\mathcal{S}$. In our algorithmic framework, we consider the intersection between fractals and $\mathcal{S}$. A fractal, or a part of a fractal, cannot be outside the initial search space $\mathcal{S}$. 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 partial partitions, we have to measure what the overlap and coverage are for fractal-based decomposition algorithms. For a tree $\mathcal{T}^{(d)}$, let $\Sigma = \mathcal{P}(\mathcal{S})$ be a σ-algebra of $\mathcal{S}$. So $(\mathcal{S}, \Sigma)$ is a measurable set, where $\mathcal{L}_{\mathcal{T}^{(d)}} \subseteq \Sigma$ and $\mathcal{L}_{\mathcal{T}^{(d)}} \in \Sigma$. We can 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 $\mathcal{S}$ by $\mathcal{L}_{\mathcal{T}^{(d)}}$ and their overlap.

Definition 4 (Coverage)**.** *Let $(\mathcal{S}, \Sigma, \mu)$ a measure space, and $A, B \subseteq \mathcal{S}$, $A, B \in \Sigma$ 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: $C(\mathcal{S}, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}})$ 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 5 (Overlap)**.** *Let $(\mathcal{S}, \Sigma, \mu)$ 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 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 $\mathcal{S}$.

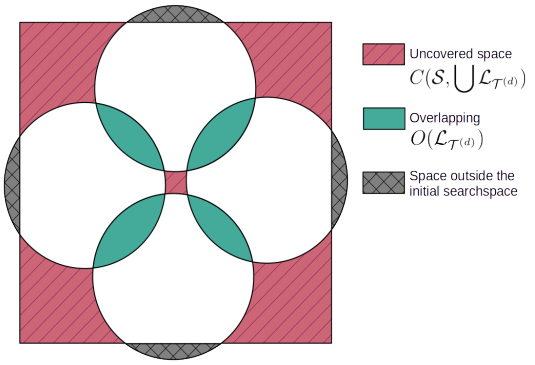


Fig. 3: 2-dimensional illustration of covering and overlap applied on a FDA scheme.

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 $\mathcal{S}$ by only looking at $C(\mathcal{S}, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}})$. Thus, several assumptions are made:

- For continuous dimensions, we consider that $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}$ covers $\mathcal{S}$ if $C(\mathcal{S}, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) = 0$, even if $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}$ does not include $\mathcal{S}$ boundaries.
- By construction, we consider that F, the decomposition function, produces fractals that are strictly smaller than the parent: $\forall A_i \in F(A), \mu(A_i) < \mu(A)$

Hence, we can modify Theorem 1 for overlap and partial k-partition.

Theorem 2. *Let $\mathcal{S}$ be a search space, $\Sigma = \mathcal{P}(\mathcal{S})$ a σ-algebra on $\mathcal{S}$, and $(\mathcal{S}, \Sigma, \mu)$ a measure space. $\mathcal{S}$ is the root of a k-ary partition tree $\mathcal{T}^{(d)}$ of depth d and $\mathcal{L}_{\mathcal{T}^{(d)}}$ its leaves. $C(.,.)$ is a measure of the coverage according to Definition 4. 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 $C(\mathcal{S}, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) = 0$. Then,*

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

2. ***Low-sacrifice based:*** *if $C(\mathcal{S}, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > 0$ and $C(\mathcal{S}, \bigcup F(\mathcal{S})) > 0$ and if $\forall l > 1$, $C(S_{l,j,i}, \bigcup F(S_{l,j,i})) = 0$. We have,*

$$\mu(\mathcal{S}) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) \tag{10}$$

with $\forall d > 1$, $\mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) \leq \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}\right)$

3. ***High-sacrifice based:*** *if $C(\mathcal{S}, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > 0$ and $C(\mathcal{S}, \bigcup F(\mathcal{S})) > 0$ and if $\forall l > 1$, $C(S_{l,j,i}, \bigcup F(S_{l,j,i})) > 0$ and $F(S_{l,j,i})/S_{l,j,i} \neq \emptyset$. Then, $\forall d > 1$, we have,*

$$\mu(\mathcal{S}) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}\right) \tag{11}$$

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 decomposition function F defined by Equation 3.

Preservative:
If $\forall d > 1$, $C(\mathcal{S}, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) = 0$.
Then $\mathcal{S} = \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}$, because by construction $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \subseteq \mathcal{S}$
and $C(A, B) = 0 \iff \mu(A \setminus (A \bigcap B)) = 0 \iff A = B$.
So, we have:

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

Low-sacrifice based:
If $\forall d > 1$, $C(\mathcal{S}, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > 0$ and $C(S_{d,j,i}, \bigcup F(S_{d,j,i})) = 0$.
Then, $\mathcal{S} \supset \bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \implies \mu(\mathcal{S}) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right)$
and $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \subseteq \bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}$.
So we have,

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

High-sacrifice based:
If $\forall d > 1, C(\mathcal{S}, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}}) > 0, C(S_{d,j,i}, F(S_{d,j,i})) > 0$ and $F(A) \setminus A = \emptyset$ (Subsets cannot have solutions outside their parents).
Then, $\mathcal{S} \supset \bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \implies \mu(\mathcal{S}) > \mu\left(\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}\right)$
and $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}} \supset \bigcup \mathcal{L}_{\mathcal{T}^{(d+1)}}$.

So we have,

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

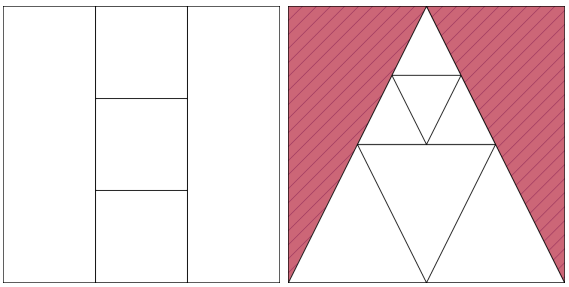
□

This theorem describes behaviors of a partial, k-refinement where leaves might not fully cover the initial search space. Preservative algorithms describe a partition that covers $\mathcal{S}$ no matter whether there is overlap between leaves or not, as we consider $\bigcup \mathcal{L}_{\mathcal{T}^{(d)}}$.

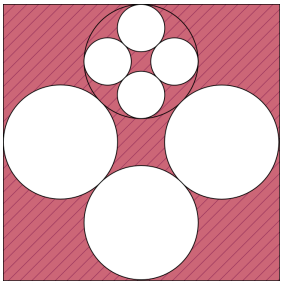
However, low-sacrifice based algorithms describe leaves of the tree that do not fully cover $\mathcal{S}$, but where children of the fractal cover its parent. In this case, one accepts to lose a part of the search space at the first refinement of $\mathcal{S}$. Then future refinements should not sacrifice solutions anymore. Moreover, sometimes the refinement of a fractal can result in children, which when unified form a bigger object than the parent. This is the case for fractals that can cover solutions outside the parent space.

According to Theorem 2, let us consider an initial search space $\mathcal{S}$ defined by a hypercube. In FDA [6], 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 a k-refinement using hyperspheres. To overcome this, the exploitation phase of FDA can ignore fractals' borders, so it can reach uncovered solutions. The algorithm is based on a high sacrifice rate when such a sacrifice happens at each refinement. Hence, some solutions are lost at each refinement. So, the deeper is the tree, the fewer solutions are reachable by the algorithm.

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. One should be aware of this property, and determine if it is acceptable to lose some solutions to carefully choose the right decomposition function F.



(a) Preservative fractals (b) Low sacrifice fractals



(c) High sacrifice fractals

Fig. 4: 2-dimensional illustrations of the three levels of sacrifice of the fractal search component. The hatched-magenta area corresponds to uncovered space resulting from successive decomposition: $C(S_{l,j,i}, \bigcup F(S_{l,j,i}))$

These three levels of sacrifice are described in Figure 4. Preservative fractals in Figure 4a, based on SOO in this example, preserve the initial search space $\mathcal{S}$, no space is lost. While low sacrifice fractals, on Figure 4b, here based on Sierpinski triangles, sacrifice space of $\mathcal{S}$ to use a geometry based on triangles. And finally, the example on Figure 4c, based on an FDA scheme, sacrifices space after each refinement.

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 [5] and simplices [16] 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 [17], Fortune's algorithm [18], Bowyer-Watson algorithm [19]). 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 [20–24], and then one has to use stochastic methods to build an approximation, such as SpokeDart with hyperplane sampling [22]. 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 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 another 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 [3][5][6].

Furthermore, if we look at the coverage and overlap properties, one can clearly see the major drawback of hyperspheres compared to other hypervolumes. $C(\mathcal{S}, \mathcal{L}_{\mathcal{T}^{(d)}})$ 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 [6], 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.

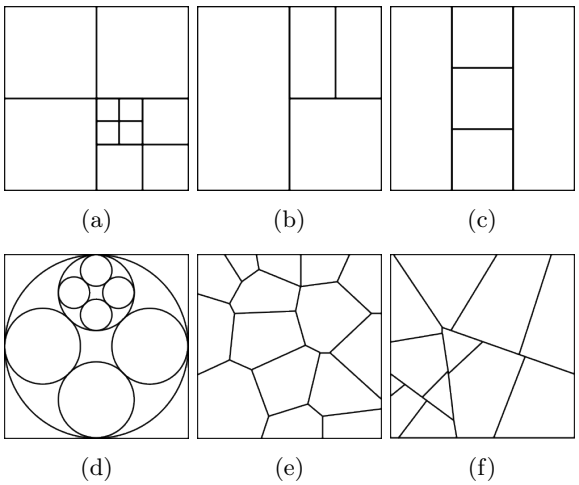


Fig. 5: Various examples of fractals in 2 dimensions: (a) hypercubes, (b) bisections, (c) trisections, (d) hyperspheres, (e) dynamic Voronoï, (f) fixed Voronoï.

According to Figure 5, 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.

4 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 $s = |\mathcal{L}_{\mathcal{T}^{(d)}}|$:

$$\tau : \mathcal{L}_{\mathcal{T}^{(d)}}, \mathbb{R}^s \rightarrow \{e1, ..., e_Q\} \tag{12}$$

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.

One can instantiate tree search algorithms using an `OPEN` and `CLOSED` lists [8]. 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.

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 $\mathcal{S}$	$C(\mathcal{S}, \bigcup \mathcal{L}_{\mathcal{T}^{(d)}})$	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	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	$\emptyset$	[5]	[12]	[25]	[6]	[15, 20]	[16]

[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.

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 *Potentially Optimal Rectangle* (POR), and many variations of DIRECT are based on this selection strategy [26][27]. 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* (BFS) 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 seen as a greedy *exploitation only*, whereas Breadth First Search (BFS) can be qualified as a greedy *exploration only* algorithm [28]. Both strategies are uneffective, as the scores given to fractals do not impact the selection. We lose the notion of hierarchy between fractals.

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 BFS, is the Best First Search [8]. Some of these algorithms are stochastic (e.g. Epsilon Greedy Search, Diverse Best First Search [29]) or, others, allow having the hand on the exploration-exploitation tradeoff (e.g. Cyclic best First Search [30]). 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 [9], 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. BFS

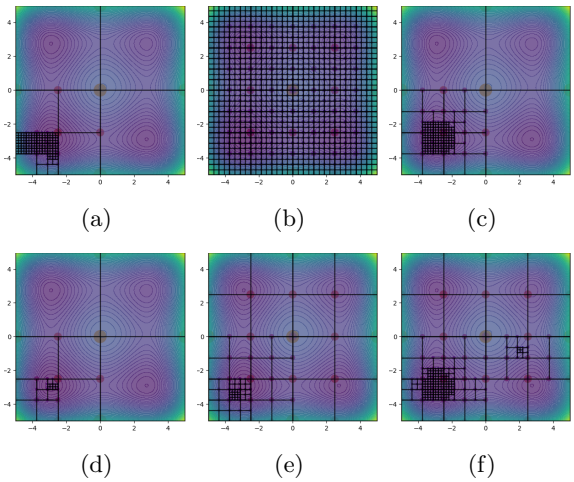


Fig. 6: Fractal decomposition with hypercubes applied on 2D Styblinski-Tang function with various tree search: (a) Depth First Search, (b) Breadth First Search, (c) Best First Search, (d) Beam Search, (e) Cyclic Best First Search, (f) Epsilon Greedy Search.

explored the entirety of each level before tackling the next one, no matter the hierarchy. Whereas DFS is focused on the deepest fractal.

5 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:

$$\gamma : \circ[l, j, i], P, f|_{\circ[l,j,i]}(P) \to \mathbb{R} \tag{13}$$

The scoring method γ takes a leaf $\circ[l, j, i]$, a sample of solutions P restricted to this leaf, and their corresponding objective values $f|_{\circ[l,j,i]}(P)$. It returns a score within $\mathbb{R}$ which defines the quality of $\circ[l, j, i]$.

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.

We have noticed that 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 [26]. 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.

6 Exploration and exploitation strategies

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

$$\begin{aligned} &Explore : \alpha \to P, f|_\alpha(P) \\ &Exploit : \alpha \to P, f|_\alpha(P) \end{aligned} \tag{14}$$

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 upper and lower bounds. For example, low discrepancy sequences such as Sobol, Halton, and Kronecker methods can be used [13]. Sampling in a hypersphere requires a few tricks to satisfy the equation of a n-ball. The Box-Muller method can be a solution [31–33]. However, sampling inside a Voronoï cell (i.e. polytope) is a complex procedure. One could use methods to approximate the Lebesgue measure [34], hit-and-run sampling [35], hyperplanes sampling [22], or MCMC sampling [36]. 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.

7 Extension of popular decomposition-based algorithms

This section instantiates some popular fractal-based optimization algorithms to our algorithmic framework. Moreover, it shows how one can extend these algorithms using various search strategies for the different components.

7.1 FRACTOP, FDA and PolyFRAC: Fractal-based optimization algorithms

FRACTOP is one of the first algorithms based on fractal decomposition [5]. It uses hypercubes to decompose the search space, a genetic algorithm to explore each fractal, and simulated annealing to exploit a promising leaf fractal. The algorithm implements a fuzzy measure, called *belief*, as a scoring method. One major drawback of this algorithm is its poor scalability in high dimension (see Table 1).

The FDA algorithm partly solves the *curse of dimensionality* problem of FRACTOP [6]. Instead of a hypercubes-based decomposition, it uses

Algorithm 1 Fractal-based decomposition algorithm

Inputs:
$\mathcal{S}$ *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 $\leftarrow \mathcal{S}$ *List of non-expanded fractals*
CLOSED $\leftarrow [\cdot]$ *List of expanded fractals*
current $\leftarrow \mathcal{S}$
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**
        P, values $\leftarrow$ *Explore*(child)
        score $\leftarrow \gamma$(child, P, 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 $\hat{x}$

hyperspheres. By using such fractals, the decomposition has a lower complexity, but at the cost of overlapping fractals due to an inflation ratio (see Table 1). This ratio partially reduces the lack of space coverage implied by hyperspheres decomposition. The exploration component, called *Promising Hypersphere Selection* (PHS), computes three points: the center of the hypersphere and two opposite points equidistant to the center. The heuristic, used to score a fractal, is the *distance-to-the-best* solution found so far. Finally, a leaf at

the maximum depth level of the tree, is exploited with an *Intensive Local Search* (ILS), which is a coordinate descent algorithm with adaptive step size.

The polyFRAC algorithm is a modification of FDA [15]. Rather than using hyperspheres, polyFRAC takes advantage of H-polytope fractals. Since, finding the vertices (i.e. n-faces) of a H-polytope is a complex procedure, thus polyFRAC approximates them. The exploration, exploitation, and scoring strategies are similar to those of FDA.

Thanks to our generic and flexible framework, we have extended and improved those algorithms by replacing the tree search component of FDA (Move-up procedure), by a beam search[2], we called this extension FDABs. We also tested FDA and FDABs using deep trees[3], we named them FDAD and FDADBs. Additionally, we replaced the distance-to-the-best (DTTB) scoring: $\frac{f(x)}{\|x-Best\|}$ by a centered version of it (C-DTTB). The measure is centered on the best solution found so far, $\frac{f(x)-f(Best)}{\|x-Best\|}$, where x is a sampled point and $Best$ the best solution found so far. As we select the highest ratio, on the first version, the ratio associated to a better solution can be considered as non-promising if the point is far from $Best$, whereas a bad solution can be considered as good if it is close to $Best$. We named this version, FDAC.

7.2 Direct: Dividing Rectangles

The DIRECT algorithm is initially a modification of the Schubert's algorithm [37]. It assumes that the objective function is Lipschitz-continuous, with a positive constant K:

$$|f(x) - f(y)| \leq K\|x - y\|, \quad \forall x, y \in \mathcal{S}$$

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

However, Lipschitzian optimization has several drawbacks. The constant K must be known; there is a poor balance between exploration and exploitation, and the computation complexity is subject to the *curse of dimensionality*. The algorithm computes a solution at the center of a hyperrectangle before decomposing. At each iteration, the algorithm uses a series of trisections on the longest sides to subdivide the search space into smaller hyperrectangles. DIRECT introduces the concept of *Potentially Optimal Rectangle* (POR), which is a selection criterion that determines if a fractal is promising [3]. Many extended versions of DIRECT have been proposed in the literature. These can be considered within our five search components:

- Fractal component: BIRECT (bisection) [25], eDIRECT (Voronoï cell) [20], DISIMPL (simplices) [16].
- Tree search component: Pareto-Lipschitzian optimization [38], DIRECT Restart [27].
- Exploitation component: DIRMIN [39].

These modifications try to overcome some DIRECT drawbacks, such as lower performances in high dimensions, low convergence rate when trapped by local optima, and a lack of a local optimizer. In this paper, we introduce another extension of DIRECT, called DIRECTBs, where we replace the POR tree search, by a *Beam search*. Furthermore, we have implemented under our framework two well-known versions of DIRECT, which are locally biased DIRECT (DIRECTL) [26], and DIRECT Restart (DIRECTR) [27].

7.3 SOO: Simultaneous 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 $\hat{x}$ [12] :

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

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

[2]The beam length was set to 3000

[3]The maximum depth was set to 5 for FDA and FDABs, and to 10 for FDAD and FDADBs

partition tree, the best fractal is selected according to the evaluation of a representative solution inside it (e.g. center). Here, the balance between exploration and exploitation relies on the tree search algorithm. In addition, a stochastic version called Sto-SOO has been designed for noisy loss function, where each fractal has to be evaluated multiple times [40]. In the same way as DIRECTBs, we replaced the SOO tree search algorithm, by a *Beam Search* component, denoted by SOOBs algorithm.

8 Experimental setup

8.1 Algorithms selection

We have experimented more than 20 extensions of the presented popular algorithms. We tried different tree search algorithms, such as *Best First Search* and *Cyclic Best First Search*, different beam lengths for *Beam Search*, different maximum tree depth and scoring methods. In this paper, we have selected the most relevant ones. We focus on deterministic fractal-based decomposition algorithms. Hence, only one run on each function is necessary to evaluate their performances. We have excluded some popular stochastic algorithms such as FRACTOP, Sto-SOO and some versions of DIRECT, such as eDIRECT [20] or glcCluster [41]. Even that some designed algorithms have been parallelized, a focus is made on a sequential version of the algorithms. Indeed, some parallel algorithms may have different behaviors in terms of convergence and search time. For a fair comparison, all the studied algorithms have been implemented under the *Zellij* framework.

The first aim of this comparison is to show that we can instantiate 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. Thus, we have instantiated FDA [6], DIRECT [3], DIRECTL [26], DIRECTR [27] and SOO [12]. We resumed all 11 implemented algorithms and their different search components in Table 2. An unsought property of our selection is that all algorithms sample points in an axis-aligned fashion. This is a behavior that we must keep in mind when selecting benchmark functions.

8.2 Benchmark selection

For the selection of benchmarks, five essential properties have been considered. Functions must be shifted or rotated, or at least not have their global optimum located at the center of the search space. Indeed, for some well-known benchmark functions (e.g. Rastrigin, Ackley, Sphere), some selected algorithms (e.g. DIRECT, FDA, SOO) can directly place a point, or the center of a fractal onto the global optimum. Moreover, some benchmark functions must be multimodal, so to trap the algorithms into local optima, and thus evaluate their exploration capability. The benchmark functions must be non-separable, meaning that all dimensions have to be optimized to find the global optimum. This prevents algorithms from taking advantage of their axis align property to optimize only one or a few dimensions. Moreover, functions must be non-symmetrical, as some algorithms use symmetry to sample points. Therefore, we decided to select the CEC2020 benchmark [42]. We consider all functions as black box, and we reduce the maximum number of calls to the objective function at $5000n$, as in [6][15]. We also tried to reproduce the computational results obtained in [6], on the SOCO2011 single objective benchmark. However, we select the first six functions, as the others are not shifted nor rotated, and their global optimum are located on $(0, 0, ..., 0)$.

In our results' analysis, we use the two-sided Wilcoxon signed-rank test and corresponding mean ranks for each of the 11 algorithms. We performed this test on each benchmark. Thus, we were able to see their reliability. We take an error rate $\alpha = 0.05$ for the statistical test. Tested dimensions are 10, 15, 20, 30, 50, 100, which are the dimensions that both benchmarks have in common. For the Wilcoxon test, the two hypotheses are:

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

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	[5]
FDA	n-sphere	Sorted DFS	PHS	ILS	DTTB	5	[6]
DIRECT	Trisection	All POR	Center	$\emptyset$	$\emptyset$	600 [a]	[3]
DIRECTL	Trisection	1 POR per level	Center	$\emptyset$	$\emptyset$	600 [a]	[26]
DIRECTR	Trisection	Adaptive POR	Center	$\emptyset$	$\emptyset$	600 [a]	[27]
SOO	Trisection	Best fractal at each level	Center	$\emptyset$	$\emptyset$	600	[12]
FDABs	n-sphere	Beam Search [b]	PHS	ILS	DTTB	5	This work
DIRECTBs	Trisection	Beam Search[b]	Center	$\emptyset$	$\emptyset$	600 [a]	This work
SOOBs	Trisection	Beam Search[b]	Center	$\emptyset$	$\emptyset$	600	This work
FDAC	n-sphere	Sorted DFS	PHS	ILS	C-DTTB	5	This work
FDAD	n-sphere	Sorted DFS	PHS	ILS	DTTB	10	[6]
FDADBs	n-sphere	Beam Search[b]	PHS	ILS	DTTB	10	This work

[a]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 a lower and an upper bound is inferior to a value ϵ set to $1e-13$, the fractal cannot be decomposed anymore.

[b]The beam length was set to 600.

9 Results analysis and discussion

First, all results are based on the gaps between the best solution found by each algorithm and the known global optimum of each function. Figure 7 shows the mean of the ranking of all 11 algorithms on all CEC2020 benchmark functions 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 does not mean that the first one is significantly better. We need a corresponding statistical test and p-value for each algorithm. We resumed the comparison between ranks and p-values in Figure.8. 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

Table 3: Functions of the CEC2020 benchmark[42]

#	Function	Shifted	Rotated	Unimodality	Separability	Symmetrical
1	Bent $f1$	Yes	Yes	Yes	No	Yes
2	Schwefel	Yes	Yes	No	No	No
3	Lunacek Bi-Rastrigin	Yes	Yes	No	No	No
4	Rosenbrock + Griewangk	No	No	Yes	No	Yes
5	Hybrid 1	No	Yes	No	No	No
6	Hybrid 2	No	Yes	No	No	No
7	Hybrid 3	No	Yes	No	No	No
8	Composition 1	Yes	Yes	No	No	No
9	Composition 2	Yes	Yes	No	No	No
10	Composition 3	Yes	Yes	No	No	No

Table 4: Functions of the SOCCO2011 benchmark[6]

#	Function	Shifted	Rotated	Unimodality	Separable	Symmetrical
1	Sphere	Yes	No	Yes	Yes	Yes
2	Schwefel Problem 2.21	Yes	No	Yes	Yes	Yes
3	Rosenbrock	Yes	No	Yes	No	Yes
4	Rastrigin	Yes	No	No	No	Yes
5	Griewangk	Yes	No	No	No	Yes
6	Ackley	Yes	No	No	No	Yes
7	Schwefel Problem 2.22	No	No	Yes	No	Yes
8	Schwefel Problem 1.2	No	No	Yes	Yes	Yes
9	Bohachevsky	No	No	No	Yes	Yes
10	Schaffer	No	No	No	No	Yes

example, on CEC2020, for dimensions 10, if we focus on FDA (first column), we can compare it with SOO (fifth row). The color indicates that there is no statistical evidence that FDA is worse or better than SOO. In Figure 8, Figure 9 and Figure 10, 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.

The following analysis is based on the CEC2020, as it satisfies the properties we have previously defined.

9.1 Sensitivity of dimensionality

The initial observations reveal that for dimensions from 10 to 20, FDAD and FDADBs are the worst algorithms. Moreover, for all dimensions, we can see that these two algorithms are the worst among other FDA versions. This confirms what authors in [6] noticed, the deeper the tree, the worse the results. Which appears to be true even if we replace the tree search (FDADBs).

For dimensions from 10 to 50, it is not certain that FDA is better than other algorithms, except FDAD and FDADBs, which are always worse. But for dimensions 100, FDA is better than DIRECT, DIRECTL and DIRECTR, which was expected. Indeed, DIRECT does not scale well in dimensions. In dimension 50, one can note that FDA, DIRECT, DIRECTL, DIRECTR, SOO, DIRECTBs, FDABs are equivalent.

Concerning SOO, we can observe that it is unclear if it is working better than other algorithms besides FDAD and FDADBs. Because SOO must generate many fractals to significantly reduce the search space, SOOBs quickly fill the beam, which can explain a significant decrease in terms of performances as n increases. SOO becomes significantly less effective than FDA, DIRECTL, DIRECTBs, FDABs and FDAC in dimension $n = 100$.

In low dimensions, DIRECTBs does not perform better compared to other DIRECT variations. However, it scales better when dimension increases. Indeed for $n = 100$, DIRECTBs, DIRECTR and DIRECTL are better than the original DIRECT. It shows the effect of the tree search algorithm replacement. We can say that depending on which dimension DIRECT is applied, we should adapt some of its features to make it scale better.

Finally, the best algorithms for high dimensional continuous problems, are, FDA, FDAC, FDABs and DIRECTBs.

9.2 Sensitivity of tree search

The selected algorithms are sensitive to different parameters. For FDA, increasing D, the maximum tree depth, reduces drastically the performances. It is interesting to compare the maximum tree depth of each algorithm. DIRECT and SOO require a deeper tree to perform well, conversely to FDA.

However, in FDA, modifying *Move-up* by a *Beam search* does not have a significant impact. For DIRECTL we can see that modifying the tree search, *All POR* by *1 POR per level* and the measure of the size of a fractal (from σ_2 to σ_∞), makes DIRECTL better than DIRECT for higher dimensions. Nonetheless, it does not scale well. For SOO, the replacement of the tree search by a Beam Search, worsens it.

Finally, this work, empirically shows that by modifying some parts of *fractal decomposition-based* algorithms, we can obtain different behaviors and scalability. The most prominent example is the comparison between DIRECT, DIRECTL and DIRECTBs. DIRECT does not scale well for dimensions 100, while other versions can perform better.

9.3 Sensitivity of scoring

FDAC and FDA are similar for all dimensions. We noticed that the modification of the scoring

method did not have a significant impact on the algorithm's behaviors. It is not clear which part of the algorithm makes it work. We have changed the scoring and tree search components, but we did not notice any change in terms of efficacy, except for FDAD and FDADBs. The step size of ILS depends on the radius of the hypersphere in which it is applied, this can explain bad performances when we increase the maximum depth. Future works might focus on the exploitation phase, and its consequences on the performances of FDA.

9.4 Sensitivity of instances

The goal here, is to compare results obtained on the CEC2020 with the SOCO2011 benchmarks, in terms of sensitivity to the benchmark properties. The following analysis is based on Figure 9 and Figure 10.

With the 6 functions SOCO2011, the obtained results do not permit to clearly demonstrate which of FDA, FDAC, FDABs, DIRECT, DIRECTL, DIRECTR, DIRECTBs, SOO is the best. Even-if DIRECTBs seems to have better performances for $n = 100$ than the other DIRECT based versions. As for CEC2020, FDAD seems to be the worst algorithm for all dimensions, followed by FDADBs. Moreover, for low dimensions from 10 to 20, it is unclear, we cannot conclude that any of the algorithms performs better than the others (except FDAD and FDADBs). For dimensions 30 to 100, DIRECTBs, is better than all other DIRECT versions.

We even obtain different scalability. According to SOCO2011, the best algorithm to use in very high dimension ($n = 100$) is DIRECTBs, whereas CEC2020 shows that FDA, FDAC, FDABs and DIRECTBs are suitable solutions for this dimensionality.

Selecting all 10 functions of the SOCO2011 is not relevant, as $F7$ to $F10$ have their global optimum located at $(0, 0, ..., 0)$. But here we want to show that when selecting a benchmark we have to study behaviors of the selected algorithms and properties of the benchmark to avoid worthless comparisons.

We can notice that by adding 4 new functions that are obviously easy to solve as, DIRECT, DIRECTL, DIRECTR, SOO, DIRECTBs,

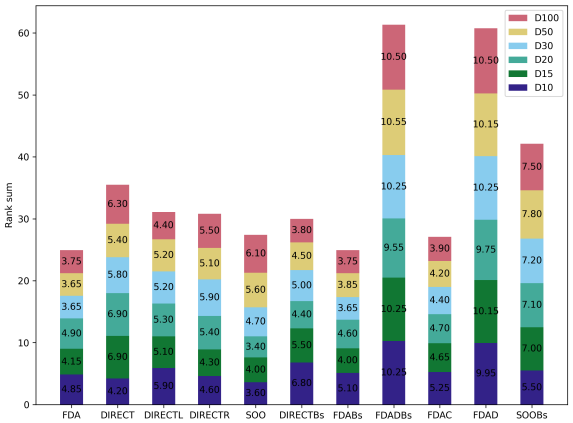


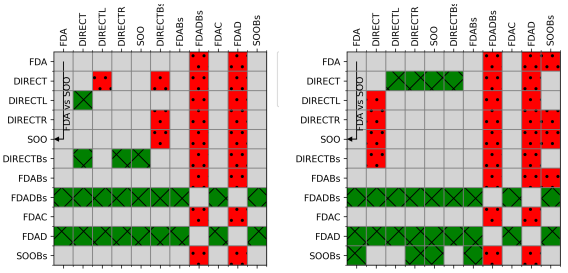
Fig. 7: Mean ranks on CEC2020 benchmark.

SOOBs find instantly the global optimum, does not impact the results. Except FDAC which becomes better than FDADBs and FDAD in dimensions $n = 20$, $n = 30$ and $n = 50$. By adding these worthless functions to the benchmark and seeing no significant difference, we can suppose that SOCO2011 is not sufficiently robust to assess performances of selected algorithms. Therefore, we must question the relevance of this benchmark to compare decomposition-based algorithms.

9.5 Real-world problem: Sharpe ratio maximization

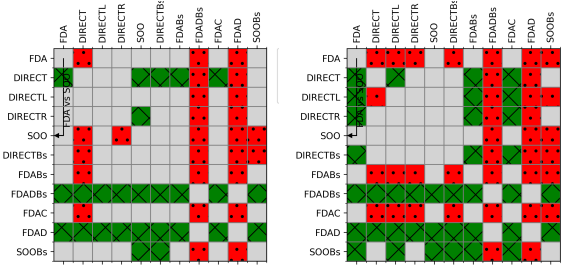
Portfolio optimization is a problem coming from the financial and investment management fields. It aims to find a portfolio of assets that offers the best possible trade-off between expected return and risk. The primary goal of portfolio optimization is to help investors make informed decisions about how to allocate their investments among different assets or securities. A portfolio is made of various assets such as bonds or real estate. These are defined by different levels of risks and returns. The expected return is the average return one can expect from a given portfolio over a specific period.

Let's consider w the weight vector of assets and μ the expected return for each one. Then the portfolio expected return is equal to $R_p = w^T \mu$. Weights w are subject to the following constraints: $\sum_{i=0}^{n} (w_i) = 1$ and $w_i \geq 0$. Here the goal is not to tackle constrained optimization, so we do



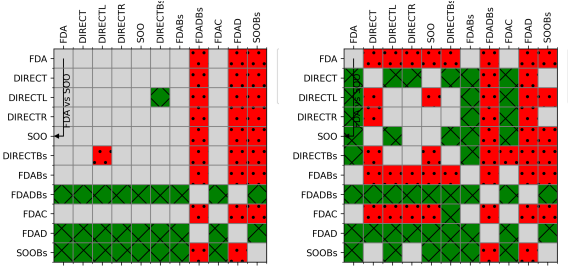
(a) $n = 10$

(b) $n = 15$



(c) $n = 20$

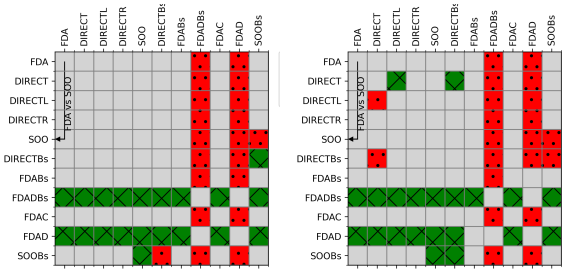
(d) $n = 30$



(e) $n = 50$

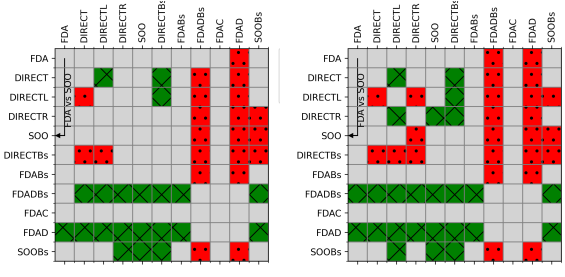
(f) $n = 100$

Fig. 8: Ranks and Pair Wise Wilcoxon test comparison on CEC2020. **Solid-Grey**: Statistically unsignificative ($\alpha > 0.05$). **Gridded-Green**: Better. **Dotted-Red**: Worse



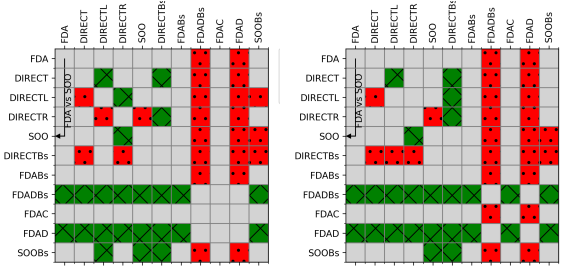
(a) $n = 10$

(b) $n = 15$



(c) $n = 20$

(d) $n = 30$



(e) $n = 50$

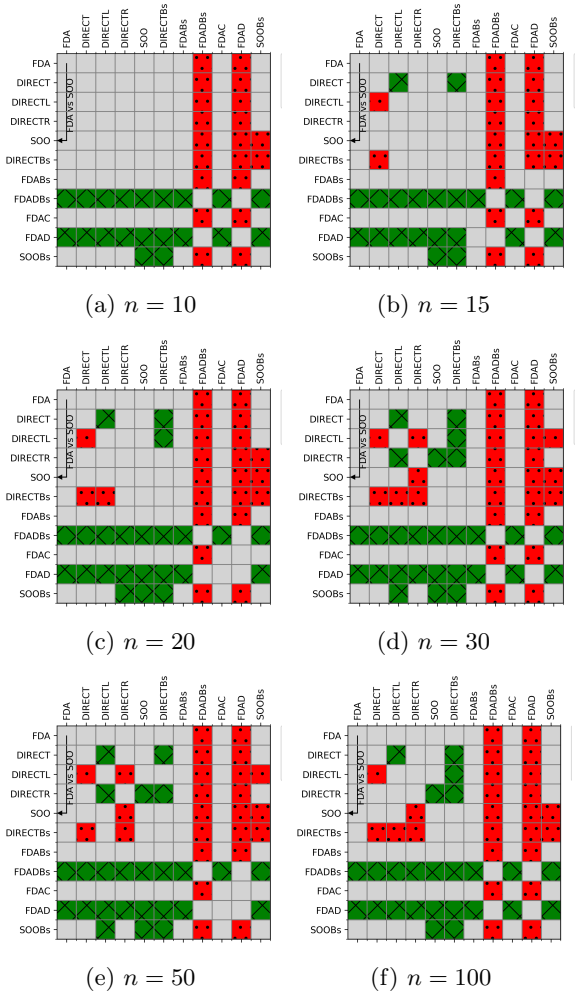
(f) $n = 100$

Fig. 9: Ranks and Pair Wise Wilcoxon test comparison on 6 functions SOCO2011. **Solid-Grey**: Statistically unsignificative ($\alpha > 0.05$). **Gridded-Green**: Better. **Dotted-Red**: Worse

not directly model $\sum_{i=0}^{n}(w_i) = 1$. However, when weights w_i are passed to the objective functions, these are normalized to follow the constraint. So the search space is a unit hypercube of dimension n, where n is the number of assets, and the bounds are $[0, 1]^n$.

One way to select the best portfolio, is to maximize the Sharpe ratio: $S_p = \frac{R_p - R_f}{\sigma_p}$ With, S_p the Sharpe ratio, R_p the expected return of a portfolio p, R_f the risk-free rate, which is the expected minimum return on a zero risk investment, and σ_p the standard error of p's expected return. A negative Sharpe ratio figures out a lower performance than a risk-free investment. A ratio within $[0, 1]$ defines a positive expected return where the risk taken is too high. When $S_p > 1$, then the expected return of the portfolio is positive under a reasonable risk.

So Sharpe ratio optimization is a high dimensional and scalable problem, where some dimensions have to be set at 0 (no investment for a given asset). The dataset contains stocks from the New York Stock Exchange and Nasdaq Stock Market (SP500) from January 7, 2000, to December 29, 2017 [43]. The base Sharpe ratio for comparison



(a) $n = 10$ (b) $n = 15$

(c) $n = 20$ (d) $n = 30$

(e) $n = 50$ (f) $n = 100$

Fig. 10: Ranks and Pair Wise Wilcoxon test comparison on 10 functions SOCO2011. **Solid-Grey**: Statistically unsignificative ($\alpha > 0.05$). **Gridded-Green**: Better. **Dotted-Red**: Worse

is given by the Expected Frontier algorithm, often used for this problem. Here $n = 356$ and $R_f = 2\%$

Results in Table 5 show that FDA-based algorithms can find the same optimum as the Efficient Frontier algorithm. Even FDAD and FDADBs were able to optimize this problem, whereas they were poorly ranked by the CEC2020 experiments. Other fractal-based decomposition algorithms are significantly less efficient. We can state that in very high dimension, here $n = 356$, a local optimizer appears to be needed.

Table 5: Portfolio optimization of SP500

Algorithm	Best Sharpe ratio
FDA	**6.14E+00**
DIRECT	2.45E+00
DIRECTL	2.64E+00
DIRECTR	3.78E+00
SOO	2.89E+00
DIRECTBs	3.94E+00
FDABs	**6.14E+00**
FDADBs	**6.14E+00**
FDAC	**6.14E+00**
FDAD	**6.14E+00**
SOOBs	2.90E+00
Efficient Frontier[a]	**6.14E+00**

[a]The problem is converted into a convex one within the Mean-Variance framework.

10 Conclusions and perspectives

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[4]. 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.

Relevant 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 the benchmark instances have also been analyzed. The proposed framework opens a door for developing and analyzing various search components. We need larger and well-defined benchmarks with more functions so that we can have a sufficiently

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

large sample to compute significant statistical analysis.

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 such as Voronoï fractals and Latin hypercubes, which implies stochasticity. We will have the opportunity to experiment the algorithms for noisy functions and multi-objective optimization problems.

From a long-term perspective, we will 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. Thus, we could figure out autonomous *fractal decomposition-based* algorithms dedicated to specific problems. In terms of application perspective, some computational experiments are under development on solving high-impact optimization problems such as hyperparameter 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 at https://doi.org/10.57745/0JEUEK. The code used for experiments is available on GitHub at https://github.com/ThomasFirmin/fdb_zellij_exp.

References

[1] 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

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

[3] 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

[4] 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

[5] Demirhan, M., Özdamar, L., Helvacıoğlu, L., Birbil, I.: 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

[6] 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

[7] 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

[8] 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

[9] 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

[10] 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

[11] 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

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

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

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

[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] 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

[17] 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

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

[19] 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

[20] 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

[21] 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

[22] 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

[23] Villagran, A., Huerta, G., Vannucci, M., Jackson, C.S., Nosedal, A.: Nonparametric 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

[24] 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

[25] 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

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

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

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

[29] 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

[30] 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

[31] 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

[32] 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

[33] 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

[34] 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

[35] 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

[36] 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

[37] 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

[38] 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

[39] 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

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

[41] Rios, L.M., Sahinidis, N.V.: Derivative-free optimization: a review of algorithms and comparison of software implementations. Journal of Global Optimization **56**(3), 1247–1293 (2013) https://doi.org/10.1007/s10898-012-9951-y . Accessed 2022-10-24

[42] Liang, J., Suganthan, P., Qu, B., Gong, D., Yue, C.: Problem Definitions and Evaluation Criteria for the CEC 2020 Special Session on Multimodal Multiobjective Optimization. https://doi.org/10.13140/RG.

2.2.31746.02247

[43] Man-Fai Leung: Datasets for Portfolio Optimization. Mendeley (2022). https://doi.org/10.17632/G5579MMC9K.2 . https://data.mendeley.com/datasets/g5579mmc9k/2