Universiteit Leiden

Master Computer Science

Dynamic Configuration of Operators and Parameters in Differential Evolution through Combined Fitness and Diversity-Driven Adaptation Methods

Name:	Rick Boks
Student ID:	1862979
Date:	July 5, 2021
Specialisation:	Artificial Intelligence
1st supervisor:	Anna V. Kononova
2nd supervisor:	Hao Wang

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The widespread success of metaheuristics can be partially credited to the developments in the realm of hyper-heuristics, where the automation of the otherwise manual, problem-specific tuning of a metaheuristic is studied. While some hyper-heuristics tune the algorithm prior to its application, they can also be leveraged to adapt the algorithm's behavior *during* the optimization process, in response to changes in the environment. The metaheuristic known as Differential Evolution (DE) has endured many modifications aiming to adaptively control its parameters, and more recently, to adaptively select the operators employed to generate new candidate solutions. Most such methods greedily select strategies that give the most rapid improvements with regard to the objective function, risking convergence on a local optimum. This thesis sets up a framework for intertwined adaptation of operators and parameters for population-based metaheuristics, where the adaptation process addresses the exploration/exploitation dilemma by explicitly encouraging exploratory behavior. In detail, we devise diversity-based metrics that capture the exploratory or exploitative tendency of a single individual, which can then be used to reward or penalize its associated strategy. Several methods for balancing the fitness- and diversity-based metrics are considered, as well as different methods that determine which operators are applied in future iterations. Through an extensive tuning process, seven adaptive DE algorithms are obtained, which are then benchmarked on BBOB/COCO to assess their empirical performances, and analyzed with regard to several aspects of the algorithmic behavior. Further, we perform an elaborate experiment to assess the optimal composition of the configuration space, which holds the operator configurations available to the hyper-heuristic. The results show the efficiency of the joint parameter and operator adaptation, driven by both fitness- and diversity-related metrics, making DE more versatile across different problem types. In addition, we provide practical recommendations regarding the composition of the configuration space.

Contents

1 Introduction

Metaheuristics are stochastic algorithms capable of finding (near-)optimal solutions to non-linear problems without any prior knowledge about the problem at hand. Although metaheuristics do not guarantee that the optimal solution is found, they have been widely successful in solving highly complex real-world problems [70, 61] in relatively little time, by intelligently sampling a subset of the search space, when exploring it entirely is simply not an option.

Despite the widespread success of this class of algorithms, there is still difficulty in selecting the correct metaheuristic for a particular (unseen) problem, and tuning its control parameters. Although there is no hope for finding the best algorithm for *all* optimization problems, as dictated by the No Free Lunch theorem [74], recent efforts in the field of hyper-heuristics have shown significant progress in algorithm selection and tuning [12], making it much simpler to find and configure an appropriate algorithm to apply to a new problem.

Instead of operating directly on the search space of the optimization problem's solutions, hyper-heuristics operate on a *space of (meta)heuristics*. One might wonder what the benefit of such hyper-heuristics is, as they, in turn, also introduce additional hyperparameters. Is it worth replacing a metaheuristic's parameters with those of the hyper-heuristic? Generally, a hyper-heuristic can provide two main benefits [47]:

- A hyper-heuristic acts as an abstraction layer *on top of* the metaheuristic, controlling it from a higher level. Generally, the result is that the parameters have a less direct impact on the performance of the algorithm, making it perform more consistently across different types of problems, but perhaps worse on a specific (type of) problem.
- The parameters of a hyper-heuristic are generally more comprehensible for a human. For example, a hyper-heuristic might have a parameter that directly controls the exploitation-exploration balance, which abstracts underlying parameters that require more technical knowledge about the algorithm, such as a mutation or crossover rate.

Two main types of hyper-heuristics can be discerned: offline and online approaches. Offline hyper-heuristics are concerned with tuning the algorithm on a certain problem or a class of problems, before actually applying it to the problem at hand. Although offline methods have been successful in the past (e.g. [37, 40, 43]), there is commonly a significant computational overhead associated with them. Further, once a tuned algorithm configuration is returned by the hyper-heuristic, it cannot alter the algorithm's strategy in real time, to react to changes in the environment during the various stages of the search.

Online approaches, on the other hand, tune the algorithm *during the search*, further reducing the total effort of applying an optimization algorithm to a problem, and allowing the hyper-heuristic to adapt its strategy throughout the search. Online methods have in particular shown their merit in the context of adaptive parameter control [16, 41], and, more recently, in Adaptive Operator Selection (AOS) [17], where the aim is to select variation operators of Evolutionary Algorithms (EAs) in an online fashion [45].

One EA, which gained wide popularity after its introduction by Storn and Price in 1997, is Differential Evolution (DE) [64]. The algorithm was originally proposed for single-objective optimization in continuous (real-valued) search spaces, but many variants have surfaced since, including DEs for discrete and mixed-integer optimization [39], and multi-objective optimization [4]. An extensive survey of many more DE variants can be found in [15]. For the purposes of this research, only the single-objective continuous case is considered here:

$$f : \mathcal{F} \in \mathbb{R}^n \rightarrow \mathbb{R}, \tag{1}$$

where f is the function to be minimized and $\mathcal{F}$ is the feasible region of the search space, subject to box constraints: $\forall \boldsymbol{x} = \{x_j\}_{j=1}^{n} : \forall j \in [1..n]\, x_j \in [x_j^{\min}, x_j^{\max}]$. Differential Evolution has elegantly simple inner workings (as outlined in Section 2), relying mostly on vector differences between candidate solutions in the population. This simple

structure, coupled with its highly competitive performance, makes DE a good test-bed for hyper-heuristics, which is perhaps the reason why many hyper-heuristic techniques have been proposed and tested for DE. Significant progress was made in the adaptation of DE's control parameters (e.g. [57, 11, 76, 66]), and it has recently become one of the main test-beds for AOS methods (e.g. [60, 27, 28, 21]).

Most AOS and parameter adaptation methods in the current literature solely aim to maximize the improvements w.r.t. the objective function values. While this is generally an effective strategy to accelerate convergence, it can also cause the algorithm to converge prematurely, which is an issue DE already suffers from [63]. Some proposals [47, 46] suggest simultaneously balancing metrics related to solution fitness and population diversity, in order to better tackle the notorious exploration versus exploitation trade-off. This thesis builds on these works and sets up a general framework for diversity-based online tuning of DE, in which we implement novel and existing AOS methods in a modular fashion. Further, we intertwine the AOS component with the adaptation of DE's control parameters. The adaptive algorithms resulting from the framework are then compared to each other and to the state-of-the-art in terms of performance. Furthermore, several aspects of the algorithmic behavior are analyzed.

The structure of the remainder of this thesis is as follows: Section 2 outlines the structure of Differential Evolution and describes its individual components. Section 3 covers the adaptation of DE's operators and parameters, and describes how both components are integrated into the proposed framework. In Section 4, we perform a tuning experiment to find the best settings for each considered AOS method. Section 5 describes the experiments performed with the tuned algorithms, and analyzes the performance results and the algorithmic behavior. The impact of the configuration space, containing the strategies available to the AOS, is assessed in Section 6. Finally, we summarize the main findings of the thesis in Section 7, and give directions for future work in Section 8.

2 Differential Evolution

Like most other Evolutionary Algorithms [5], Differential Evolution (DE) [64] iteratively evolves a population of candidate solutions by means of mutation, crossover, and selection. In a *black-box* scenario, where there is no knowledge about the fitness landscape of the problem to be optimized, the population P is usually initialized uniformly at random (u.a.r.) subject to the boundary constraints:

$$P = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_M\} \subset \mathcal{F}, \tag{2}$$

where M denotes the population size and $\boldsymbol{x}_i$ is a candidate solution, i.e., a n-dimensional vector representing a potential solution to the optimization problem. Hereinafter, we will indicate the jth component of the ith candidate solution as $x_{i,j}$.

For each individual $\boldsymbol{x}_i$, a donor vector $\boldsymbol{v}_i$ (a.k.a. mutant) is generated through mutation, a process where scaled difference vectors are added to some member of the current population, which is called the *base vector*. See Section 2.1 for a selection of DE's mutation strategies which are used in this thesis. Afterwards, for each donor vector $\boldsymbol{v}_i$, a *trial vector* $\boldsymbol{x}'_i$ is created by means of crossover, where components are exchanged between the target vector $\boldsymbol{x}_i$ and the donor vector $\boldsymbol{v}_i$. Two crossover methods are most prominent in DE literature, both of which are described in Section 2.2. Elitist selection is applied between $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$, where the better individual of the two, w.r.t. the objective function, progresses to the next iteration. The pseudocode of DE with binomial crossover and rand/1 mutation, which is commonly referred to as DE/rand/1/bin, is outlined in Alg. 1.

2.1 Mutation

In the mutation step, M *donor vectors* are generated by adding one or more scaled difference vectors to a base vector. Typically, vectors with indices $r_1, r_2, \ldots, r_j$ are selected uniformly at random without replacement, where $\forall j : r_j \neq i$. Depending on the chosen mutation scheme, these randomly selected vectors can appear both in

Algorithm 1 Differential Evolution using rand/1 Mutation and Binomial Crossover

$\boldsymbol{x}_i \leftarrow \boldsymbol{\mathcal{U}}(\boldsymbol{x}^{\min}, \boldsymbol{x}^{\max}), \quad i = 1, \ldots, M.$ ▷ Initialize
while termination criteria are not met **do**
  for $i = 1 \rightarrow M$ **do**
    Choose $r_1 \neq r_2 \neq r_3 \neq i \in [1..M]$ u.a.r.
    $\boldsymbol{v}_i \leftarrow \boldsymbol{x}_{r_1} + F \cdot (\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3})$ ▷ Mutate
    Choose $j_{\text{rand}} \in [1..n]$ u.a.r.
    for $j = 1 \rightarrow n$ **do**
      if $\mathcal{U}(0, 1) \leq Cr$ or $j = j_{\text{rand}}$ **then**
        $x'_{i,j} \leftarrow v_{i,j}$ ▷ Crossover
      else
        $x'_{i,j} \leftarrow x_{i,j}$
      end if
    end for
  end for
  for $i = 1 \rightarrow M$ **do**
    if $f(\boldsymbol{x}'_i) < f(\boldsymbol{x}_i)$ **then**
      $\boldsymbol{x}_i \leftarrow \boldsymbol{x}'_i$ ▷ Select
    end if
  end for
end while

the difference vectors and as the base vector. The difference vectors are scaled by the so-called *mutation rate* $F > 0$, which controls the strength of the mutation operator. Although there is no upper limit, $F > 1$ is considered rarely effective [54]. This research utilizes the following six mutation schemes:

Rand/1 [54] A scaled difference vector of two randomly selected vectors is added to another randomly selected vector.

$$\boldsymbol{v}_i \leftarrow \boldsymbol{x}_{r_1} + F \cdot (\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3}) \tag{3}$$

Best/1 [54] The base vector is chosen as the best member of the population, $\boldsymbol{x}_{\text{best}}$:

$$\boldsymbol{v}_i \leftarrow \boldsymbol{x}_{\text{best}} + F \cdot (\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2}) \tag{4}$$

Target-to-$p$best/1 [66, 76] The base vector is chosen as the target vector, and in addition to the difference vector with randomly selected indices, a difference vector between $\boldsymbol{x}^p_{best}$ and the target vector is used. $\boldsymbol{x}^p_{best}$ is selected uniformly at random from the best $p \cdot 100\%$ members of the population, $p \in (0, 1]$. As recommended in [66], we generate a new value for p every time the operator is applied: $p = \mathcal{U}(\frac{2}{M}, 0.2)$.

$$\boldsymbol{v}_i \leftarrow \boldsymbol{x}_i + F \cdot (\boldsymbol{x}^p_{best} - \boldsymbol{x}_i) + F \cdot (\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2}) \tag{5}$$

Target-to-best/2 [26] Compared to target-to-$p$best/1, another differential vector of vectors with randomly selected indices is used, and $\boldsymbol{x}^p_{best}$ is replaced by the very best vector in the current population, $\boldsymbol{x}_{\text{best}}$.

$$\boldsymbol{v}_i \leftarrow \boldsymbol{x}_i + F \cdot (\boldsymbol{x}_{\text{best}} - \boldsymbol{x}_i) + F \cdot (\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2}) + F \cdot (\boldsymbol{x}_{r_3} - \boldsymbol{x}_{r_4}) \tag{6}$$

Target-to-rand/1 [55, 60, 56] A more exploratory version of target-to-$p$best/1, moving the target vector towards a randomly selected vector instead of $\boldsymbol{x}^p_{best}$:

$$\boldsymbol{v}_i \leftarrow \boldsymbol{x}_i + F \cdot (\boldsymbol{x}_{r_1} - \boldsymbol{x}_i) + F \cdot (\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3}) \tag{7}$$

2-Opt/1 In desire to accelerate the convergence of DE, 2-Opt based DE was proposed in [13]. One of the mutation schemes which was proposed as part of this algorithm, 2-Opt/1, is shown in Eq. 8. The mutation scheme modifies rand/1 by swapping the roles of $\boldsymbol{x}_{r_1}$ and $\boldsymbol{x}_{r_2}$ if the latter has a better objective function value.

$$\boldsymbol{v}_i \leftarrow \begin{cases} \boldsymbol{x}_{r_1} + F \cdot (\boldsymbol{x}_{r_2} - \boldsymbol{x}_{r_3}) & \text{if } f(\boldsymbol{x}_{r_1}) < f(\boldsymbol{x}_{r_2}) \\ \boldsymbol{x}_{r_2} + F \cdot (\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_3}) & \text{otherwise} \end{cases} \tag{8}$$

2.2 Crossover

The crossover step in DE exchanges elements between the target vector $\boldsymbol{x}_i$ and the donor vector $\boldsymbol{v}_i$ (resulting from the mutation step). The resulting vector is called the *trial* vector, denotes as $\boldsymbol{x}'_i$. Here, we describe the two crossover strategies which are most commonly used: binomial and exponential.

Binomial Crossover In the so-called binomial crossover [64], each component $x'_{i,j}$ ($j = 1, \ldots, n$) of $\boldsymbol{x}'_i$ is copied from $v_{i,j}$ with a probability $Cr \in [0, 1]$ (a.k.a. crossover rate), or when j equals an index $j_{\text{rand}} \in [1..n]$ chosen u.a.r.:

$$x'_{i,j} \leftarrow \begin{cases} v_{i,j} & \text{if } \mathcal{U}(0,1) \leq Cr \text{ or } j = j_{\text{rand}} \\ x_{i,j} & \text{otherwise} \end{cases} \tag{9}$$

Exponential Crossover In exponential crossover [64], two integers $p, q \in \{1, \ldots, n\}$ are chosen. The integer p acts as the starting point where the exchange of components begins, and is chosen uniformly at random. q represents the number of elements that will be inherited from the donor vector, and is chosen using the procedure outlined in Algorithm 2.

Algorithm 2 Assigning a value to q in exponential crossover

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

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

$$x'_{i,j} \leftarrow \begin{cases} v_{i,j} \text{ for } j = \langle p \rangle_n, \langle p+1 \rangle_n, \ldots, \langle p+q-1 \rangle_n \\ x_{i,j} \text{ for all other } j \in \{1, \ldots, n\} \end{cases} \tag{10}$$

The angular brackets $\langle\rangle_n$ denote the modulo operator with modulus n.

2.3 Boundary constraint handling

As we are dealing with box constraints, a boundary constraint handling method (BCHM) needs to be implemented to repair, penalize, or handle infeasible solution vectors, that violate the boundaries of the search space, in some other way. The authors of [38, 9] show that, perhaps contrary to the beliefs of many practitioners, situations where 100% of candidate solutions were originally generated outside of the feasible domain are far from unthinkable, especially when dealing with highly dimensional problems. It is easy to understand the importance of the BCHM with this knowledge, considering that, in such situations, nearly all of the generated solutions are a product of this operator. An extensive empirical study was performed in [9], where many BCHMs were tested in combination with a large number of DE variants. The so-called 'Resampling' BCHM [1] showed the best overall performance, which is why it is employed in the experiments for this thesis.

When an infeasible *trial vector* is generated, the mutation operator is simply re-applied (on all vector components) until the resulting vector is located in the feasible domain. Because this process of resampling can potentially go on endlessly, we set the

maximum number of resamples to 100, after which the violating components are placed on their respective boundaries.

2.4 Stopping criteria and independent restarts

Independent restarts are encouraged by the benchmarking procedure guidelines of BBOB/COCO [35], to ensure that function evaluations are not wasted when the algorithm has converged prematurely on a local optimum. Such a restart 'resets' the internal state of the algorithm, losing all the accumulated knowledge about the fitness landscape. It is possible for the algorithm to 'learn' from a failed run that results in a restart, by, for example, increasing the population size [3]. For the sake of simplicity, we do not change DEs parameters between restarts.

Of course, it is important that the algorithm is not restarted too early, potentially preventing the algorithm to converge at all. On the other hand, a restart should be triggered early enough such that as few objective function evaluations as possible are wasted, because they can be extremely time-consuming, especially in real-world situations.

In [77], a thorough examination of many stopping criteria was performed. We adopt the criterion that yielded the best result, named *Diff* in [77]. Let $\boldsymbol{x}_{\text{best}}$ and $\boldsymbol{x}_{\text{worst}}$ denote the best and worst individuals in the current population, respectively. The criterion determines if the algorithm has converged as follows:

$$\texttt{converged} = \begin{cases} \texttt{true} & \text{if } f(\boldsymbol{x}_{\text{worst}}) - f(\boldsymbol{x}_{\text{best}}) < \epsilon \\ \texttt{false} & \text{otherwise} \end{cases} \tag{11}$$

According to [77], setting ϵ one order of magnitude smaller than the desired accuracy of the returned solution is sufficient. Since, in this thesis, we enforce a target precision of $f_{\text{opt}} + 10^{-8}$, we set $\epsilon = 10^{-9}$. A disadvantage of this criterion is that it can be triggered prematurely on problems with plateaus in the fitness landscape.

The algorithm is conclusively terminated if any of the following conditions is satisfied:

- The evaluation budget is exhausted;
- The final target is hit.

3 Strategy adaptation

As mentioned previously, the present work aims to intertwine the adaptation of operators and parameters, and let them work towards a common goal. In the following, we describe the two components – adaptive operator selection (AOS) and parameter adaptation – in detail, and discuss how they are integrated into our proposed framework. An overview of our proposed adaptation framework is shown in Figure 2.

3.1 Adaptive operator selection

Differential Evolution has endured many proposed modifications, which mostly come in the shape of a modified mutation or crossover operator. While such a wide array of strategies is extremely beneficial for tackling problems with different characteristics, making DE more versatile, it also introduces a difficulty: selecting an appropriate strategy becomes an intimidating task, especially if the characteristics of the problem to solve are unknown. Motivated by this, a growing body of literature is dedicated to Adaptive Operator Selection (AOS), where the aim is to select an appropriate strategy for a given problem in an online fasion, i.e., during the search. Further, such AOS methods can adapt the strategy throughout the stages of the search when needed. This is a considerable advantage over *offline* tuning methods, which are not only very time-consuming, but are also incapable of tuning the algorithm in real-time, in response to changes in the environment.

3.1.1 Existing AOS methods

To the best of our knowledge, SaDE [57] is the first Differential Evolution algorithm involving adaptive selection of trial vector generation strategies. Due to their complementing strengths, the strategies rand/1/bin and best/2/bin are considered in the strategy pool, and their application probabilities are adapted based on their respective success rates in a predefined learning period of 50 iterations. The success rates are reset at the start of each learning period so the algorithm can find the optimal strategy for each stage of the search. Additionally, the crossover rate Cr is adapted using previously successful values, and the mutation rate F is generated at random using a normal distribution with mean 0.5 and standard deviation 0.3. SaDE was later improved in [56], by including more operator configurations and enhancing the parameter adaptation procedure.

A completely different approach is EPSDE [44], where a predetermined set of mutation operators and parameter values is used. Initially, each individual is assigned a mutation operator and a parameter settings which is randomly selected from these pools. Individuals that produce improved offspring pass on their configuration of mutation operator and parameter values to the next iteration, while unsuccessful individuals have their configurations reinitialized at random.

CoDE [72] is a relatively simple paradigm, where three popular operator configurations are selected, as well as three commonly used parameter settings. Instead of generating a single trial vector, *three* trial vectors are generated for each target vector, using each of the three strategies once, and each with a parameter configuration selected at random from the available three. Only the best of the three trial vectors is then considered in the selection step. Due to the tripled number of function evaluations per iteration, with the same function evaluation budget, the number of iterations in a run of CoDE will be a third of that of traditional DE.

In SspDE [51], successful values of F and Cr, i.e., values that produced improved offspring, as well as successful operators, are recorded in a set of 'winners'. Parameter values and operators are then either selected from a list, which is refilled with the 'winners' at predetermined intervals, or uniformly at random in order to explore new configurations.

AdapSS-DE [27] considers four trial vector generation strategies, and several reward assignment methods are assessed in combination with the Probability Matching [25] algorithm, which is responsible for assigning the selection probability of each trial vector generation strategy based on their empirically estimated qualities. Later, in [28], the algorithm is extended with parameter adaptation from JADE [76], and another method for assigning application probabilities is used: Adaptive Pursuit [69].

An AOS method based on the multi-armed bandit framework [2] is proposed in [14], in the context of Genetic Algorithms. The algorithm uses the well-known Upper Confidence Bound (UCB) formula [2] to balance exploration and exploitation when selecting strategies. It is acknowledged in [14] that traditional multi-armed bandits are not suitable for the dynamic environment which is encountered when applying AOS to optimization algorithms. Therefore, a *dynamic* multi-armed bandit strategy is proposed, where the multi-armed bandit is restarted when any change in the environment is detected. Such a change in the distribution of rewards is detected by Page-Hinkley test [50]. The robustness of the bandit-based AOS was later reinforced [19] by rewarding configurations based on an Area Under Curve (AUC) scheme, calculating the area under the curve of decayed previous impact measures. This idea was applied to DE in [21], resulting in the algorithm called DE-F-AUC, which yielded competitive results.

In [46], in addition to adaptive operator selection (AOS), the problem of adaptive operator *management* (AOM) is tackled. The so-called 'Blacksmith', responsible for the AOM component, can dynamically add and remove operator configurations from the pool, from which the AOS component, in turn, selects configurations to apply to the population. A constant number of configurations is maintained in the pool of configurations. Operator configurations are removed from the pool if they have been applied for a sufficient number of times and show poor performance. The replacing configurations are then selected uniformly at random.

A classification and survey of many AOS methods can be found in [60]. The authors propose a framework for generating AOS methods, and use this framework to generate

a single AOS method for DE using a racing algorithm. This algorithm, termed U-AOS-FW, is then benchmarked along with several existing AOS methods and other DE variants, yielding favorable results on various classes of problems. The U-AOS-FW is used as a reference algorithm during the performance comparison in Section 5.2.

3.1.2 Proposed approach

We propose a novel AOS framework, the structure of which builds on the aforementioned AdapSS-DE algorithm, especially in the sense that individual-wise rewards (i.e., per candidate solution) are used, which are then aggregated to obtain the reward for each operator configuration. Such an approach has additional benefits, including the intuitive integration with parameter adaptation schemes, discussed later. The main contributions of our AOS framework (compared to existing methods) are:

- We devise a diversity-based metric which captures the direction of movement of a single individual, relative to the mean position of the entire population. This metric is then used to develop several novel *credit assignment* schemes, as well as to adapt some well-known existing methods. Most such credit assignment schemes are intended to stimulate operators and parameter settings that *successfully explore the search space*, i.e., those that moved the individual away from the current population and found an improved solution (in terms of the objective function) as a result. The diversity-based metric offers high flexibility for implementation in combination with methods that aggregate diversity and fitness related metrics, as will become evident later. Further, it allows direct integration with existing state-of-the art parameter adaptation schemes, that rely on individual-wise performance indicators. The AOS framework of [60] does not consider the population diversity in the adaptation process, but it does include a diversity measure over fitness-related metrics. Population diversity management is a core component of our framework, as the direction of the entire adaptation process is controlled by the diversity-based credit values.
- The adaptation of operators is intertwined with the adaptation of parameters, such that both work towards the same goal, which is dictated by the credit assignment scheme. Further, parameter adaptation is performed for each operator configuration *individually*. Previous works have considered using parameter adaptation in combination with AOS, but, to the best of our knowledge, these works adapted the parameters collectively for all operator configurations, and the adaptation always relied exclusively on fitness improvements. In the framework of [60], parameter adaptation is not considered, and static values of F and Cr are used.
- Contrary to existing approaches, which only adapt the mutation operator, our implementation allows adaptation of mutation *and* crossover operators. An *operator configuration* is considered a combination of a mutation and crossover operator. In the framework of [60], nine mutation operators are involved in the AOS process, and only binomial crossover is considered.

The remainder of this section covers the four main components of the proposed AOS framework (credit, reward, quality, and probability), and discusses the considered variants of each component.

3.1.3 Credit: an individual-wise improvement metric

In the literature (e.g., [27, 28, 45, 60]), 'credit' and 'reward' are commonly used interchangeably in the context of AOS, but we make a clear distinction between the two. In this thesis, 'credit' will be used to refer to an *individual-wise* metric based on a difference in fitness, diversity, or both, between a trial vector and its parent: the target vector. A high credit value indicates that the trial vector generation was deemed successful by the employed credit assignment scheme.

Although population diversity is a metric that is computed over the entire population, we need a way to measure 'diversity' for a single individual. Population diversity can be measured in many different ways [49], one of which is the standard deviation of positions of candidate solutions in the search space:

$$\delta = \frac{\sum_{i=1}^{M} ||\boldsymbol{x}_i - \bar{\boldsymbol{x}}||}{M}, \tag{12}$$

where $||\boldsymbol{a} - \boldsymbol{b}||$ denotes the Euclidean distance between individuals $\boldsymbol{a}$ and $\boldsymbol{b}$, and $\bar{\boldsymbol{x}}$ is the mean position of the population:

$$\bar{x}_j = \frac{\sum_{i=1}^{M} x_{i,j}}{M}. \tag{13}$$

Clearly, the 'contribution' of one individual (i) to the population diversity is proportional to the distance from that individual to the mean position of the population: $||\boldsymbol{x}_i - \bar{\boldsymbol{x}}||$. To measure the relative impact of one individual on the population diversity, we define the **diversity ratio** as the ratio between (1) the distance between the trial vector and the mean position and (2) the distance between the target vector and the mean position:

$$r_i^d = \frac{||\boldsymbol{x}_i' - \bar{\boldsymbol{x}}||}{||\boldsymbol{x}_i - \bar{\boldsymbol{x}}||} \tag{14}$$

A diversity ratio of $0 \leq r_i^d < 1$ indicates that the individual moved towards the mean, and thus contributed to reducing the population diversity, while a diversity ratio of $r_i^d > 1$ indicates that the individual moved away from the mean, and thereby potentially increased the population diversity. Of course, whether the individual has *actually* increased the population diversity also depends on the positions of the rest of the population, in the next iteration. It is important to consider that, with small population sizes, the mean position can be heavily affected by outliers, which can have an undesirable effect on the resulting diversity ratios. Seven credit assignment methods are assessed in this work, some of which use this diversity ratio. Additionally, each of them uses the fitness difference w.r.t. the target vector:

$$\Delta f_i = f(\boldsymbol{x}_i) - f(\boldsymbol{x}_i') \tag{15}$$

Diversity Ratio This method simply assigns the diversity ratio of a trial vector to its credit, but only if it has improved w.r.t. the corresponding target vector. If the individual has deteriorated in terms of fitness, a null credit value is assigned. This way, the highest rewards are associated with individuals that successfully explored the search space. Individuals that successfully exploit the current best position(s) and, as a result, move towards the mean position of the population, still get rewarded, but to a lesser extent. Note that the magnitude of the fitness improvement has no influence on this credit value at all.

$$c_i = \begin{cases} r_i^d & \text{if } \Delta f_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

Squared Diversity Ratio To amplify the reward for exploring the search space, and reward exploitative behavior even less, the diversity ratio is squared:

$$c_i = \begin{cases} (r_i^d)^2 & \text{if } \Delta f_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

Fitness Improvement This is the only credit assignment method considered in this thesis which does not take into account diversity, and simply assigns the fitness difference w.r.t. the target vector, if is positive. This, or something similar, is the approach most AOS and parameter adaptation methods in the current literature take (e.g. [66, 76, 28, 18]).

$$c_i = \begin{cases} \Delta f_i & \text{if } \Delta f_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

Combined Fitness and Diversity Here, we combine the fitness and diversity measures by scaling the fitness improvement with the diversity ratio:

$$c_i = \begin{cases} \Delta f_i \cdot r_i^d & \text{if } \Delta f_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

Combined Fitness and Squared Diversity Here, the fitness improvement is scaled by the squared diversity ratio:

$$c_i = \begin{cases} \Delta f_i \cdot (r_i^d)^2 & \text{if } \Delta f_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

Compass We propose a slightly modified version of the Compass method proposed in [47], which balances the fitness and diversity. Instead of the diversity ratio, here we use the **diversity difference**, denoted Δd_i:

$$\Delta d_i = ||\boldsymbol{x}_i' - \bar{\boldsymbol{x}}|| - ||\boldsymbol{x}_i - \bar{\boldsymbol{x}}||, \tag{21}$$

as it is more compatible with the Compass method. For the fitness measure, we use the fitness difference Δf_i (Eq. 15). First, the diversity and fitness differences are normalized to the range $[-1, 1]$ using the respective largest absolute values in the current trial population:

$$\Delta d_i' = \frac{\Delta d_i}{\max_j |\Delta d_j|}, \quad \Delta f_i' = \frac{\Delta f_i}{\max_j |\Delta f_j|} \tag{22}$$

The coordinates $\boldsymbol{V}_i = (\Delta d_i', \Delta f_i')$ are interpreted as vectors. The desired balance between exploration and exploitation is defined by a unit vector $\boldsymbol{C}$, obtained from parameter $\Theta \in [0, \frac{\pi}{2}]$, the angle of this vector in radians:

$$\boldsymbol{C} = (\cos(\Theta), \sin(\Theta)) \tag{23}$$

The amount of credit for each individual is then based on the 'similarity' of $\boldsymbol{V}_i$ to $\boldsymbol{C}$. Thus, it is computed as the scalar projection of $\boldsymbol{V}_i$ in the direction of $\boldsymbol{C}$:

$$c_i = \boldsymbol{V}_i \cdot \boldsymbol{C}, \tag{24}$$

where the '$\cdot$' operator denotes the dot product. We choose an angle of $\Theta = \frac{\pi}{4}$, signaling an equal importance of the fitness and diversity metrics. The authors of [45] found that other values cause an undesirable positive-feedback phenomenon, where no balance is maintained between exploration and exploitation, but the adaptation moves toward one of the extremes.

An important modification we make w.r.t. the Compass method proposed in [47], is that the method is applied in an individual-wise manner, instead of configuration-wise. In other words, the original paper used the method to assign a reward to operator configurations based on their most recent applications, while we use it to measure the improvement of a trail vector compared to its parent.

For clarification, Figure 1 shows a simple example of the inner workings of the Compass method. The values of $\Delta f_i'$ and $\Delta d_i'$ of four individuals are shown in this case, and the derivation of c_i is only shown for the vector $\boldsymbol{V}_i$, which is closest to the 'needle of the compass' $\boldsymbol{C}$.

It is worth noting that the original Compass method [47] also takes into account the wall clock time that elapsed during the application of each operator, to reward operators with a short execution time. We discard this aspect, as we are not interested in the execution time of operators.

Pareto Dominance This credit assignment scheme is adapted from the *Pareto Dominance* method proposed in [46]. Similar to the Compass method described hereinbefore, the diversity difference (Eq. 21) and the fitness difference (Eq. 15) are used here. Since we want to simultaneously maximize fitness and diversity, a coordinate $\boldsymbol{V}_i = (\Delta d_i, \Delta f_i)$ *dominates* another coordinate $\boldsymbol{V}_j$ if all of its components are larger or

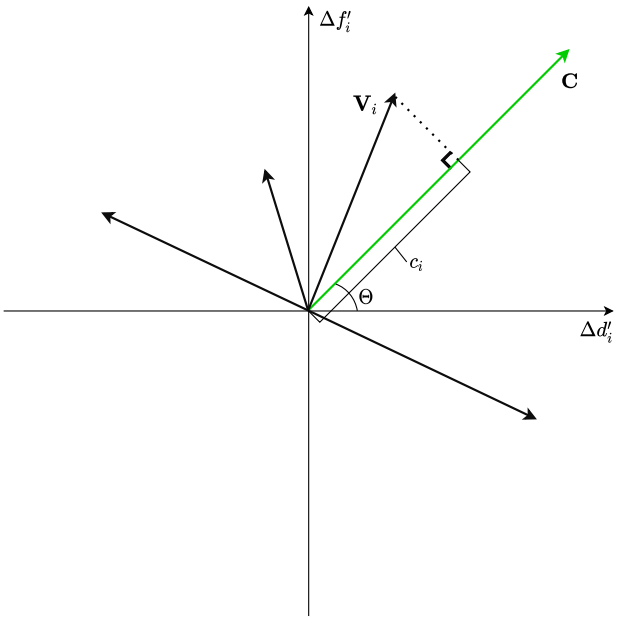


Figure 1: Illustration of the inner workings of the Compass credit method.

equal than the respective components of $\boldsymbol{V}_j$, and at least one component is larger. In this case, it can be expressed as follows:

$$\texttt{dominates}(i,j) = \begin{cases} \texttt{true} & \text{if } (\Delta d_i > \Delta d_j \texttt{ and } \Delta f_i \geq \Delta f_j) \texttt{ or } (\Delta f_i > \Delta f_j \texttt{ and } \Delta d_i \geq \Delta d_j) \\ \texttt{false} & \text{otherwise} \end{cases} \tag{25}$$

Now, each individual's credit is computed as the number of other individuals it dominates. Again, the original paper [46] used this method to compare configurations, while we employ it in an individual-wise manner.

3.1.4 Reward: a configuration-wise improvement metric

After computing the individual-wise credit metric, these values need to be aggregated into configuration-wise reward values, to be able to assess the performance of each configuration in the past iteration. Let $\boldsymbol{C}_k$ denote the set of **nonzero** credit values of individuals that were generated by configuration k in the previous iteration. We consider six schemes that aggregate the credit values in $\boldsymbol{C}_k$ into a *reward* value r_k for the associated configuration. In all cases, if $\boldsymbol{C}_k = \varnothing$, then $r_k = 0$. The first four methods listed here were proposed in [20] and later also used in AdapSS-DE [27] and AdapSS-JADE [28]:

Average Absolute (AA)

$$r_k = \frac{\sum_{j=1}^{|\boldsymbol{C}_k|} C_{k,j}}{|\boldsymbol{C}_k|}, \tag{26}$$

where $|\boldsymbol{C}_k|$ denotes the size of $\boldsymbol{C}_k$.

Average Normalized (AN)

$$r_k = \frac{r'_k}{\max_j r'_j}, \text{ where } r'_k = \frac{\sum_{j=1}^{|\boldsymbol{C}_k|} C_{k,j}}{|\boldsymbol{C}_k|} \tag{27}$$

Extreme Absolute (EA)

$$r_k = \max_j C_{k,j} \tag{28}$$

Extreme Normalized (EN)

$$\frac{r'_k}{\max_j r'_j}, \text{ where } r'_k = \max_j C_{k,j} \tag{29}$$

Finally, we propose two ranking-based reward schemes. The offspring is ranked in ascending order according to their credit values c_i. The reward r_k is then based on the ranks $\boldsymbol{R}_k$ of the individuals that are associated with configuration k. Individuals with a null credit obtain $R_{k,j} = 0$. The ranking of individuals with positive credit starts at 1, and individuals with equal credit obtain the same rank. The ranks can then be aggregated in two ways:

Average Rank (AR)

$$r_k = \frac{\sum_{j=1}^{|\boldsymbol{R}_k|} R_{k,j}}{|\boldsymbol{R}_k|} \tag{30}$$

Extreme Rank (ER)

$$r_k = \max_j R_{k,j} \tag{31}$$

Since the ranking-based rewards are always approximately in the same range, there is no need to also consider normalized variants.

3.1.5 Quality: a configuration-wise performance indicator

The rewards which are generated for each configuration are accumulated in the *quality* metric, which is meant to capture the quality of the configuration in the current stage of the search. It is initialized as $q_k = 0$ $(k = 1, \ldots, K)$, where K is the number of operator configurations. We employ a simple weighted sum [69], which takes into account the current quality and the newly acquired reward:

$$q_k \leftarrow q_k + \alpha \cdot (r_k - q_k) \tag{32}$$

The parameter α (where $0 < \alpha \leq 1$) controls the speed at which the quality is changed. Intuitively, the parameter should be sufficiently high such that the algorithm can react quickly when strategy adaptation is required due to, for example, changes in the fitness landscape. On the other hand, α should not be *too* large, to ensure that the empirical qualities are not heavily affected by randomness.

We only update the quality of an operator configuration if it was used by at least one individual in the previous iteration, as it is unfair to reduce the quality of a configuration if it was, by chance, not selected for application.

3.1.6 Application probability

The final step in our framework of adaptive operator selection is assigning the application probability of each configuration based on the empirical quality measures. We consider two methods for adapting this probability. Both methods ensure that a configuration never obtains a null probability. This is necessary, because such a configuration would never be able to re-enter the pool of available configurations, as it is impossible to increase its quality measure by accumulating rewards, if it is never selected. In a stationary environment, this property may not be problematic, as such a poorly performing configuration might better be avoided. However, it is important to consider that we are dealing with a *dynamic* environment, where the optimal configuration not only depends on the problem to solve, but also on the stage of the search. An operator configuration that performs poorly at the start of the search might still become superior in a later stage. The probabilities p_k are initially all equal: $p_k = \frac{1}{K}$ $(k = 1, \ldots, K)$.

Probability Matching (PM) Probability Matching [25] is a simple method where each configuration is assigned at least the minimal probability $p_{\min}$, and the remaining probability is distributed proportionally according to the empirical quality estimates q_i.

$$p_k \leftarrow p_{\min} + (1 - K \cdot p_{\min}) \cdot \frac{q_k}{\sum_{j=1}^{K} q_j}, \tag{33}$$

Adaptive Pursuit (AP) In desire to accelerate the convergence on the optimal operator configuration, Adaptive Pursuit [69] was proposed in the context of Genetic Algorithms. The algorithm is inspired by *pursuit algorithms* for learning automata. Adaptive Pursuit is made suitable for an adaptive environment by enforcing minimal application probabilities, similar to Probability Matching. First, the index of the configuration with the current highest quality estimate is denoted as θ:

$$\theta = \text{argmax}_k q_k \tag{34}$$

Then, the application probabilities are computed as follows:

$$p_k \leftarrow \begin{cases} p_k + \beta \cdot (p_{\max} - p_k) & \text{if } k = \theta \\ p_k + \beta \cdot (p_{\min} - p_k) & \text{otherwise} \end{cases}, \tag{35}$$

where $p_{\max}$ is the *maximum* application probability, simply computed as:

$$p_{\max} = 1 - (K - 1) \cdot p_{\min} \tag{36}$$

A configuration that has the highest quality in consecutive iterations will eventually converge to $p_{\max}$, at a rate dictated by the parameter β. All other configurations will converge to $p_{\min}$.

3.2 Adaptation of control parameters

An ongoing problem in the field of Evolutionary Algorithms is the sensitivity to control parameters [16]. Standard Differential Evolution has relatively few parameters, namely the mutation rate F, crossover rate Cr, and population size M. Still, as the optimal settings of control parameters is problem-dependent [64, 29, 48, 58], tuning these parameters is essential in order to obtain the desired result.

For this reason, much effort has gone toward adapting the parameter values during the optimization process, for example in jDE [10], JADE [76], and SaDE [57]. In our experiments, we use the state-of-the-art adaptation scheme of SHADE [66], which adapts the parameters F and Cr. The population size M is kept constant at a constant value in our framework. Two important modifications are made w.r.t. the original SHADE:

- Parameters are adapted *per operator configuration*. There have been previous works, e.g. [27, 57], which combined AOS with parameter adaptation. However, these methods did not take into account that the optimal parameter values might also be dependent on the employed mutation and crossover operators.
- Originally, SHADE was designed to produce parameter values to maximize fitness improvements. Considering the direction of the present research, we adapt the parameters based on the values produced by the credit assignment method (see Section 3.1.3), such that the adaptation of operators and the adaptation of parameters work towards a common goal. When the 'Fitness Improvement' credit is employed, adaptation occurs based on the same values as in original SHADE.

In the original version of SHADE, two memories $\boldsymbol{M}^{Cr}$ and $\boldsymbol{M}^{F}$, both of size H, are maintained, which are used to generate values for Cr and F, respectively. Because we need to adapt the parameters per operator configuration, such a memory is maintained for each configuration individually. The original paper [66] reported good results when using $H = M$. Since, in our implementation, each operator configuration maintains its own memory, we need to make the memory size dependent on the number of configurations, such that adaptation does not slow down for large K. For this reason, and

because H should not be too small (to avoid oscillations due to randomness), we use $H = \max(\frac{M}{K}, 10)$. The memories for each configuration k are initialized as:

$$\boldsymbol{M}_k^{Cr}, \boldsymbol{M}_k^F \leftarrow 0.5 \cdot \mathbf{1} \tag{37}$$

The *successful* values of Cr and F, of the most recent iteration only, are recorded in $\boldsymbol{S}_k^{Cr}$ and $\boldsymbol{S}_k^F$, respectively. In original SHADE, parameter values are considered to be successful when the generated offspring has an improved fitness value compared to its parent. In the light of our generalized approach, where we aim to tune the parameters for maximizing the offspring's credit values, we consider a pair of F and Cr successful if it generated an individual with a positive credit value.

At the end of each iteration of the DE algorithm, the hth component of both memory vectors is updated. The index h is initialized as 1, and updated each iteration as:

$$h \leftarrow \begin{cases} h+1 & \text{if } h < H \\ 1 & \text{otherwise} \end{cases} \tag{38}$$

The memory of crossover rates is then updated as:

$$M_{k,h}^{Cr} \leftarrow \begin{cases} \text{mean}_{\text{WA}}(\boldsymbol{S}_k^{Cr}) & \text{if } \boldsymbol{S}_k^{Cr} \neq \emptyset \\ M_{k,h}^{Cr} & \text{otherwise} \end{cases}, \tag{39}$$

where mean_{WA} denotes the weighted mean:

$$\text{mean}_{\text{WA}} = \sum_{i=1}^{|\boldsymbol{S}_k^{Cr}|} w_i^k \cdot S_{k,i}^{Cr}, \tag{40}$$

and w_i^k is a weight proportional to the relative amount of credit of the ith individual associated with configuration k, compared to other successful individuals associated with the same configuration:

$$w_i^k = \frac{C_{k,i}}{\sum_{j=1}^{|\boldsymbol{C}_k|} C_{k,j}} \tag{41}$$

The memory of mutation rates is updated as:

$$M_{k,h}^F \leftarrow \begin{cases} \text{mean}_{\text{WL}}(\boldsymbol{S}_k^F) & \text{if } \boldsymbol{S}_k^F \neq \emptyset \\ M_{k,h}^F & \text{otherwise} \end{cases}, \tag{42}$$

where mean_{WL} is the weighted Lehmer mean:

$$\text{mean}_{\text{WL}}(\boldsymbol{S}_k^F) = \frac{\sum_{i=1}^{|\boldsymbol{S}_k^F|} w_i^k \cdot (S_{k,i}^F)^2}{\sum_{i=1}^{|\boldsymbol{S}_k^F|} w_i^k \cdot S_{k,i}^F} \tag{43}$$

The actual crossover and mutation rates in each iteration are then generated using the memories. For an individual that will use configuration k in the next iteration, they are generated as follows:

$$Cr_i \leftarrow \max(\min[\mathcal{N}(M_{k,r_i}^{Cr}, 0.1), 1], 0) \tag{44}$$

$$F_i \leftarrow \min(\mathcal{C}[M_{k,r_i}^{Cr}, 0.1], 1) \tag{45}$$

Where $\mathcal{C}$ is a Cauchy distribution, $\mathcal{N}$ is a normal distribution, and r_i is chosen uniformly at random in $[1, H]$ for each individual. When $F_i \leq 0$, Eq. 45 is re-applied until a valid value is obtained.

3.3 Tying it all together: the adaptation manager

The discussed adaptation of operators and parameters is combined in an elegant and modular algorithmic framework. The source code of the software, which is written in C++, is available under the GNU GPLv3 license at `https://github.com/rickboks/auto-DE`. Figure 2 outlines the general structure of the framework and the interaction between the individual components.

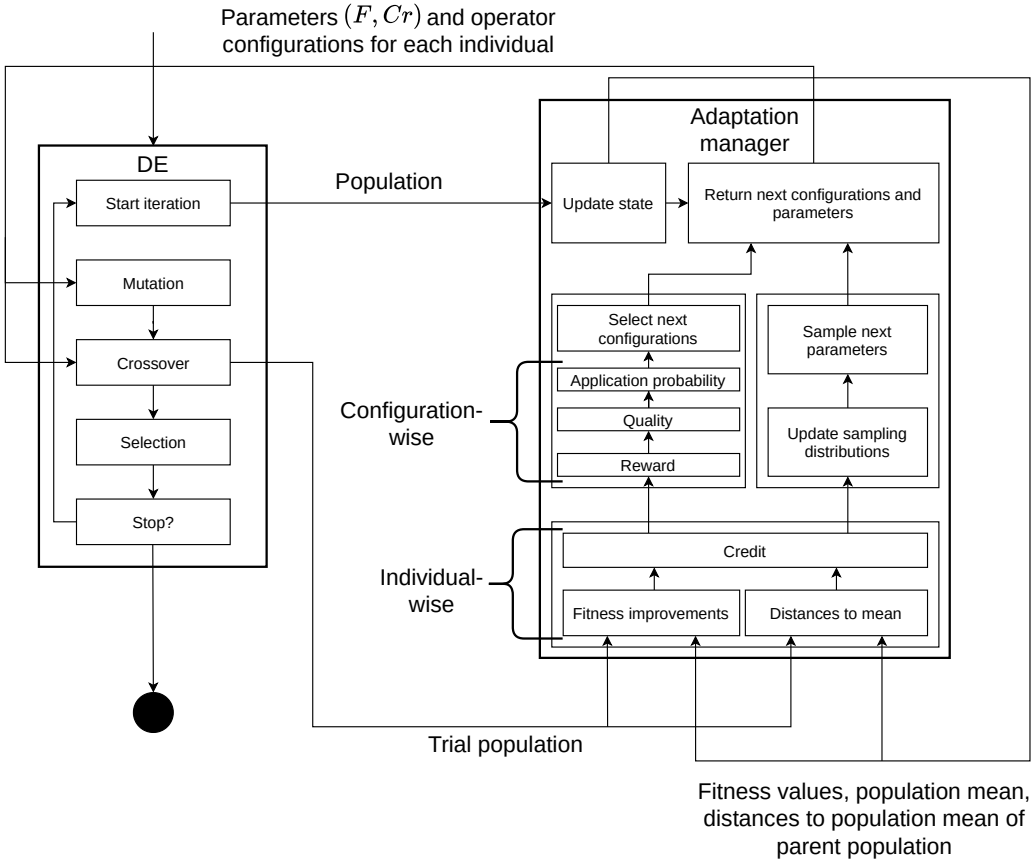


Figure 2: Diagram of the interaction between the (standard) Differential Evolution algorithm, the 'adaptation manager' and their inner workings.

The two components – the DE algorithm and the so-called 'adaptation manager' – communicate at two stages of each iteration. At the start of each iteration, the DE algorithm passes its current population with the associated fitness values to the adaptation manager. The adaptation manager then internally stores the fitness values, computes the mean position of the population, and computes the distance to this population mean for each individual. It then returns the next values of F and Cr, as well as the operator configurations for each individual, to the DE algorithm.

After the crossover step, the trial population (with associated fitness values) is passed to the adaptation manager, which computes the distances of each trial vector to the previous population mean. We now have access to, for each individual, the distance to the population mean before and after application of the mutation and crossover operators. In addition, the fitness improvements of the trial population compared to the parent population are computed. The diversity-related data and fitness improvements are used to compute the credit values, which indicate the improvements of each trial individual compared to their parents according to the employed credit assignment scheme.

The credit values have a central role in the strategy adaptation: they steer both the adaptation of operators and the adaptation of parameters. For the AOS component,

they are aggregated into reward values, which are in turn used to update the quality of each operator configuration. The quality values are then used to update the application probability of each configuration in the next iteration.

Simultaneously, the parameter adaptation scheme utilizes the credit values to determine which parameter values were most successful. This information is then used to update the sampling distributions which are used to generate future values of F and Cr.

It is worth mentioning that, while we use a modified SHADE parameter adaptation scheme in our AOS framework, the individual-wise credit values can also be directly integrated into most other parameter adaptation schemes. The reason for this is that most parameter adaptation schemes in the current literature rely on the individual-wise fitness improvement values. These values can be replaced by the values generated by a credit assignment scheme of choice, by which the objective of the parameter adaptation scheme is essentially modified. It should also be fairly trivial to transfer the proposed adaptation manager to different population-based metaheuristics altogether.

4 Parameter tuning

While the AOS mechanism is meant to reduce the manual tuning effort, it actually introduces additional parameters itself. However, instead of directly controlling DE's search strategy, these parameters control the adaptation behavior. The performance of DE across different problems should, therefore, be much less sensitive to these AOS-related parameters.

For the scope of the present research, we are mainly interested in the credit assignment scheme, which controls the degree at which diversity and fitness improvements are rewarded, respectively. In an attempt to find optimal settings for each credit assignment scheme, a racing algorithm is used to automatically tune the AOS-related parameters for each credit assignment scheme individually. This will allow us to fairly compare the performance of the seven resulting AOS methods afterwards.

Specifically, we use the `irace` package [43] for R, which implements the iterated racing procedure [6, 7]. Iterated racing is an example of *offline* tuning, where there is a clear distinction between the *tuning* phase, where it is attempted to find the optimal configuration of an algorithm, and a *testing* phase, where the tuned algorithm is assessed on unseen problem instances, or, in a real-world scenario, where it is used to optimize the problem at hand.

The algorithm consists of *races*, the number of which depends on the number of parameters to be tuned. At the start of each race, a finite set of configurations is sampled at random. Until a minimum number of configurations remains or a maximum number of experiments is performed, each configuration is tested on a problem instance, and configurations that perform significantly worse than at least one other configuration are discarded. The parameters' sampling distributions are iteratively updated towards the best found configurations, in an attempt to find better configurations in the future. To identify which configurations perform significantly worse, the Friedman test [24] is employed.

`irace` is given a budget of 10^5 experiments, where each experiment entails a single run (which may include restarts) on a single benchmark function, which in turn is given a budget of $10^5 \cdot n$ function evaluations. The 24 benchmark functions from the BBOB/-COCO [35] suite are incorporated in the so-called training set, with a dimensionality of $n = 20$. To avoid overfitting on a specific function instance, the training set includes the first two instances of each benchmark function. `irace` randomly samples new problem instances from this training set at the start of each race.

Since we are dealing with a *heterogeneous* set of problems, i.e., it is expected that the performance of each algorithm can vary greatly across different problems, five new problem instances from the training set are introduced at the start of each race instead of the default setting of 1. This setting is recommended in the documentation of the `irace` package [42], in order to find configurations which perform best overall, but not necessarily on any individual problem. The remaining parameters of `irace` are left at their default values.

`irace` requires a metric to be returned after each run, indicating the performance of the generated configuration on the current problem instance. The objective function value of the best found solution is used for this purpose. Considering the extremely high computational load of the described tuning experiment, it is parallelized within `irace`, and the seven tuning experiments, one for each credit assignment scheme, are further parallelized using GNU Parallel [68].

There are five parameters to be tuned by `irace`, listed in Table 1. The parameter γ controls the $p_{\min}$ parameter used in Probability Matching and Adaptive Pursuit. Such a parameter is employed to make the setting of $p_{\min}$ valid for all configuration space sizes. For example, a setting of $p_{\min} = 0.1$ might be reasonable for a scenario with a number of $K = 4$ configurations, but for $K = 12$, the value is invalid because $12 \cdot 0.1 > 1$. Additionally, in most situations, it is probably desirable to set $p_{\min}$ sufficiently low such that $p_{\max}$ is considerably larger, allowing the best configuration to be activated much more frequently than the others.

The parameter γ indirectly controls the minimal total application probability of the $K - 1$ inferior configurations:

$$p_{\min} = \frac{1}{\gamma \cdot (K - 1)} \tag{46}$$

In this manner, the best configuration can attain an application probability of at most $\frac{\gamma-1}{\gamma}$, in which case the combined probability of all other configurations is $\frac{1}{\gamma}$.

Table 1: Parameters to be tuned by `irace`.

Parameter	Valid settings	Constraints
Reward	{AN, AA, EN, EA, ER, AR}	—
Probability	{AP, PM}	—
α	$\{x \in \mathbb{R} \mid 0 < x \leq 1\}$	—
β	$\{x \in \mathbb{R} \mid 0 < x \leq 1\}$	Probability = AP
γ	$\{x \in \mathbb{R} \mid 2 \leq x \leq 10\}$	—

A population size of $M = 5n = 100$ is used following the findings of [52]. The algorithms are run with a *configuration space* consisting of three mutation and two crossover strategies, resulting in $3 \cdot 2 = 6$ operator configurations. We selected the rand/1, best/1, and target-to-best/2 mutation strategies for this experiment, as they possess diverse characteristics and showed good performance in specific function groups in our previous experiments [9], indicating that they, as an 'ensemble', might be successful across a large variety of problems. The same experiments also showed a large advantage in terms of performance for DE with either binomial or exponential crossover, depending on the characteristics of the problem. For this reason we expect the AOS to benefit from incorporating both of these crossover schemes in the configuration space.

Table 2: Abbreviations of the credit assignment methods. The last three columns indicate which metrics (Δf_i, r_i^d and Δd_i) are used by each one. A black cell indicates that the corresponding metric is used.

Credit assignment method	Abbreviation	Δf_i	r_i^d	Δd_i
Fitness Improvement (Eq. 18)	Fit	■		
Combined Fitness and Diversity (Eq. 19)	FitDiv	■	■	
Combined Fitness and Squared Diversity (Eq. 20)	FitSqDiv	■	■	
Diversity Ratio (Eq. 16)	Div	■	■	
Squared Diversity Ratio (Eq. 17)	SqDiv	■	■	
Pareto Dominance (Eq. 25)	Pareto	■		■
Compass (Eq. 24)	Compass	■		■

In the following, we abbreviate some of the credit assignment methods described in Section 3.1.3 in the interest of legibility. The abbreviations of the credit assignment methods are listed in Table 2, as well as the diversity and fitness-related metrics they each use. The tuned parameters, returned by `irace`, are listed in Table 3. Interestingly, all seven algorithms converged on the Adaptive Pursuit probability scheme and a reward scheme which uses the extreme credit values. All seven tuning experiments settled on a high value of γ, signaling that it is beneficial to allow the best configuration to obtain a high application probability of close to 90%. The algorithms with Compass and Pareto Dominance credit have converged on a significantly higher value of α than the others, indicating that a rapid adaptation of the quality metrics (q_k) is desirable for these algorithms. There are also huge differences in the values of β, which controls the speed at which an operator configuration can increase its application probability once its quality has surpassed all others. We expect this parameter to have a lesser impact on the algorithm's behavior compared to α, as Adaptive Pursuit adapts quite quickly, even with a low value of β. The very low values of β for Div and SqDiv will, however, certainly have an impact on the speed of adaptation.

Table 3: Parameters returned by the `irace` tuning procedure, tuned on the 24 functions of BBOB/COCO

Parameter	Credit assignment method						
	Fit	FitDiv	FitSqDiv	Div	SqDiv	Compass	Pareto
Reward	ER	EN	EA	EN	EN	EA	EA
Probability	AP	AP	AP	AP	AP	AP	AP
α	0.1	0.07	0.03	0.05	0.11	0.59	0.54
β	0.68	0.33	0.52	0.03	0.05	0.13	0.3
γ	9.65	8.74	8.15	9.92	8.38	9.52	8.98

5 Benchmarking tuned adaptive DEs

To find out which of the credit assignment schemes is most successful in directing the course of DE's strategy adaptation, the seven tuned adaptive DEs resulting from the tuning procedure described in Section 4 are benchmarked on the 24 functions of the BBOB suite, with dimensionality $n = 20$, an objective function evaluation budget of $10^5 \cdot n = 2 \cdot 10^6$, and the same further parameter settings as described in Section 4. Each algorithm is run on the first five instances of each function, each of which is repeated 20 times, resulting in 100 independent runs of each algorithm on each function. In addition, we include the six operator configurations used in the configuration space of the AOS methods as 'static' DE instances. As a baseline for the AOS methods, we consider a non-adaptive method that always assigns the application probability uniformly across each of the six configurations and is termed 'Uniform'. By considering the Uniform method, we can determine if the AOS component actually has an impact on the performance, or if any performance differences are simply caused by the fact that multiple operator configurations are involved. The parameter adaptation of this, and the six static DEs, is based on the fitness improvements, matching the 'Fit' credit scheme and the original SHADE algorithm.

5.1 Fixed-budget comparison

Table 4 lists the mean best-reached error values and the number of successful runs (out of 100) for each algorithm on each benchmark function. Here, the error value is the objective function value of the best-found solution minus the objective function value of the optimum, f_{opt}. A trial was considered successful when an objective function value of $f_{\text{opt}} + 10^{-8}$ was obtained, after which the trial was terminated. Therefore, 10^{-8} is the smallest possible mean error value. The statistical significance of the results is tested with the one-sided Wilcoxon signed-rank test [73] from the `scipy` [71] package, which tests if the distribution of best-reached function values (100 for each function) differs

between two algorithms with a confidence level of 5%. In Table 4, the lowest mean error value(s) on each function are marked in **bold**. Cells are colored light gray when the corresponding algorithm was found significantly worse than at least one other algorithm on the function in question. A dark gray background is used if the corresponding algorithm is significantly better than *all* other algorithms. In the table, 'target-to-best' is abbreviated as 'ttb'.

Results From the 'static' DE algorithms (shown in the last 6 columns of Table 4), i.e., those using only a single operator configuration, target-to-best/2/bin seems to show to best overall performance. Apart from one failed trial on $f13$, it consistently solved functions $f1 - f14$. Target-to-best/2/exp performed slightly worse on these functions. The other static DEs had varying success depending on the problem. The rand/1/$*$ and best/1/$*$ DEs did not have a single successful run on functions $f13$ and $f14$, where the target-to-best/$*$ variants succeeded in most runs. Overall, it is clear that the different operators can have a huge impact on the performance across different problems. For example, rand/1/bin performs much better than the other static configurations on $f17$, while best/1/bin, which underperforms on many other functions, outperforms the other static DEs on $f22$.

Some algorithms, most notably best/1/$*$, are seemingly suffering from the plateaus in the fitness landscape of $f7$, which can trigger premature restarts due to the employed restart criterion (see Section 2.4). However, for other algorithms, such as target-to-best/2/$*$, this appears not to be an issue, as they have a 100% success rate on this problem.

The most competitive DE with AOS, and perhaps the best algorithm overall, seems to be the Compass method, which has a 100% success rate on functions $f1 - f14$. It solved several problems where the other AOS methods could not successfully complete a single run. Interestingly, the Compass and Pareto methods were are able to solve problems which the static DEs could not: for example, they hit $f16$'s final target 11 and 10 times respectively, where all other algorithms failed in every trial. Furthermore, the same algorithms were also able to solve $f23$ several times where no other algorithm could, and Compass was the only algorithm able to solve $f18$.

This might indicate that these AOS methods, instead of simply selecting the one 'best' strategy (which is already quite remarkable), are capable of utilizing the strengths of the available strategies at the appropriate stages of the search, in such a way that it performs better than any of the strategies individually. While Pareto shows this quality on some functions, it also lagged behind on a number of other functions, making it much less reliable than Compass. Another possible reason for the improved performance compared to the static DEs is the modified 'direction' of the parameter adaptation, dictated by the credit assignment scheme.

In general, the Uniform method performs considerably worse than the 'real' AOS methods. At the very least, this shows that the AOS methods adapt their strategies in some meaningful way. The three Fit$*$ methods perform very similarly, as do the Div and SqDiv methods. Div and SqDiv generally perform better that the Fit$*$ variants on most functions, indicating that the diversity ratio alone is already capable of steering the direction of adaptation, without necessarily having to consider a second, fitness-related, measure. The Fit$*$ methods did however perform better on a few functions, showing that both metrics have their merit in different situations. The Compass method seems to be able to balance the fitness- and diversity-related measures the best out of all AOS methods.

The statistical testing results are summarized in Table 5, which shows the number of problems on which each algorithm is significantly better or worse than another algorithm. It also shows the number of functions on which they are indistinguishable (neither significantly better nor worse). The columns are compartmentalized into 'wins', 'ties', and 'losses' (W/T/L) from the perspective of the algorithm corresponding to the row. For example, Compass is significantly better than Fit 9 times, indistinguishable 10 times, and worse 5 times. The table is symmetrical in the main diagonal, except the wins and losses are swapped. The algorithms that have the most wins against a certain algorithm are highlighted in **bold**, and those with the most losses are signified with a light gray background.

Table 4: Mean best-reached error values and number of successful runs (out of 100) for seven AOS methods, a baseline method 'Uniform' and the six static DEs. See Section 5.1 for an explanation of usage of boldface fonts and cell coloring.

	Fit		FitDiv		FitSqDiv		Div		SqDiv		Compass		Pareto		Uniform		rand/1/bin		rand/1/exp		best/1/bin		best/1/exp		ttb/2/bin		ttb/2/exp	
function	mean	succ	mean	succ	mean	succ	mean	succ	mean	succ	mean	succ	mean	succ	mean	succ	mean	succ	mean	succ	mean	succ	mean	succ	mean	succ	mean	succ
f1	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**
f2	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**
f3	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	6.19	0	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**
f4	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	1.17e+1	0	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**
f5	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**
f6	**1e−8**	**100**	1.0e−8	99	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	7.75e−3	88	**1e−8**	**100**	**1e−8**	**100**	2.33e−1	88	**1e−8**	**100**	7.36e−2	93	**1e−8**	**100**	**1e−8**	**100**
f7	1.96e−3	89	3.01e−3	85	9.0e−3	82	6.99e−3	83	1.08e−2	69	**1e−8**	**100**	**1e−8**	**100**	4.2e−1	1	2.45e−2	31	7.13e−3	62	1.02	0	2.67e−1	0	**1e−8**	**100**	**1e−8**	**100**
f8	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**
f9	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	5.2e−1	94	**1e−8**	**100**	**1e−8**	**100**	9.85e−2	99	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	5.99	41
f10	2.57e−6	22	2.56e−6	24	7.16e−6	34	6.49e−7	80	6.17e−7	75	**1e−8**	**100**	5.5e−7	78	2.68e−6	0	7.21e+1	5	3.81e−3	1	1.17e−5	0	5.91e−1	0	**1e−8**	**100**	**1e−8**	**100**
f11	1.07e−8	90	1.02e−8	93	1.03e−8	91	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	1.12e−8	73	1.01e−8	95	**1e−8**	**100**	2.5e−8	11	2.01e−8	14	**1e−8**	**100**	**1e−8**	**100**
f12	**1e−8**	**100**	**1e−8**	**100**	1.0e−8	98	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	1.86e−4	98	**1e−8**	**100**	1.01e−8	97	1.0e−8	98	1.01e−8	87	1.33e−8	34	**1e−8**	**100**	2.01e−4	96
f13	7.48e−8	12	2.16e−7	6	3.5e−6	7	2.25e−8	33	2.96e−8	23	**1e−8**	**100**	5.33e−7	41	2.18e−3	0	5.15e−4	0	5.09e−7	0	5.58e−3	0	3.83e−3	0	1.0e−8	99	1.04e−8	95
f14	4.43e−7	4	3.96e−7	5	6.32e−7	7	5.19e−8	51	1.6e−7	55	**1e−8**	**100**	1.75e−7	50	1.45e−6	0	1.42e−5	0	7.37e−4	0	1.96e−6	0	1.84e−5	0	**1e−8**	**100**	1.1e−8	89
f15	1.59e+1	0	1.67e+1	0	1.62e+1	0	1.62e+1	0	1.64e+1	0	1.87e+1	0	1.57e+1	0	1.62e+1	0	1.58e+1	0	4.12e+1	0	1.88e+1	0	2.85e+1	0	**1.55e+1**	**0**	4.15e+1	0
f16	2.35	0	1.9	0	1.55	0	1.28	0	1.51	0	**1.31e−1**	**11**	3.02e−1	10	1.25	0	2.81	0	3.48	0	2.08	0	1.79	0	3.59	0	3.63	0
f17	8.38e−4	0	1.13e−3	0	2.45e−3	0	3.63e−3	0	5.33e−3	0	**1.07e−5**	**32**	9.15e−2	7	3.66e−2	0	3.29e−5	35	9.78e−2	1	9.09e−3	0	1.55e−2	0	4.22e−3	1	1.69	0
f18	6.3e−2	0	5.88e−2	0	4.8e−2	0	7.26e−2	0	7.74e−2	0	**3.74e−4**	**3**	1.34e−1	0	3.57e−1	0	7.25e−3	0	9.1e−2	0	1.38e−1	0	2.18e−1	0	2.52e−2	0	5.57	0
f19	6.92e−1	0	6.78e−1	0	5.91e−1	0	6.14e−1	0	6.63e−1	0	9.74e−1	0	9.42e−1	0	**4.44e−1**	**0**	6.62e−1	0	8.54e−1	0	6.07e−1	0	6.49e−1	0	1.01	0	1.74	0
f20	3.43e−2	59	1.1e−1	26	8.25e−2	36	1.14e−2	83	8.42e−3	87	3.12e−2	60	2.73e−3	96	1.14e−1	19	1.13e−6	76	**1e−8**	**100**	5.73e−1	0	**1e−8**	**100**	1.69e−1	0	2.39e−3	43
f21	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	7.07e−3	99	3.63e−5	99	1.77e−1	85	2.43e−1	82	**1e−8**	**100**	5.92e−1	62	**1e−8**	**100**	**1e−8**	**100**	**1e−8**	**100**	1.95e−2	99	**1e−8**	**100**
f22	3.46e−1	47	3.12e−1	51	3.82e−1	44	5.51e−1	18	5.72e−1	16	8.9e−1	1	8.98e−1	2	3.55e−1	49	1.53	0	5.8e−1	17	**1.27e−1**	**66**	3.68e−1	37	4.02e−1	21	3.08e−1	47
f23	3.26e−1	0	2.42e−1	0	2.53e−1	0	1.6e−1	0	2.07e−1	0	**3.9e−2**	**6**	5.35e−2	8	1.58e−1	0	3.34e−1	0	4.6e−1	0	1.72e−1	0	1.89e−1	0	4.03e−1	0	4.09e−1	0
f24	3.09e+1	0	3.01e+1	0	2.95e+1	0	3.17e+1	0	3.2e+1	0	3.91e+1	0	3.59e+1	0	**2.3e+1**	**0**	4.09e+1	0	5.08e+1	0	3.55e+1	0	3.2e+1	0	3.55e+1	0	5.86e+1	0

Table 5: Number of times the algorithm corresponding to the table's row is significantly better (Win), significantly worse (Loss), or indistinguishable (Tie) compared to the algorithm corresponding to a column, out of the 24 test functions. See Section 5.1 for an explanation of the usage of boldface and cell coloring.

	Fit			FitDiv			FitSqDiv			Div			SqDiv			Compass			Pareto			Uniform			rand/1/bin			rand/1/exp			best/1/bin			best/1/exp			ttb/2/bin			ttb/2/exp			total		
algorithm	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
Fit		—		3	18	3	5	16	3	3	14	7	4	13	7	5	10	9	7	9	8	7	13	4	7	12	5	9	11	4	13	9	2	10	11	3	**7**	**11**	**6**	9	9	6	89	156	67
FitDiv	3	18	3		—		3	19	2	2	15	7	4	14	6	5	9	10	7	9	8	6	14	4	8	12	4	9	11	4	13	9	2	10	12	2	**7**	**10**	**7**	9	9	6	86	161	65
FitSqDiv	3	16	5	2	19	3		—		3	15	6	4	14	6	5	9	10	7	8	9	7	14	3	7	13	4	10	10	4	**14**	**8**	**2**	10	12	2	6	12	6	9	9	6	87	159	66
Div	7	14	3	7	15	2	6	15	3		—		4	20	0	**6**	**10**	**8**	**9**	**10**	**5**	8	13	3	10	11	3	10	11	3	**14**	**9**	**1**	**12**	**10**	**2**	5	13	6	9	10	5	107	161	44
SqDiv	7	13	4	6	14	4	6	14	4	0	20	4		—		**6**	**10**	**8**	**9**	**10**	**5**	8	11	5	10	12	2	8	13	3	**14**	**7**	**3**	11	11	2	5	13	6	10	9	5	100	157	55
Compass	**9**	**10**	**5**	**10**	**9**	**5**	**10**	**9**	**5**	**8**	**10**	**6**	**8**	**10**	**6**		—		7	14	3	**10**	**9**	**5**	10	10	4	**11**	**9**	**4**	13	7	4	**12**	**7**	**5**	6	14	4	**11**	**10**	**3**	**125**	**128**	**59**
Pareto	8	9	7	8	9	7	9	8	7	5	10	9	5	10	9	3	14	7		—		9	9	6	**11**	**8**	**5**	10	8	6	13	5	6	10	7	7	4	12	8	9	9	6	104	118	90
Uniform	4	13	7	4	14	6	3	14	7	3	13	8	5	11	8	5	9	10	6	9	9		—		5	11	8	8	10	6	**14**	**7**	**3**	11	9	4	6	11	7	9	9	6	83	140	89
rand/1/bin	5	12	7	4	12	8	4	13	7	3	11	10	2	12	10	4	10	10	5	8	11	8	11	5		—		10	8	6	12	7	5	10	8	6	6	10	8	9	8	7	82	130	100
rand/1/exp	4	11	9	4	11	9	4	10	10	3	11	10	3	13	8	4	9	11	6	8	10	6	10	8	6	8	10		—		10	6	8	7	10	7	3	12	9	6	12	6	66	131	115
best/1/bin	2	9	13	2	9	13	2	8	14	1	9	14	3	7	14	4	7	13	6	5	13	3	7	14	5	7	12	8	6	10		—		8	9	7	4	8	12	9	7	8	57	98	157
best/1/exp	3	11	10	2	12	10	2	12	10	2	10	12	2	11	11	5	7	12	7	7	10	4	9	11	6	8	10	7	10	7	7	9	8		—		5	9	10	9	8	7	61	123	128
ttb/2/bin	6	11	7	7	10	7	6	12	6	6	13	5	6	13	5	4	14	6	8	12	4	7	11	6	8	10	6	9	12	3	12	8	4	10	9	5		—		9	14	1	98	149	65
ttb/2/exp	6	9	9	6	9	9	6	9	9	5	10	9	5	9	10	3	10	11	6	9	9	6	9	9	7	8	9	6	12	6	8	7	9	7	8	9	1	14	9		—		72	123	117

The algorithms Div, SqDiv, Compass and Pareto are significantly better than other algorithms most frequently overall. Compass has the most total wins by a sizable margin, and also against most algorithms individually. Div has the fewest losses overall, and it won about 2.5 times as often as it has lost. This is quite interesting, as Table 4 did not necessarily showcase any functions on which Div excelled, but this statistic clearly demonstrates the consistent performance of the algorithm across many problems. Div is significantly better than SqDiv on 4 problems, and never significantly worse, suggesting that SqDiv might be obsolete. On the other hand, FitSqDiv *is* significantly better than FitDiv on 3 functions. It seems that, when *only* using the squared diversity ratio as credit, the focus on stimulating diversity is too strong, but it still has merit when combined with a fitness-based metric.

The best/1/bin and best/1/exp have considerably many losses, signaling that these DEs are not very versatile. Pareto has more losses than the other AOS methods, as a result of its sub-par performance on some functions, as discussed earlier. It is interesting that none of the static DE configurations ever has the most wins against another algorithm, while the AOS methods never have the most losses, indicating that the AOS methods, Compass and Div in particular, are more versatile.

5.2 Fixed-target comparison

Figures 3 - 5 show the Empirical Cumulative Distribution Functions (ECDFs) of runtimes of all the algorithms on each of the 24 functions. The ECDFs show the proportion of targets hit within a specified budget, aggregated over all runs on each function. A total number of 51 targets $f_{\text{target}} = f_{\text{opt}} + \Delta f$ is used, where the Δf are spaced uniformly on a logarithmic scale in $[10^2, 10^{-8}]$. A large cross ($\times$) in the ECDFs indicates the median length of unsuccessful runs. In this case, unsuccessful runs always have the same length: the evaluation budget, i.e., $2 \cdot 10^6$. Runtimes to the right of the cross have at least one unsuccessful run, and have been computed artificially through simulated restarts [34]. The small dots to the right side of the plot indicate, for each algorithm, the fraction of targets which were hit at least once, across all independent runs.

In addition to the 15 algorithms shown in Table 4, we benchmark two other state-of-the-art optimizers using the exact same evaluation budget, number of independent runs, etc., as a reference:

- **BIPOP-CMA-ES** The Bi-Population Covariance Matrix Evolution Strategies [32] is a highly competitive variant of the well-known CMA-ES [31], leveraging two restart strategies: one with increasing population size and one with a small population size. The optimizer showed a considerable advantage over its competitors on the more difficult BBOB functions in [36]. We used the `pycma` package [33] to benchmark[1] the algorithm on BBOB with its default parameters.
- **U-AOS-FW** The U-AOS-FW is a Differential Evolution algorithm with adaptive operator selection resulting from a hyperparameter tuning experiment from the Unified AOS Framework [60], which outperformed other prominent AOS methods. For this reason, the U-AOS-FW is included in the performance comparison as the state-of-the art AOS method for DE. The most distinctive feature of the algorithm is the *recursive* probability matching mechanism [59], which uses the well-known Bellman equation, ubiquitous in the field of reinforcement learning [65]. The algorithm has nine different mutation methods at its disposal. U-AOS-FW is benchmarked[2]using the source code provided by the original authors with the hyperparameters listed in [60].

Note that the aim of this thesis is not necessarily to 'beat' these state-of-the art optimizers, but rather to devise a robust adaptation methodology that is capable of selecting an appropriate strategy for a given problem in real time, taking into account the exploration-exploitation dilemma. While Differential Evolution is under consideration in this work, the proposed approach could, if deemed successful, be applied to

[1]The source code for performing the BIPOP-CMA-ES benchmarking experiment is available at: `https://github.com/rickboks/benchmark-bipop-cmaes-bbob`.

[2]The source code for performing the U-AOS-FW benchmarking experiment is available at: `https://github.com/rickboks/Tune-AOS-bbob`.

other (state-of-the-art) optimizers to reinforce their robustness across problems with varying characteristics.

Results From the ECDFs, we can see that the BIPOP-CMA-ES generally needs fewer function evaluations to hit any target than any of the DE variants. In addition, it is able to consistently solve $f15$, where all DEs failed, $f16$, where only Pareto and Compass had a few successful runs, and $f18$, where only the Compass succeeded a few times. However, the BIPOP-CMA-ES is also outperformed on a few functions, most notably $f3$, $f4$ and $f20$.

On the 'simple' functions, which most algorithms solve consistently, we see that a static DE method often converges faster than the best AOS method. This is not surprising, as there is an overhead associated with finding the appropriate operator configurations. However, on the harder functions, where static configurations show varying performance depending on the problem, the AOS methods are much more consistent. As we also saw in Table 4, the Compass method seems to perform best; its performance is comparable to the best static DE algorithm on most functions, and on some functions, it is even (significantly) better than any of the static DEs. This is a great result for any AOS method: it signals that the AOS is at least able to successfully identify a good operator configuration for a given problem, and in some cases, it is even able to combine the strengths of different operators in such a way that the performance of the 'ensemble' is better than that of any of the individual configurations.

Most notable are the results on functions $f16$, $f18$ and $f23$, where the Compass DE is able to solve problems where the static DEs were only able to reach about 50% of the targets. The only problem where Compass has a considerable disadvantage compared to most other DEs, is $f22$.

The U-AOS-FW outperformed all other DE methods on functions $f17$ and $f18$, but performs rather poorly on a much larger number of functions. We find the performance of the other AOS methods, and Compass in particular, to be much more competitive.

5.3 Behavioral analysis

During the experiments described in Section 5, we collected behavioral data to gain an insight into the course of adaptation. Three types of data are collected at the end of each iteration:

- **Operator configuration activations:** In each iteration, we record for each configuration the number of individuals (out of M) which used that configuration in that iteration. Not only does this give us an insight into which operators are deemed most effective by each credit assignment scheme, but also into the course of adaptation: how quickly does the algorithm converge on a certain operator configuration, how many transitions occur during the search, etc. ?
- **Mean parameter values:** the *mean* values of the generated parameter values of F and Cr across the population are logged. Of course, it is interesting to observe the course of the adaptation of parameters on different problems, and see if they follow the canonical recommendations in literature [62, 54, 75]. More importantly, in our framework, the adaptation of parameters is based on the generated credit values. We are interested in discovering whether the usage of different credit assignment schemes results in different behavior of the employed parameter adaptation scheme. For example, are large values of F more common when using diversity-driven credit assignment schemes?
- **Population diversity:** the population diversity is computed as the mean of the Euclidean distances of each individual to the mean position of the population (see Eq. 12). It is especially interesting to study the course of the population diversity when different credit assignment schemes are used. Also, we might gain knowledge about how the population reacts when transitions in parameter values or operator configurations occur. The perspective can also be flipped: how does the strategy adaptation react when the population suddenly converges or diverges?

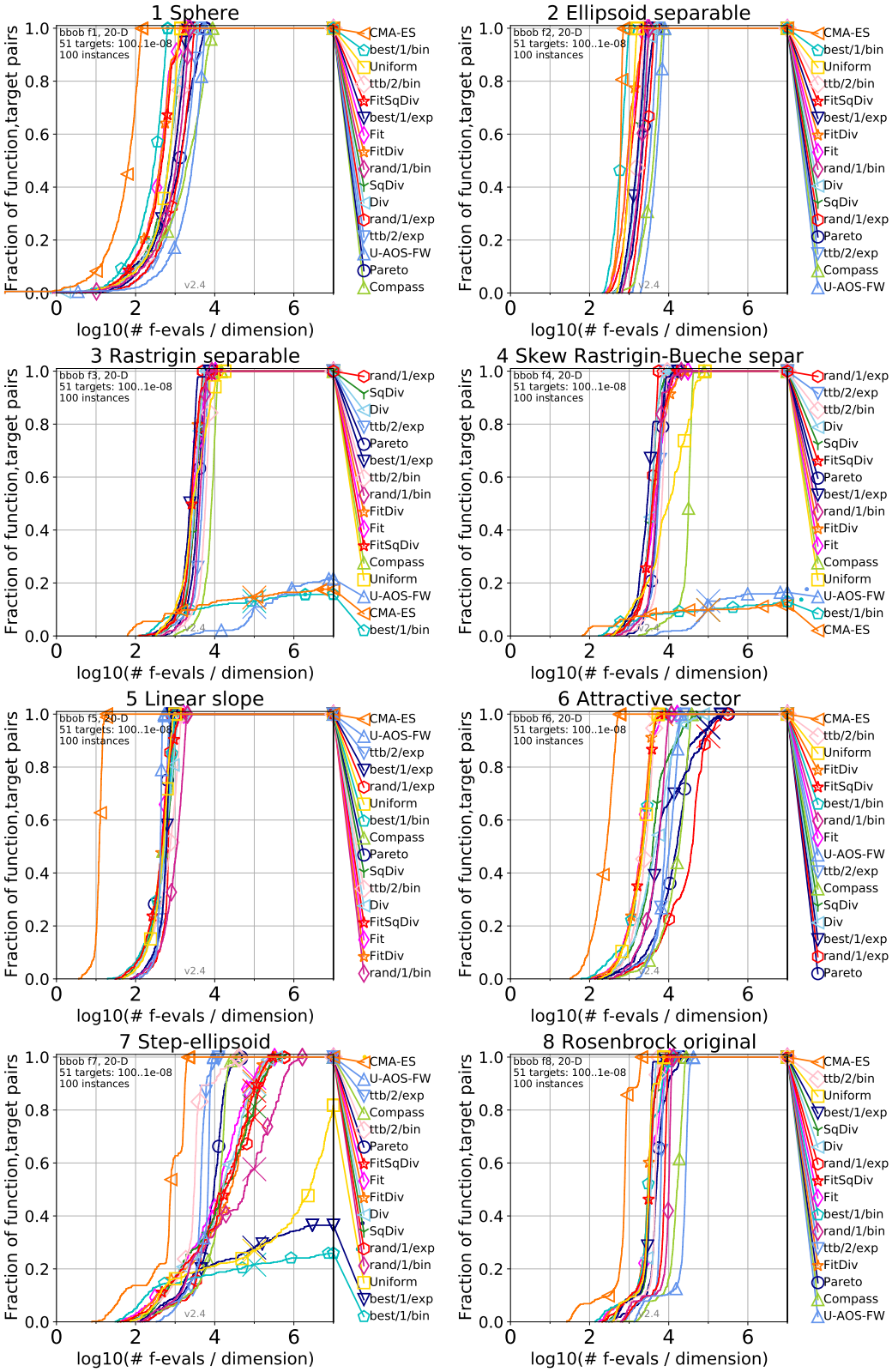


Figure 3: Empirical Cumulative Distribution Functions of runtimes for functions $f1 - f8$.

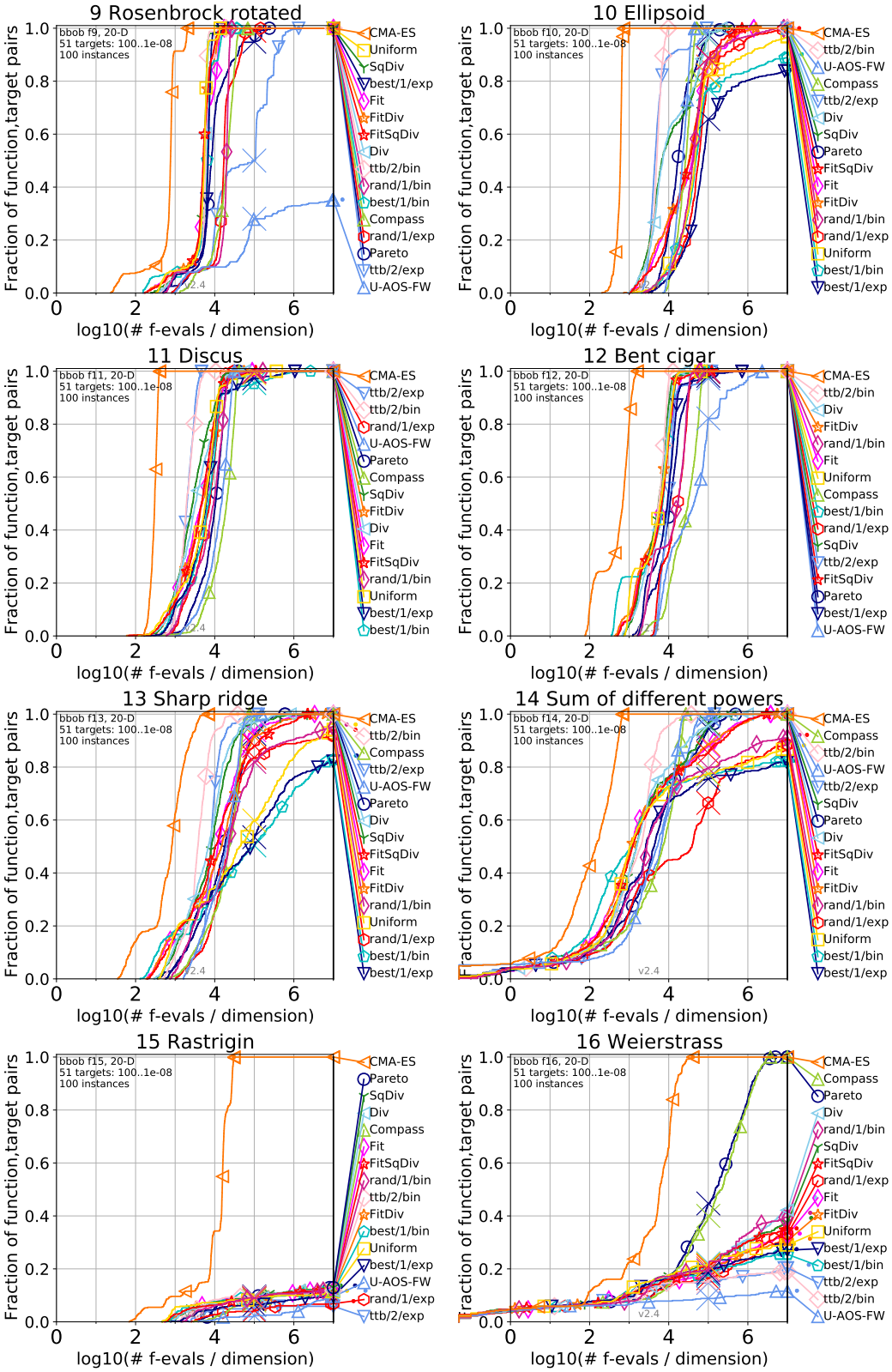


Figure 4: Empirical Cumulative Distribution Functions of runtimes for functions $f9 - f16$.

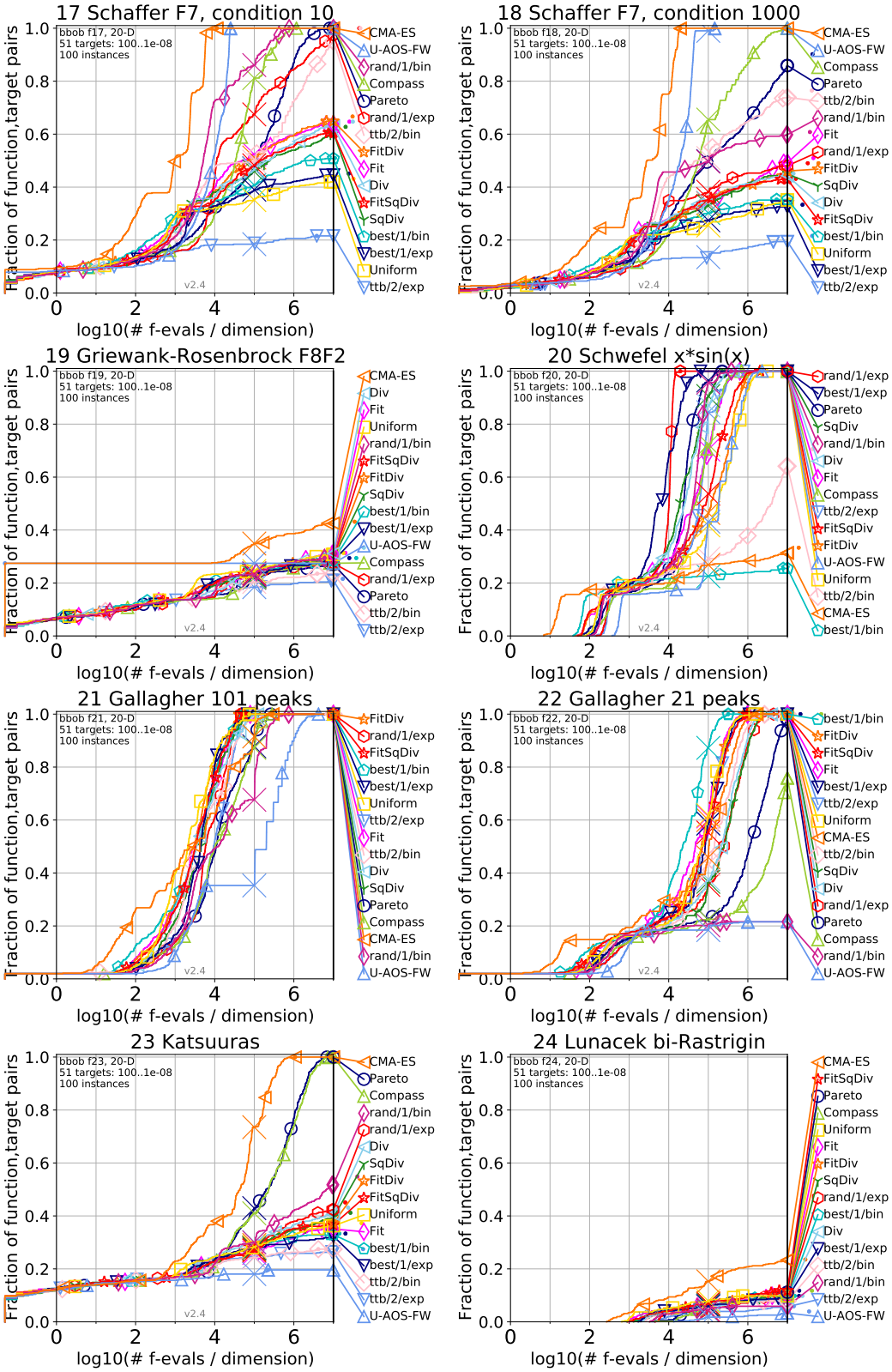


Figure 5: Empirical Cumulative Distribution Functions of runtimes for functions $f17 - f24$.

5.3.1 Operator configuration activations

Figure 6 shows a heatmap for each of the seven AOS methods (excluding 'Uniform', for obvious reasons), containing the proportion of activations for each of the six operator configurations, aggregated across all runs on each function. The proportions are obtained simply by adding all activations of each operator across all 100 runs together, and dividing by the combined number of activations, such that the proportions of activations of the six configurations sums to 1.

From the figure, it is immediately evident that binomial crossover is, in general, activated much more frequently than exponential crossover. On many functions, the $*$/exp configurations are only activated slightly more frequently than their minimal application probability $p_{\min}$. However, for all algorithms except Compass and Pareto, target-to-best/2/exp still is used frequently on $f15$, $f16$, $f23$ and $f24$.

Within the Compass algorithm, target-to-best/2/bin is activated extremely frequently, and it also consistently uses rand/1/bin (but to a lesser degree) on most functions. Compared to the other algorithms, Compass' activation rates are very similar across all functions. It is very interesting that on $f20$, on which the static target-to-best/2/bin DE performed very poorly (see Table 4), Compass activates this operator configuration much less frequently, and uses rand/1/bin prominently instead. The other six algorithms use all three of the $*$/bin configurations quite frequently, and the proportions differ drastically across functions. It is surprising that the rand/1/exp and best/1/exp configurations are not activated more frequently on $f20$, as they, as static DEs, performed significantly better than the $*$/bin variants on this function, as shown in Table 4. The only AOS method that did activate these configurations somewhat frequently is Pareto, which out of the AOS methods also performed best on $f20$.

The activation rates are quite similar between the three Fit$*$ algorithms, as are those between Div and SqDiv. The diversity-driven Div and SqDiv activate best/1/bin less frequently, which is understandable as this strategy is highly exploitative and will likely produce small credit values. Instead, these two algorithms utilize target-to-best/2/bin more prominently. Pareto uses the rand/1/bin strategy much more frequently than other credit schemes, and also utilizes exponential crossover relatively often on some functions.

Overall, it is clear that the selected credit assignment scheme has a real impact on the activation of operators. The presented results do raise some questions, which will be addressed in the remainder of this thesis:

- Since exponential crossover is activated so infrequently, can we assume that it is unnecessary to include it in the configuration space, and that excluding it will result in an improved performance?
- Should other operators be included in the configuration space to improve the performance or robustness of the AOS methods?
- Do the AOS methods converge on one operator configuration and stick with it during the entire run, or do 'transitions' between operator configurations occur commonly? For example, does Compass converge on target-to-best/2/bin in approximately 75% of the runs and on rand/1/bin in 10% of the runs, or do most runs consist of transitions between these two configurations (at certain stages)?

5.3.2 Transitions between operator configurations

Figure 6 gives a good insight into which operators are most prominent on certain functions, but we are unable to extract knowledge about transitions between operator configurations *within a run*. To address this, we measure the number of transitions within each (restarted) run.

A transition is considered a change in the operator configuration which has the majority of activations during a streak of at least 10 iterations. The initial transition, i.e., the first time any operator configuration obtains a majority, is also counted. During the tuning phase (see Section 4), all the AOS methods converged on the Adaptive Pursuit method for probability assignment, which results in rapid and pronounced transitions once the quality of the current best configuration is surpassed. Due to this, prolonged

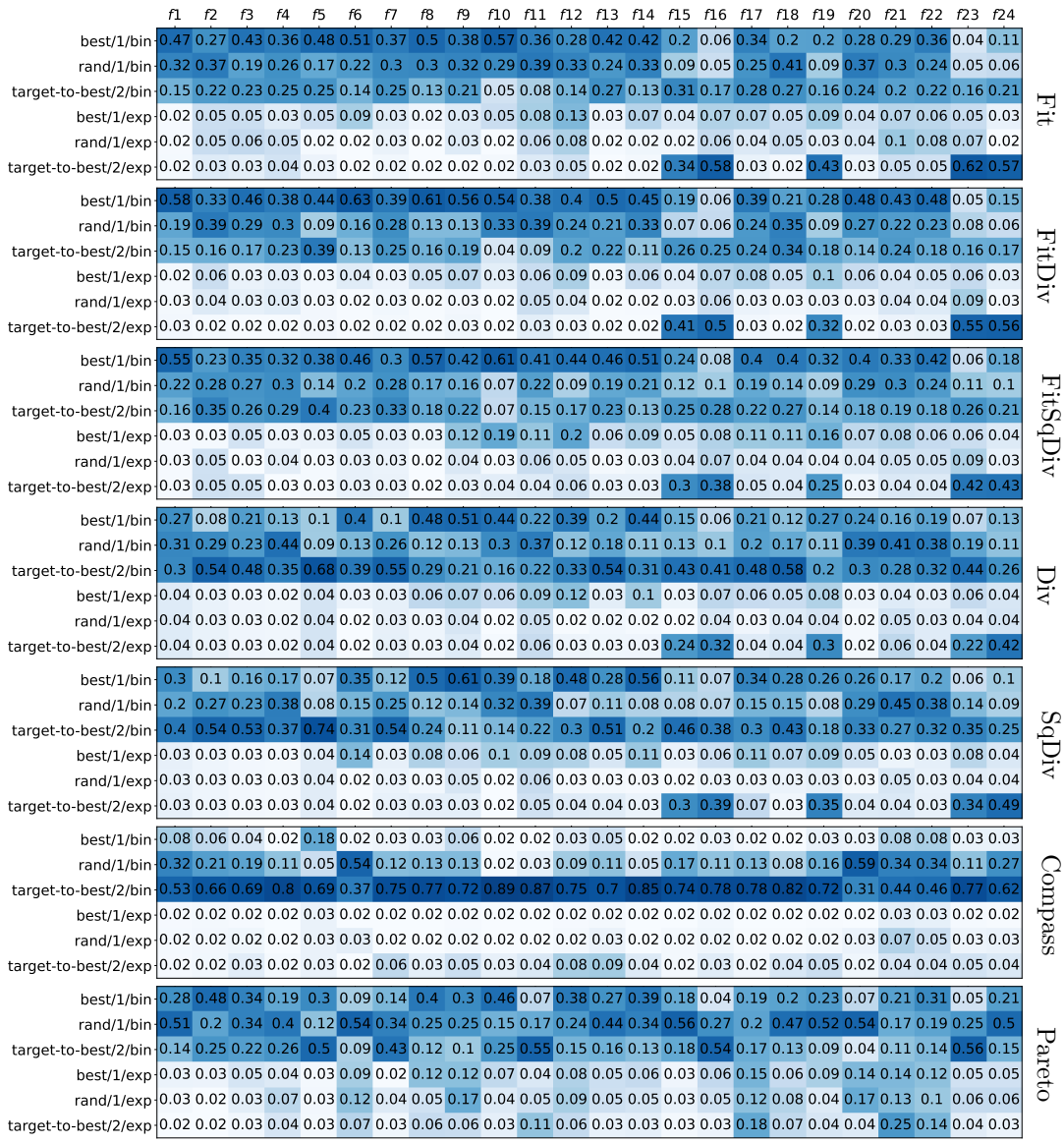


Figure 6: Activation rates of each operator configuration for each of the seven AOS methods, aggregated over the 100 runs on each function.

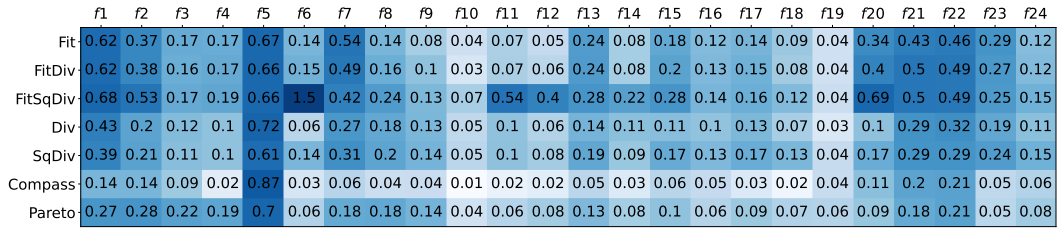


Figure 7: Number of transitions per 100 iterations for each AOS method, aggregated over all the (restarted) runs on each function.

periods where no configuration has a large majority of applications are very uncommon. All AOS methods allow a single operator configuration to have an application probability of at least 85% due to the high values of γ (see Table 3). Therefore, we consider a configuration to have the current majority when it is responsible for at least $\frac{2}{3}$ of the applications. To ensure that the majority is not a result of randomness, short streaks consisting of fewer than 10 iterations are ignored. To account for different run lengths, which can vary wildly across different functions, we study the number of transitions per 100 iterations, averaged over all (restarted) runs on a function. Figure 7 displays this metric for the seven AOS methods and all 24 test functions.

The Fit and FitDiv algorithms have very similar transition rates to each other. FitSqDiv, on the other hand, has much higher transition rates on some problems, most notably on $f6$, $f11$, $f12$, $f14$ and $f20$. It is possible that these frequent transitions are caused by the increased reward that can be obtained through exploration, and the increased 'penalty' that is associated with exploitation. Perhaps, the algorithm is constantly oscillating between exploration and exploitation, switching to exploration when exploitation is no longer rewarding enough (i.e, the rewards are not large enough to compensate for the penalty) and vice-versa. The fact that FitDiv has nearly identical transition rates (and similar activation rates, as shown in Figure 6) to Fit might indicate that its reward for exploration and the penalty for exploitation are not a strong enough to alter the algorithm's behavior. Compass has considerably lower transition rates on most problems, compared to all other algorithms. This could either mean that on these problems Compass is able to accurately converge on the optimal configuration without having to switch, or conversely, that it is unable to react appropriately when a change of strategy is needed. We find the first explanation more likely, due to its good performance results. It should be noted that the relatively high transition rates on some functions, most notably $f5$, are largely caused by the fact that the problems are solved in very few iterations, making the initial transition have a large impact on the transition frequency.

5.3.3 Generated parameter values

The employed parameter adaptation method generates values of F and Cr for each individual in each iteration, adapting the parameters in an attempt to maximize the credit values. To gain an insight into the generated parameter values, Figure 8 shows the mean parameter values of each AOS method aggregated for each test function. It is important to realize that the these mean parameters are indeed aggregated across all runs of a problem, so the presented results could be deceptive. The parameter values could vary wildly across different runs, and they can even fluctuate within a single run, especially considering that the values are adapted for each operator configuration individually.

The literature states that F should not be too small, as it can cause premature convergence [75]. A value of 0.6 is recommended as a starting point, and it should be increased if it is suspected that the algorithm converges on a local optimum [29].

The average values of F per function are very similar between all algorithms except Compass and Pareto. These two algorithms have an average value of approximately 0.85 across all functions. A possible cause for the discrepancy is that Compass and Pareto are the only credit assignment schemes that can assign a positive credit value to an individual with a deteriorated fitness value, meaning that these deteriorated individuals, too, can influence the adaptation of parameters. However, it is unclear why all their mean values of F are approximately 0.85.

All other algorithms have a more varied mean value of F depending on the function, but on most functions the values are around 0.6, which conforms to the recommendation given in [29]. On some functions, the mean values are considerably lower, around 0.4. The functions in question are, however, also the functions where these algorithms were very unsuccessful, and have not hit the final target a single time. Div and SqDiv have a slightly higher value than the Fit$*$ algorithms some functions, possibly due to their diversity-driven nature. The differences are, however, relatively small.

The general consensus in the literature is that low values of $Cr \in (0, 0.2]$ are benefi-

cial for separable functions, while high values of $Cr \in [0.9, 1]$ are more successful when solving non-separable functions [54]. Looking at the separable functions ($f1 - f5$), the mean Cr values are overall indeed rather low. Some algorithms on $f1$ and $f2$, and all algorithms on $f5$ have a mean value of Cr around 0.5. This likely has nothing to do with the optimal value of Cr, but rather with the simplicity of these functions; they are solved so quickly that the parameter adaptation scheme has no time (or pressure) to find the optimal value of Cr (or F). A clear exception to the rule of low Cr values is Compass' high mean value of 0.74 on $f4$. In general, Compass' and Pareto's mean values of Cr are again quite different from the other algorithms. The algorithms Div and SqDiv have a considerably lower value of Cr on a number of problems, especially on $f6$. We speculate that the reason is that on these functions, the probability of finding an improved solution away from the current population is larger when only a small number of components is mutated at a time.

The convention of high Cr values on non-separable functions is not always followed. Again, the problems on which the values deviate from the convention the most ($f15$, $f16$, $f23$, $f24$) are also problems on which the 'violating' algorithms were very unsuccessful, so it is likely that the parameter adaptation scheme was unable to do any meaningful adaptation due to a lack of progress during the search. Apart from the discrepancy on $f4$, Compass follows the convention of Cr values somewhat closely.

F

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16	f17	f18	f19	f20	f21	f22	f23	f24
Fit	0.56	0.58	0.73	0.71	0.76	0.6	0.55	0.62	0.61	0.68	0.66	0.64	0.57	0.66	0.41	0.34	0.62	0.61	0.41	0.64	0.61	0.59	0.34	0.32
FitDiv	0.58	0.59	0.73	0.72	0.77	0.6	0.57	0.64	0.64	0.7	0.68	0.65	0.59	0.68	0.38	0.35	0.64	0.62	0.46	0.62	0.59	0.59	0.38	0.33
FitSqDiv	0.59	0.58	0.74	0.72	0.78	0.58	0.57	0.64	0.65	0.72	0.69	0.65	0.6	0.7	0.43	0.38	0.66	0.67	0.52	0.63	0.62	0.61	0.4	0.37
Div	0.62	0.64	0.77	0.77	0.79	0.7	0.53	0.65	0.66	0.69	0.65	0.65	0.59	0.67	0.47	0.43	0.62	0.59	0.47	0.74	0.69	0.66	0.46	0.41
SqDiv	0.61	0.65	0.77	0.77	0.8	0.69	0.53	0.66	0.69	0.7	0.67	0.64	0.59	0.7	0.41	0.38	0.66	0.64	0.46	0.72	0.7	0.67	0.43	0.37
Compass	0.86	0.83	0.86	0.87	0.81	0.86	0.86	0.86	0.86	0.86	0.87	0.86	0.85	0.86	0.86	0.86	0.85	0.85	0.86	0.85	0.85	0.85	0.86	0.86
Pareto	0.83	0.8	0.85	0.85	0.81	0.84	0.84	0.81	0.83	0.86	0.88	0.84	0.83	0.86	0.84	0.86	0.84	0.84	0.85	0.83	0.84	0.84	0.86	0.85

Cr

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16	f17	f18	f19	f20	f21	f22	f23	f24
Fit	0.56	0.6	0.28	0.27	0.56	0.77	0.72	0.83	0.86	0.93	0.91	0.9	0.8	0.9	0.24	0.22	0.83	0.91	0.43	0.39	0.57	0.61	0.25	0.22
FitDiv	0.56	0.59	0.25	0.26	0.55	0.8	0.71	0.82	0.85	0.93	0.91	0.89	0.78	0.9	0.2	0.22	0.8	0.9	0.5	0.47	0.6	0.61	0.28	0.22
FitSqDiv	0.55	0.61	0.25	0.24	0.55	0.75	0.69	0.81	0.84	0.92	0.89	0.87	0.77	0.88	0.26	0.23	0.75	0.84	0.58	0.44	0.56	0.59	0.28	0.24
Div	0.38	0.31	0.14	0.12	0.46	0.15	0.58	0.69	0.75	0.91	0.88	0.78	0.67	0.79	0.15	0.2	0.4	0.79	0.47	0.13	0.29	0.32	0.34	0.19
SqDiv	0.39	0.32	0.13	0.13	0.47	0.21	0.55	0.71	0.75	0.92	0.89	0.82	0.7	0.81	0.17	0.22	0.49	0.78	0.46	0.16	0.27	0.31	0.33	0.22
Compass	0.61	0.34	0.3	0.74	0.49	0.8	0.82	0.8	0.81	0.84	0.83	0.83	0.84	0.83	0.81	0.79	0.81	0.85	0.83	0.75	0.64	0.64	0.8	0.81
Pareto	0.48	0.32	0.21	0.26	0.48	0.55	0.81	0.69	0.49	0.88	0.89	0.75	0.8	0.84	0.72	0.74	0.26	0.71	0.67	0.48	0.2	0.24	0.74	0.58

Figure 8: Mean values of F and Cr for each AOS method, aggregated over the 100 runs on each function.

5.3.4 Single-run analyses

So far, we analyzed data aggregated over many runs. To gain a better insight into the behavior of the algorithms within single runs, we create plots that show in detail how the algorithm adapts during the run. Each 'restart' is considered a separate run here. As there are, across all 7 algorithms and each function, about 10^5 (restarted) runs, there is no way to analyze them all individually. We select a few interesting cases based on the results of previous analyses in this thesis. In the following plots, we show the activations of operator configurations over time, as well as the population diversity and the mean values of F and Cr across the population in each iteration.

The AOS with Compass credit was able to solve $f16$ eleven times, where no other algorithm was able to ever hit the final target. We therefore plot the successful runs of Compass on $f16$ to see which adaptation strategy led to its success. Most of the successful runs look similar to Figures 9a and 9b: they start with target-to-best/2/bin, switch to rand/1/bin somewhere in the middle, and then switch back to target-to-best/2/bin. A transition to rand/1/bin is always associated with a sharp decline in population diversity. Considering the highly rugged and multi-modal landscape of $f16$, it is possible that the rand/1/bin strategy was able to discover a solution near the global optimum through its exploratory property, after which the rest of the population quickly followed, explaining the sudden drop in population diversity. In Figure 9a, we also see a sharp increase of Cr when rand/1/bin becomes dominant, as a result of the configuration-wise parameter adaptation.

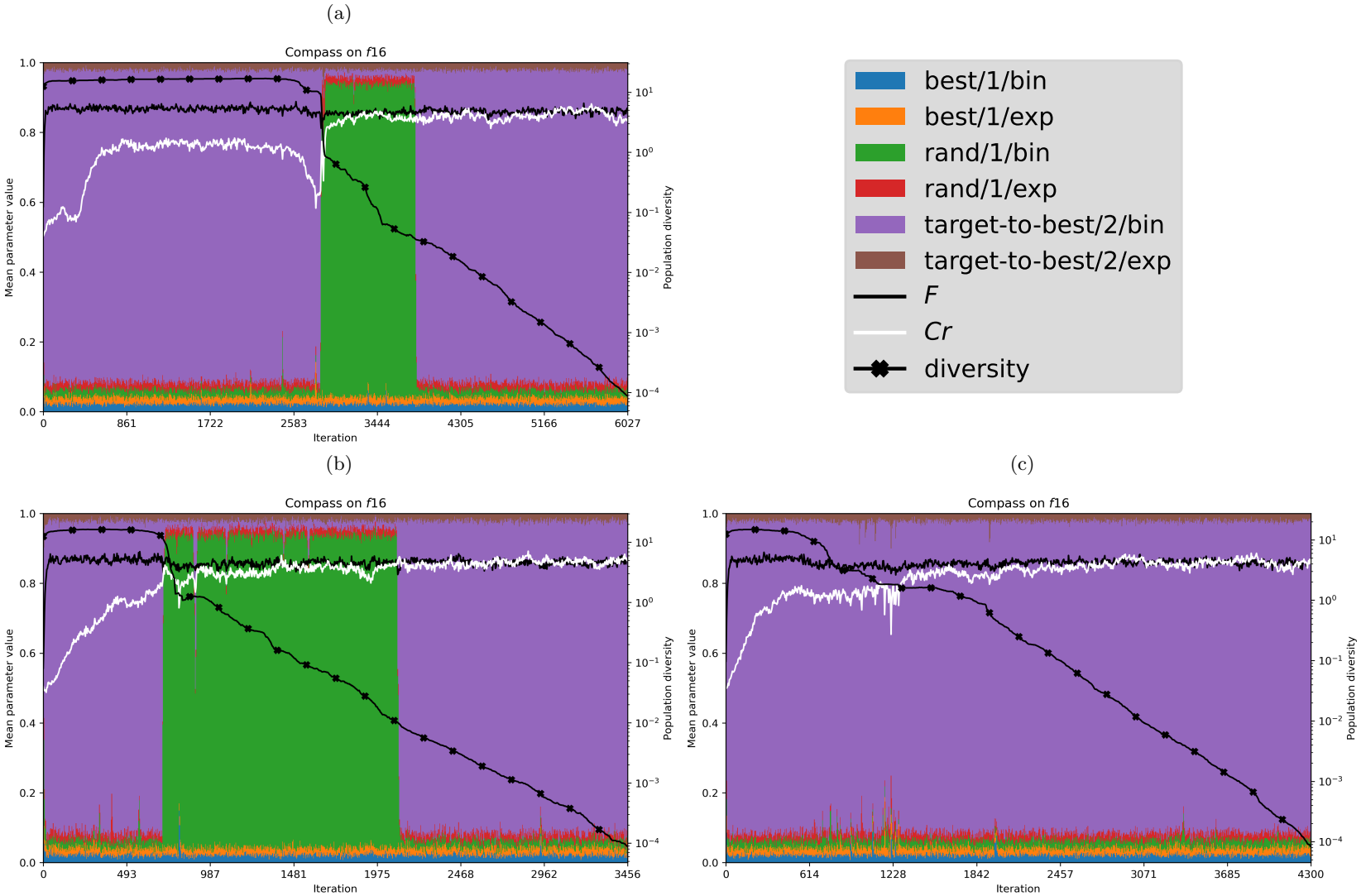


Figure 9: Plots of three successful runs of Compass on $f16$, showing the adaptation behavior.

Four runs, however, look more similar to Figure 9c, where no longstanding transitions between operator configurations occur, and target-to-best/2/bin is mostly used throughout the entire run. This is very interesting, as target-to-best/2/bin (or any other algorithm besides Compass) by itself was not able to solve $f16$. This could mean that:

- the non-dominant operator configurations, although they are used by only about 2% of the population each, still have a vital role in the search process.
- the Compass credit is better at balancing exploration and exploitation in the parameter adaptation, compared to the 'Fit' credit.

Overall, the values of F and Cr are comparatively very stable with Compass (also on other functions, not shown here). On most functions, both F and Cr converge to a value around 0.85. Another function on which Compass was the able to solve problems which the static DEs could not, is $f23$, where the course of adaptation (not shown here) is very similar to that of $f16$. Interestingly, $f16$ and $f23$ are also similar problems, in the sense that they both have repetitive and rugged fitness landscapes. Pareto was also able to solve $f23$ a few times, and the adaptation of both the configurations and the parameters looks very similar to that of Compass in the successful runs on this function.

As discussed in Section 5.3.2, the FitSqDiv credit scheme had comparatively huge transition frequencies on several functions. We plot one 'typical' successful run each of $f6$, $f11$, and $f12$, where frequencies of transitions were comparatively high, in Figure 11. Although we attempted to select some representative runs, the course of adaptation across different runs does vary greatly. The main operator configurations the algorithm is transitioning between are target-to-best/2/bin, rand/1/bin and best/1/bin. There are major fluctuations in the population diversity and the mutation rate, signaling that the algorithm is indeed actively trying to balance the fitness improvements and the diversity improvements, as hypothesized in Section 5.3.1. Operator configuration transitions can be associated with huge changes in the mean values of F and Cr. However, this does not necessarily mean that the other configuration prefers different parameter values. Sometimes, the newly dominant configuration has not had many prior activations, preventing it from adapting the parameter values appropriately. Once it is dominant for a longer streak of iterations, the parameters may quickly adapt to different values, which may or may not be similar to that of other configurations. A great example of this behavior is shown in Figure 11b. When the initial transition to rand/1/bin occurs, the mean crossover rate drops from ≈ 0.9 to ≈ 0.5. However, the parameter adaptation of rand/1/bin now has a lot of performance data to work with, and as a result the Cr rapidly rises back to ≈ 0.9. When the next transition to rand/1/bin occurs, the mean Cr values are around this value.

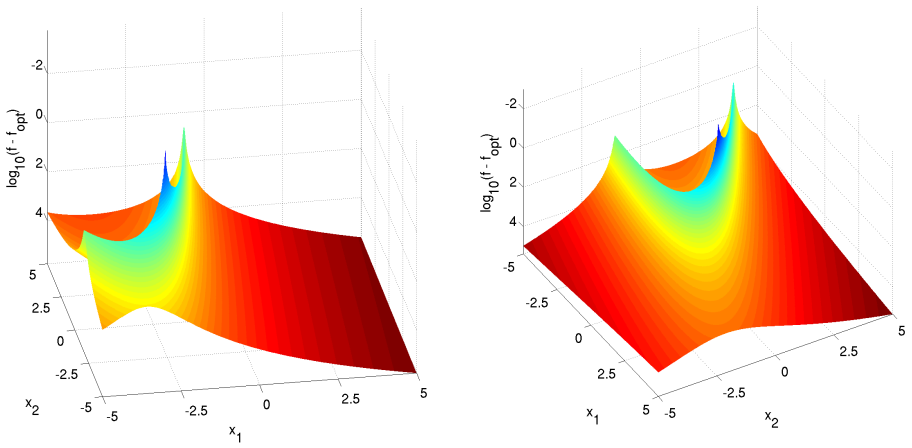


Figure 10: Fitness landscape of BBOB's Rosenbrock problems, original ($f8$, left) and rotated ($f9$, right), in 2 dimensions. The dark blue peak is the optimum. Figures are taken from [23].

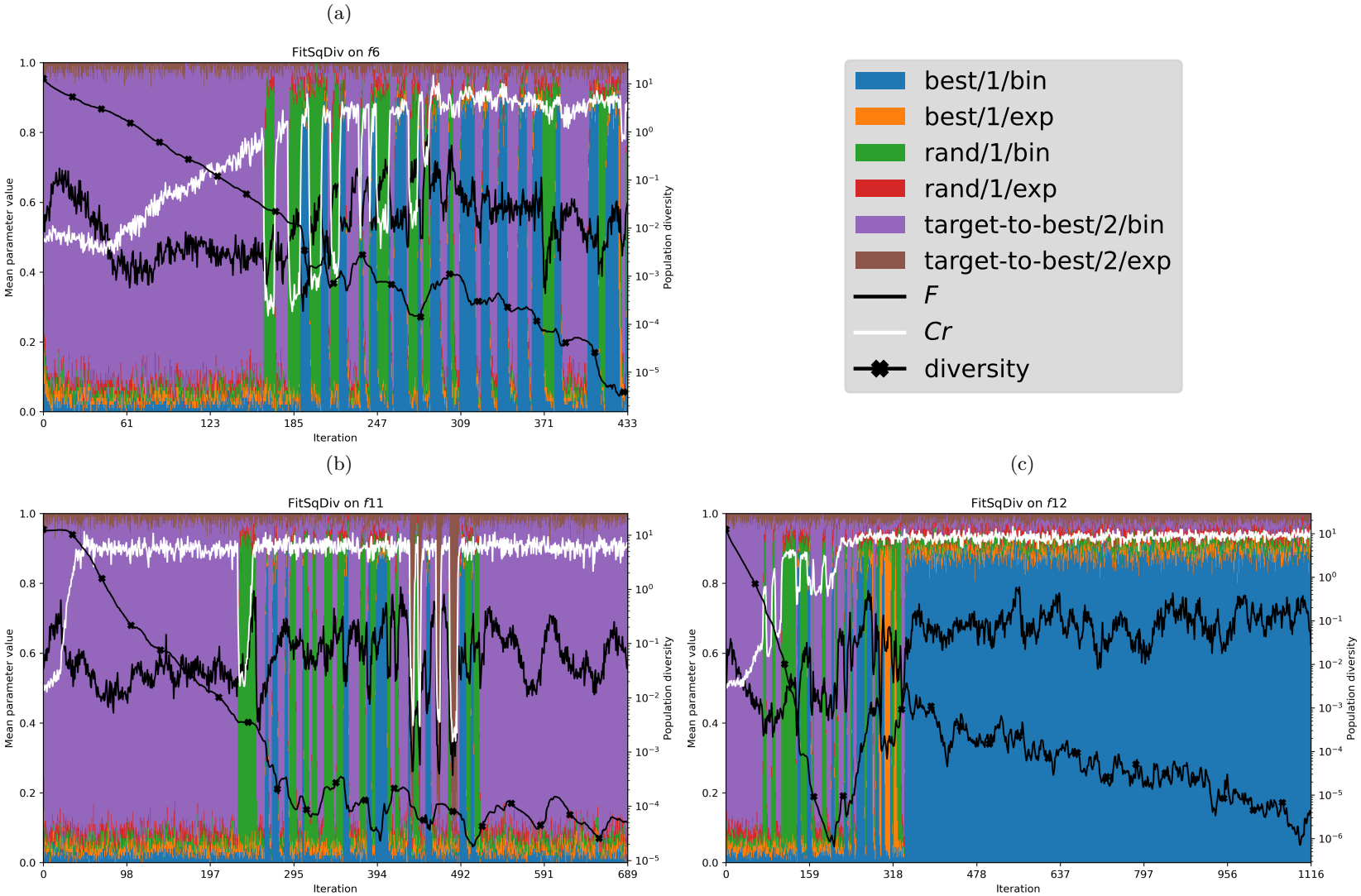
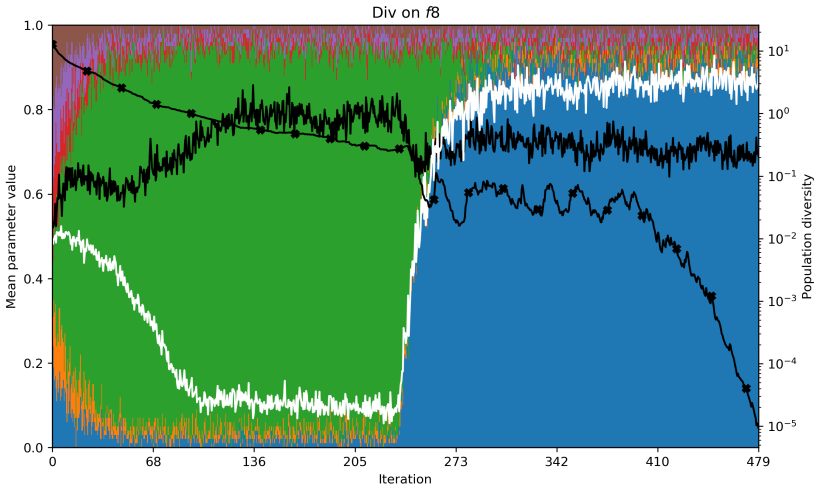
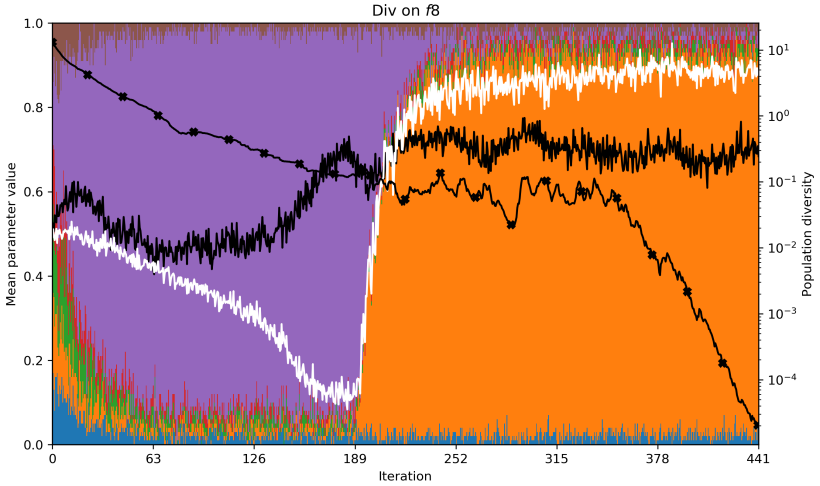


Figure 11: Behavioral plots of three successful runs of FitSqDiv on three different functions, showcasing the high transition rates.

(a)



(b)



(c)

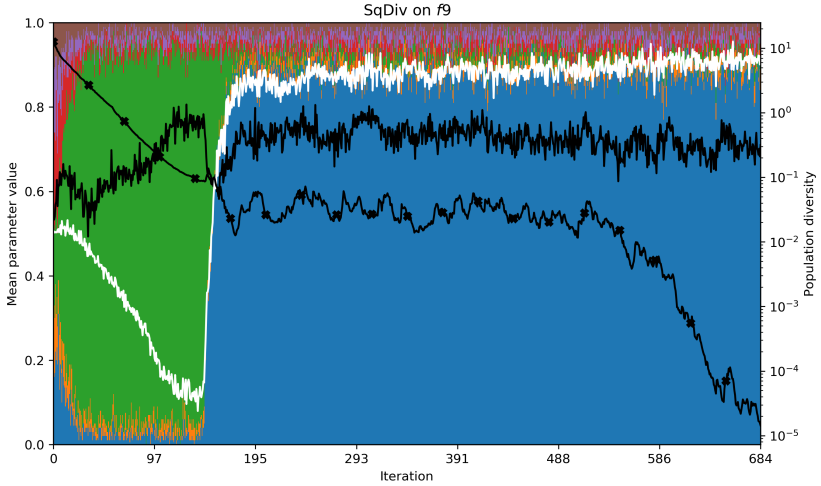


Figure 12: Plots of three selected successful runs of Div and SqDiv on BBOB's two Rosenbrock functions: $f8$ and $f9$, showing the adaptation behavior.

Finally, we want to highlight an interesting pattern occurring on functions $f8$ and $f9$. Both of these functions are the Rosenbrock function, but $f9$ is a rotated variant. The Div and SqDiv algorithms show very a similar pattern on about half of the successful trials on these functions. Figure 12 shows three such runs. An exploratory strategy (mostly rand/1/bin or target-to-best/2/bin) is dominant in the beginning, and the parameter adaptation converges on a low value of $Cr \approx 0.1$. Most commonly after about 200 iterations, there is a transition to an exploitative strategy (best/1/bin or best/1/exp), and the mean value of Cr quickly shifts to approximately 0.9. Then, the population diversity drops slightly and stabilizes again for a while (with a small oscillations). Finally, the population diversity drops quickly before the algorithm solves the problem. If we look at the fitness landscape of the Rosenbrock problems in Figure 10, we see that this switch in strategy makes sense intuitively: the algorithm needs to first explore the search space to find the ridge, and then find the optimum by following the ridge through exploitation. We expect that the initial low crossover rate is a result of the algorithm aiming for high diversity ratios, which are most easily obtained by modifying few components at a time, as discussed in Section 5.3.3. Then, after switching to best/1/$*$ when the thin ridge is found, exploitation is the only way to improve, and the high Cr accelerates the convergence. These example runs also clearly demonstrate the merit of configuration-wise parameter adaptation, as different operator configurations can converge on drastically different parameter values. In Figure 12, it is also evident that Div and SqDiv use a much smaller value of β than other algorithms, as more iterations are needed before a configuration reaches an application probability of $p_{\max}$.

6 Comparing configuration spaces

Thus far, we experimented with a single 'configuration space' consisting of three mutation operators and two crossover operators. Although this set of operators seems sensible due to the diverse set of characteristics and strengths it possesses, it is possible that better results are obtained with different configuration spaces. The results shown in Figure 6 show that exponential crossover is hardly used on most of the test functions. This observation raises the question if the performance of the AOS could be enhanced by omitting exponential crossover from the configuration space. Further, it is interesting to experiment with different ensembles of operators and configuration spaces of various sizes, and observe the performance differences.

The current literature has examples of AOS methods using small configuration spaces, such as SaDE [57] and CoDE [72], using respectively 2 and 3 strategies, but also of methods using more strategies, such as the original Compass [47], which uses 6 crossover methods, U-AOS-FW [60] which uses 9 mutation operators, and Blacksmith [46], which utilizes huge number of 307 different crossover operators. The configuration space in current literature in most commonly compiled using intuition, i.e., by selecting a set of operators with diverse characteristics. However, the number of operators that is considered is rarely justified. We aim to gain some insight into the performance impact that the composition of the configuration space can have, by benchmarking one AOS method with a large number of different configuration spaces.

In detail, we consider 6 mutation operators (see Section 2.1) and 2 crossover operators (see Section 2.2) for this experiment. By considering all possible sets of mutation operators with sizes $1, 2, \ldots 6$, we obtain 63 different options. There are only three different sets of crossover operators: 1) only binomial, 2) only exponential, and 3) both binomial and exponential. A total number of 189 operator configurations is then obtained by combining all possible sets of mutation operators with all possible sets of crossover operators: $63 \cdot 3 = 189$. The smallest configuration spaces have only $1 \cdot 1 = 1$ operator configurations at their disposal, while the largest has $6 \cdot 2 = 12$.

These configuration spaces are tested in combination with the Compass-based DE, as it showed the best performance in our experiments described in Section 5.1. Note that the 'static' DEs (those with only 1 operator configuration) are different from those we experimented with in Section 5.1: while the Compass credit metric has no effect on the adaptation of operators (since there is only 1), it does control adaptation of parameters.

An identical benchmarking procedure as the one described in Section 5 is followed,

using the same parameters (see Table 3). The 189 algorithms with different configuration spaces are then ranked as follows. Conforming to the benchmarking procedure of BBOB [22], we consider 51 target precisions, uniformly spaced on a logarithmic scale between 10^2 and 10^{-8}. For each of these targets and each algorithm-function pair, we compute the Expected Running Time (ERT) [53, 3], which captures the expected number of function evaluations that an algorithm will need to reach a target for the first time. It is computed as [22]:

$$\mathrm{ERT}(f_{target}) = \frac{\#\mathrm{FEs}(f_{\mathrm{best}} \geq f_{\mathrm{target}})}{\#\mathrm{succ}}, \tag{47}$$

where $\#\mathrm{FEs}(f_{\mathrm{best}} \geq f_{\mathrm{target}})$ is the number of function evaluations performed across all runs while the target in question was not yet hit, and #succ is the number of trials in which the target was successfully reached. If the target in question was not hit a single time in any run, the ERT is set to infinity. We rank the 189 algorithms on each of the 24 functions, according to the ERT values on the 51 targets. One algorithm (`A`) is considered better than another algorithm (`B`) on a certain function, if:

- algorithm `A` has a greater number of finite ERT values than algorithm `B` on the function in question; or
- the algorithms have an equal number of finite ERT values, but on the first target (starting from the hardest one) on which the ERT values are not equal, that of algorithm `A` is lower; or
- the algorithms have the same ERT values on *all* targets, and algorithm `A` has a lower mean best-reached objective function value. Note that it is normally extremely unlikely that the ERT values of two algorithms are the same across all 51 targets, except when both algorithms have hit none of the targets across all runs (making all ERT values infinite). We implemented this third criterion because some algorithms were unable to hit a single target on $f10$.

BBOB divides its 24 test functions into five function groups, where the members of each group share similar characteristics in terms of separability, multi-modality, conditioning, etc. The five function groups are outlined in Table 6. A complete description of the function groups and the 24 functions can be found in [23].

Table 6: BBOB function groups [23].

Group	Description	Functions
1	Separable functions	1, 2, 3, 4, 5
2	Functions with low or moderate conditioning	6, 7, 8, 9
3	Functions with high conditioning and unimodal	10, 11, 12, 13, 14
4	Multi-modal functions with adequate global structure	15, 16, 17, 18, 19
5	Multi-modal functions with weak global structure	20, 21, 22, 23, 24

To make the results more general and more easily interpretable, we aggregate the algorithms' ranks over each function group. Figures 13 through 18 show the mean ranks for each configuration space in a way that is meant to be easy to interpret, considering the large number of algorithms. Each column represents an operator, and each row, shown as a thin line, a configuration space, which are ranked from top to bottom (best to worst). When a cell is colored, it indicates that the operator is included in the corresponding configuration space. The rightmost column does not represent an operator, but instead indicates the size of each configuration space, where a darker shade of gray indicates a larger configuration space. The figures allow us to see if certain (combinations of) operators or certain sizes of configuration spaces are associated with relatively high or low performance. Table 7 shows the top-5 highest ranked algorithms for each function group and in total, where 'target-to-*p*best' is abbreviated as 'tt*p*b', target-to-best as 'ttb', and 'target-to-rand' as 'ttr'. Of course, the top-5's are also shown in Figures 13 – 18, but it can be difficult to read exactly which operators are activated in a particular configuration space.

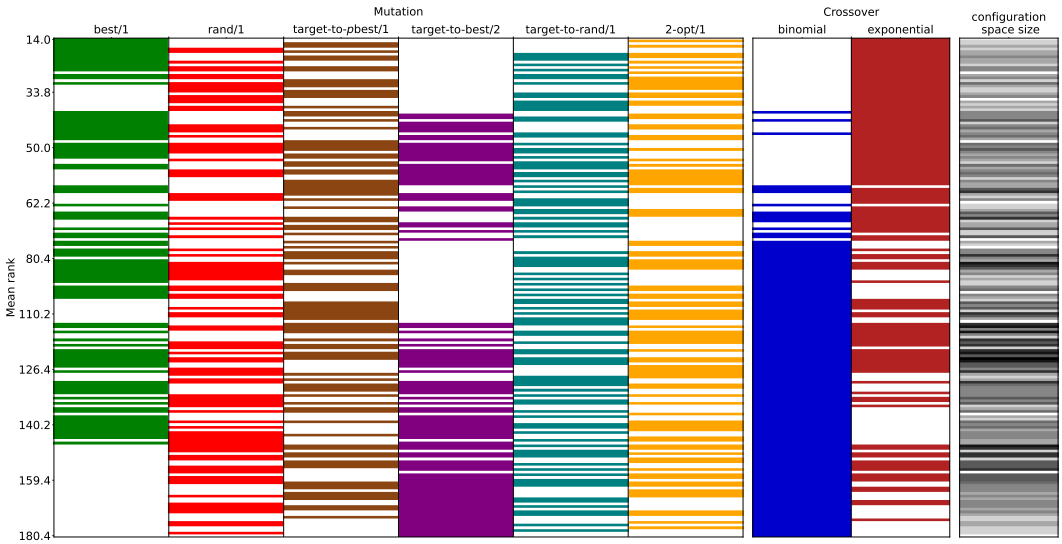


Figure 13: Ranking of DE configuration spaces in function group 1.

Function group 1 In function group 1 (see Figure 13), there is an evident preference for exponential crossover. Clearly, in addition to including exponential crossover, it is also beneficial to exclude binomial crossover from the configuration space. The functions of this function group should be solved quite easily by the Compass DE (See Table 4), and including binomial crossover in the configuration space likely extends the overhead of finding the optimal operator configuration, and the overhead of finding appropriate parameters for each configuration. Probably for the same reason, small configuration spaces, in general, seem to show better performance. The highest-ranked configuration space in fact has only one operator configuration: best/1/exp, where no AOS takes place at all. Interestingly, the target-to-best/2 mutation scheme is not included at all in the highest-ranked mutation schemes, and is associated with the worst-performing configuration spaces, even though it was activated very frequently in our previous experiments (see Figure 6). All of the most successful configuration spaces include the best/1 mutation scheme, which makes sense considering the simple nature (unimodal and separable) of three of the functions in this function group, on which a highly exploitative strategy is very effective.

It is quite surprising that, in our previous experiment, binomial crossover was activated most frequently on the functions of function group 1, rather than exponential crossover, even though Figure 13 shows that better performance is obtained when excluding binomial crossover from the configuration space.

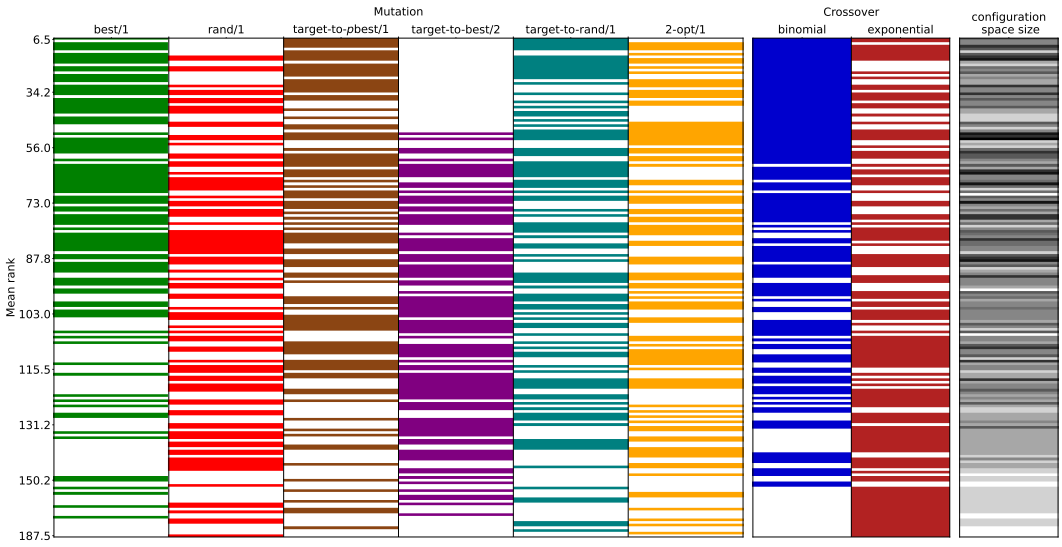


Figure 14: Ranking of DE configuration spaces in function group 2.

Function group 2 In contrast to function group 1, the inclusion of binomial crossover in the configuration space is clearly advantageous in function group 2 (see Figure 14). Most of the best-performing configuration spaces use both binomial and exponential crossover, and large configuration spaces generally perform better than small ones. Again, target-to-best/2 is excluded from the best-performing configuration spaces. The mutation operators best/1, target-to-*p*best and target-to-rand/1 are most often associated with high performance.

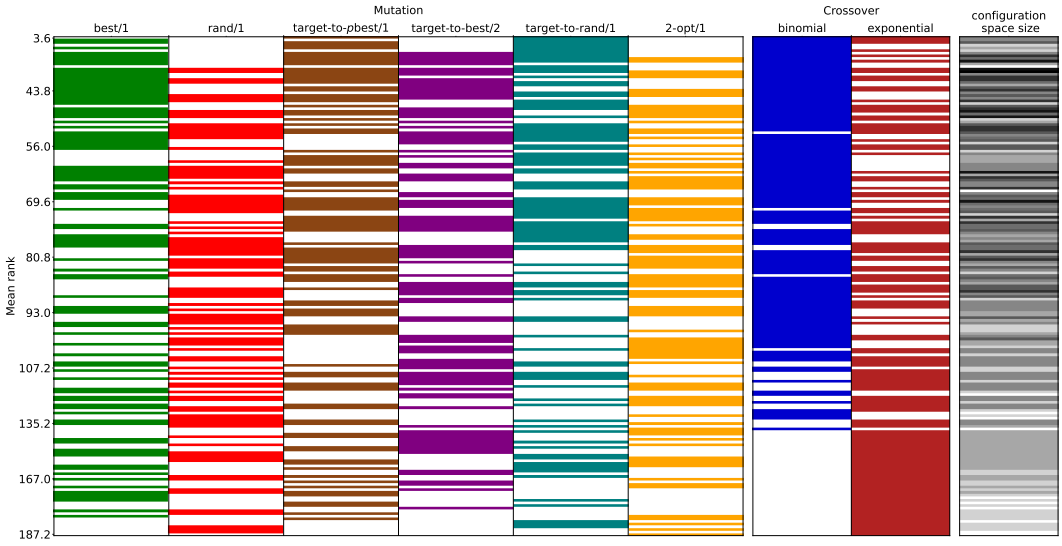


Figure 15: Ranking of DE configuration spaces in function group 3.

Function group 3 The ranking of function group 3, which is shown in Figure 15, is quite similar to that of function group 2. The largest difference is that target-to-best/2 is now more successful, but it is still not included in the very best configuration spaces. Binomial crossover seems even more crucial in this function group, and there also seems to be a stronger advantage for larger configuration space sizes.

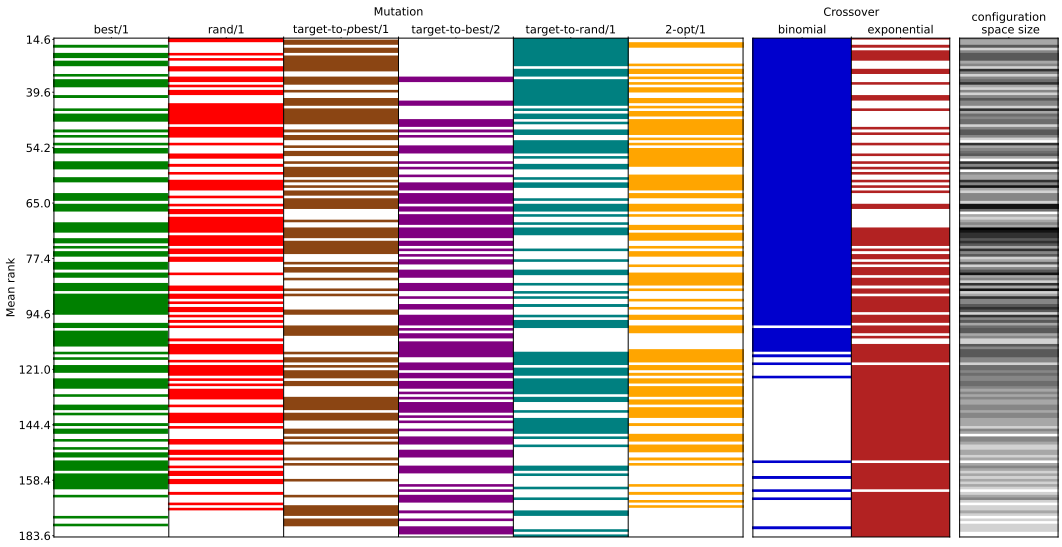


Figure 16: Ranking of DE configuration spaces in function group 4.

Function group 4 As shown in Figure 16, the advantage for binomial crossover is very apparent in function group 4. The top 57% of configuration spaces *all* include binomial crossover, some in combination with the exponential variant, and some without. The best/1 mutation scheme is much less prominent here, and most of the best-performing configuration spaces have 3 to 6 operator configurations; neither very large nor very small.

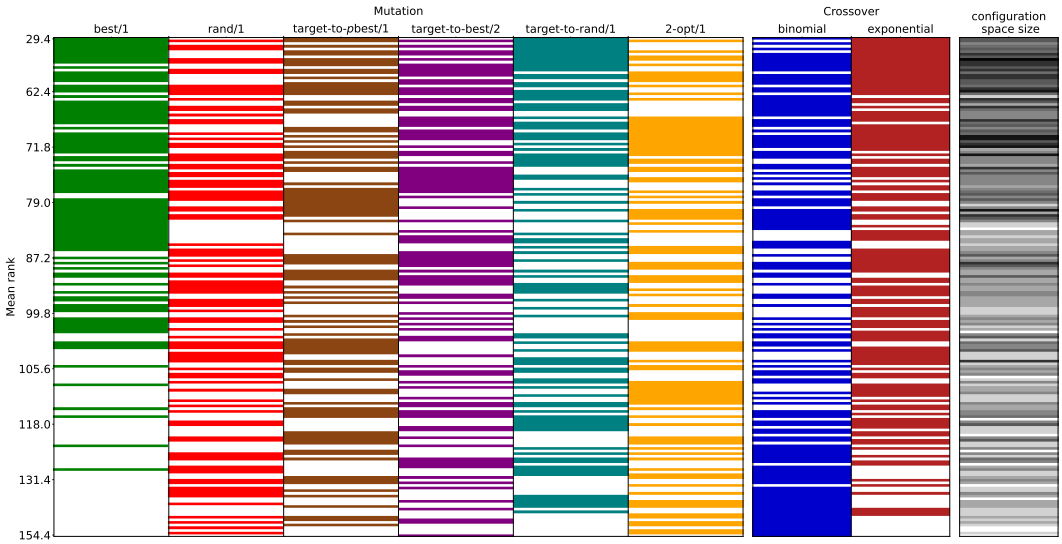


Figure 17: Ranking of DE configuration spaces in function group 5.

Function group 5 Figure 17 shows that in function group 5, there is a very strong preference towards large configuration spaces. Many of the highest ranked configuration spaces use both exponential and binomial crossover, although exponential crossover seems more crucial. Further, the best/1 mutation scheme is most often associated with a high rank, and many of the highest-ranked configuration spaces also use target-to-rand/1.

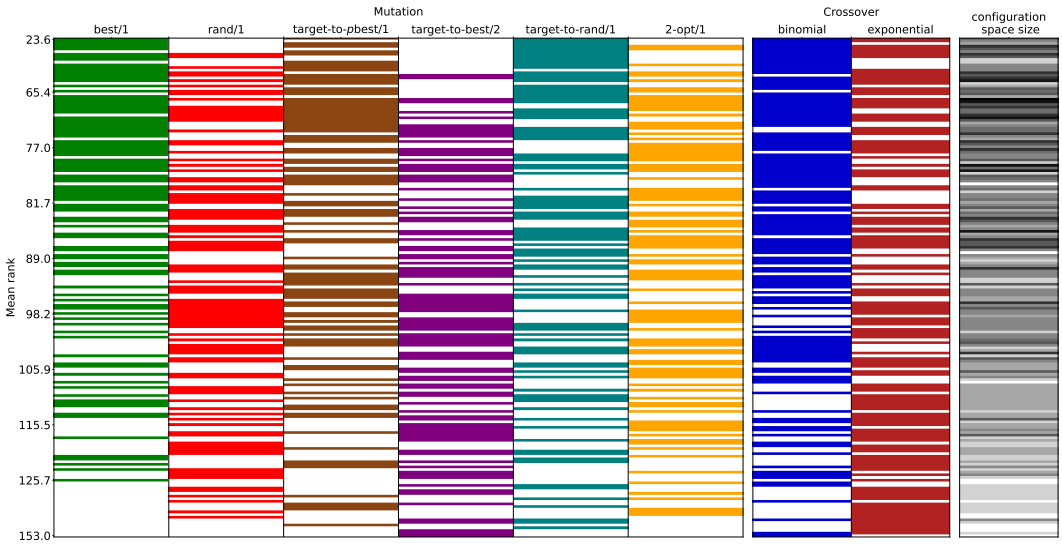


Figure 18: Ranking of DE configuration spaces over all function groups.

All functions Finally, we consider the ranks across all 24 functions, as shown in Figure 18. It seems that, in general, it is wise to include a larger number of operator configurations in the configuration space for the most versatile performance. However, the very best configuration spaces use about 6 operator configurations. The highest ranked configuration space overall is the one using: best/1, target-to-*p*best/1 and target-to-rand/1 mutations, and both crossover types. This configuration space also ranked highest in function groups 2 and 5 individually. All of its included operators individually are also frequently included in other highly-ranked configuration spaces. Clearly, the fact that exponential crossover was activated so infrequently in our previous experiment does not mean it should be excluded from the configuration space. It seems especially crucial in function groups 1 and 5, which corresponds to our findings in [9]. In fact, the highest ranked configuration space in each function group, and across all functions, all include the exponential crossover operator.

Again, it is very apparent that target-to-best/2 is not included in the highest ranked configuration spaces, considering it was activated so frequently in our previous experiments. We expect that target-to-rand/1 and target-to-*p*best/1 are simply better options, rendering target-to-best/2 obsolete. It is harder to determine what impact the rand/1 and 2-opt/1 mutation schemes have on the performance, as they are not necessarily mostly associated with either high or low ranks.

Although we found that including *all* operators in the configuration space will not result in the optimal performance, the large configuration spaces are generally still associated with high performance ranks when considering all benchmark functions. Ideally, the user of an AOS method would not have to worry about compiling a configuration space, and should be able to simply include any operator that they think *might* contribute to the performance of DE. The role of the AOS is then to determine which operators are most valuable.

To determine if it is worth fine-tuning the configuration space, or if we can simply include all operators and let the AOS figure out the best strategies during runtime, we plot ECDF graphs aggregated for all runs on each function group and one across all functions (see Figure 19). The algorithms are labeled by the function group(s) they had the highest rank on. For example, `G1` indicates the algorithm with the configuration space that performed best in function group 1. Because the configuration space that ranked highest overall also ranked highest in function groups 2 and 5 individually, it is termed `G2,G5,All`. Further, we include the configuration space with all operators activated (`Complete`), and the configuration space we used in Section 5 which was simply compiled by intuition, denoted as `Default`.

Table 7: The five highest-ranked configuration spaces according to the ranking procedure described in Section 6, for each function group and overall. When an operator is included in the configuration space, the corresponding cell is colored black.

Function group 1

rank	best/1	rand/1	tt*p*b/1	ttb/2	ttr/1	2-opt/1	bin	exp
14.0	■							■
16.2	■					■		■
16.6	■		■					■
17.8	■		■			■		■
19.6	■	■						■

Function group 2

rank	best/1	rand/1	tt*p*b/1	ttb/2	ttr/1	2-opt/1	bin	exp
6.5	■		■		■		■	■
7.2			■		■		■	
13.2	■		■		■	■	■	
13.5	■		■		■	■	■	■
15.0	■				■	■	■	■

Function group 3

rank	best/1	rand/1	tt*p*b/1	ttb/2	ttr/1	2-opt/1	bin	exp
3.6			■		■		■	■
4.6	■							
6.0	■		■		■		■	■
6.6								
8.0	■		■		■		■	

Function group 4

rank	best/1	rand/1	tt*p*b/1	ttb/2	ttr/1	2-opt/1	bin	exp
14.6		■			■		■	■
17.4		■	■		■		■	
24.4			■		■	■	■	
26.0	■				■		■	■
27.4			■		■		■	

Function group 5

rank	best/1	rand/1	tt*p*b/1	ttb/2	ttr/1	2-opt/1	bin	exp
29.4	■		■		■		■	■
34.0	■	■	■	■	■	■		■
38.2	■				■		■	■
46.4	■	■	■	■	■			■
47.6	■	■			■		■	■

All functions

rank	best/1	rand/1	tt*p*b/1	ttb/2	ttr/1	2-opt/1	bin	exp
23.6	■		■		■		■	■
31.8	■				■		■	
38.5	■		■		■		■	■
42.9	■		■		■	■	■	■
43.2	■				■	■	■	■

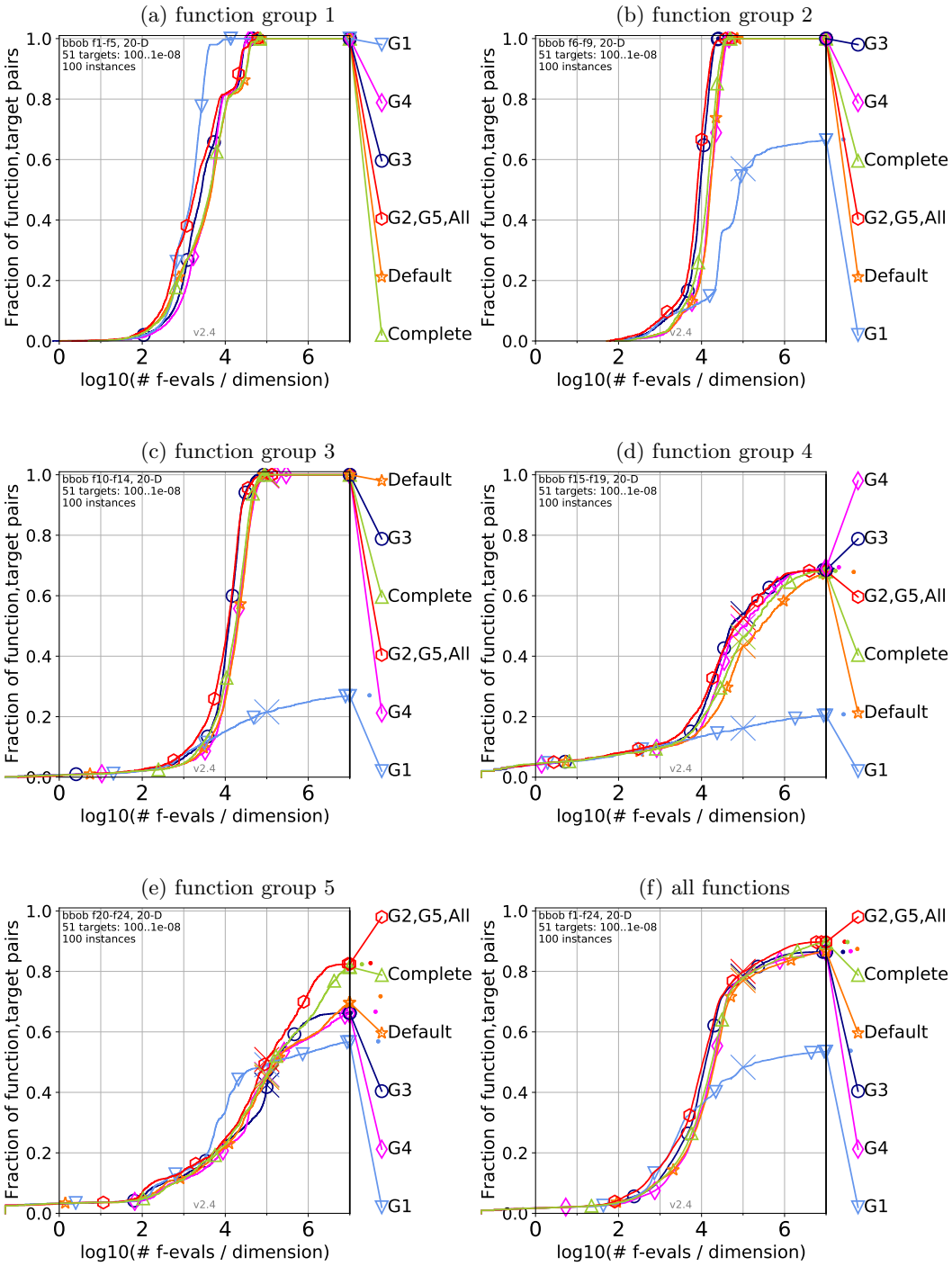


Figure 19: ECDFs of the best configuration spaces for each function group and overall, compared to the configuration space chosen by our intuition (`Default`), and the one containing all operators (`Complete`).

In Figure 19, we see that `G1` converges on the optimum faster than the other algorithms on the functions of function group 1, but that it has a major disadvantage in the other function groups. Further, it is evident that in function groups 1, 2, and 3, the performances of the other configuration spaces are very similar, with only slight differences in terms of convergence speed. In function group 4, `G4` has a small advantage over Complete and Default. The largest difference manifests itself in function group 5, where `G2,G5,All` and `Complete` are able to solve over 10% more problems than the competition. Overall, `Complete` has a slightly worse performance than the highest-ranked configuration space of each function group, but it is very consistent across different function groups. Considering that a main goal of AOS research is minimizing the need to tune an algorithm before applying it to a new problem, and having to tune the configuration space beforehand opposes this goal, we can conclude that it might not be worth fine-tuning the configuration space for a specific problem, as the AOS with Compass credit seems very capable of selecting an appropriate strategy or combinations thereof out of a large configuration space, at the cost of a slightly deteriorated convergence speed.

7 Conclusion

This thesis presents a novel algorithmic framework for Differential Evolution, with a mechanism for adapting operators and parameters that balances exploration and exploitation, by rewarding individuals that successfully explore the search space. The framework elegantly intertwines Adaptive Operator Selection (AOS) and parameter adaptation, and lets both components work towards a common goal. The AOS component selects the operator configurations, which are combinations of DE's mutation and crossover operators, to apply in each iteration, while the parameter adaptation attempts to find the optimal values of the mutation and crossover rate for each operator configuration individually.

A metric that captures the exploratory/exploitative tendency of an individual is proposed, and it is steers the direction of both the adaptation of parameters and that of operators. This diversity-based metric is combined with a fitness-based metric in different ways to give rise to seven different so-called credit assignment schemes, which measure the improvement of an offspring individual w.r.t. its parent in terms of fitness and/or diversity.

The framework also implements several options for the other components of the AOS procedure: generating reward for each operator configuration, quality assignment, and updating of the application probabilities. For each credit assignment scheme, we perform an extensive tuning experiment to find the optimal variant of each component and set their respective hyperparameters.

A benchmarking experiment on the BBOB/COCO [35] benchmark suite revealed that one of the resulting algorithms, which uses a credit assignment scheme modified from the Compass [47] AOS method, significantly outperforms all 'static' DEs on three difficult benchmark functions, and has comparable performance to the best static DE on most other functions. This result demonstrates that the AOS is consistently capable of selecting an appropriate strategy for a particular problem, and in some cases, it can combine the strengths of several operators to surpass the performance of any of the static DEs. The DE with Compass-inspired credit still lagged behind the BIPOP-CMA-ES [32] in general, but on most functions outperformed the current state-of-the-art AOS method for DE: U-AOS-FW [60]. Statistical testing results revealed that adaptation based on Compass or Div credit increases DE's versatility across different problems the most.

Several behavioral aspects of the seven AOS methods were analyzed, including the activation rates of each operator configuration, the number of transitions between configurations, and the generated parameter values. It is clear that the selected credit assignment scheme has a huge impact on the selection of operators. The Compass credit shows a quite one-sided selection of operators across different problems, while other credit assignment schemes use different (combinations of) configurations depending on the problem. Interestingly, the exponential crossover is in general utilized much

less frequently than the binomial variant. The analysis of transition frequencies showed that usage of the Compass credit results in significantly fewer transitions, while FitSq-Div is associated with the most transitions. Analyzing the generated F and Cr values revealed that the credit assignment scheme has a huge impact on the parameter adaptation. Especially Compass and Pareto showed very different behavior compared to the others.

In-depth behavioral analysis of individual runs on functions where only AOS methods were successful showed that certain combinations of operator configurations were repeatedly used in the successful trials. However, there were also trials where only one operator configuration was mainly used, hinting that the diversity-driven parameter adaptation could also be (partially) responsible for the improved performance compared to the static DEs. We also observed recurring patterns w.r.t. the adaptation of parameters in the successful runs on certain problems.

An AOS method can only perform as well as the strategies in its configuration space allow it to. Therefore, we benchmarked 189 configuration spaces of different sizes in combination with Compass, and ranked them based on their performance on each problem. We analyzed which configuration spaces performed best in each function group and overall, and found that different (combinations of) operators were successful in different function groups. However, a near-optimal performance in each function group could be obtained by simply including all considered operators in the configuration space. Considering that an AOS method is meant to reduce the manual tuning effort, we recommend using a large configuration space containing operators with varying characteristics, rather than trying to fine-tune the configuration space with only the 'essential' operators, especially if a consistent performance across different problem types is desired. The largest configuration space in our experiments contained 12 configurations. The Compass-based AOS seems to be very capable of selecting the appropriate strategies from a set of this size. The ranking of configuration spaces also revealed that the activation rates of a certain operator configuration do not necessarily give an indication of its importance to be included in the configuration space.

8 Future work

Due to the promising results of the proposed AOS framework, especially when using the Compass credit, future efforts should go towards transferring the methodology to other metaheuristics, such as Genetic Algorithms (GAs) or Particle Swarm Optimization (PSO). The proposed individual-wise diversity metrics (diversity ratio and diversity difference) should be easily transferable to other population-based metaheuristics. Since GAs typically operate on discrete search spaces, a different distance metric should be employed, such as the Hamming distance [30], and the 'mean position' needs to be computed differently (e.g., by taking the most common value for each component). For PSO, on the other hand, the adaptation manager is compatible without any modifications.

Further, it would be interesting to see if the performance of AOS paradigms based on the multi-armed bandit framework [14, 18, 45] can be improved by leveraging the diversity-based metrics proposed in this thesis. Integration of these methods into the existing framework might prove to be difficult, as the structure of these bandit-based AOS methods is fundamentally different.

The diversity-based adaptation in this thesis makes use of the Euclidean distance of coordinates in the search space. Due to the curse of dimensionality, this distance metric will not give desirable results in higher dimensions, where the Euclidean distances between any two pairs of points become more and more similar. It is plausible that even in 20 dimensions, the dimensionality of the benchmark problems used in this thesis, the Euclidean distance is not an optimal choice, making the diversity-based metrics less 'sensitive'. A distance metric that is less susceptible to this problem is the Manhattan distance [8]. Future efforts should repeat the presented experiments with this alternative distance metric, and analyze any differences in performance and algorithmic behavior. It is also worthwhile to repeat the experiments with problems in different dimensions, as results could vary.

The parameter adaptation scheme employed in our framework, which is extended from the SHADE algorithm [66], only adapts the mutation and crossover rates, while the population size is kept at a constant value. Future work should consider incorporating the population size in the parameter adaptation. Another, perhaps simpler, possibility is to follow the approach of L-SHADE [67], where the population size is linearly reduced as the run progresses.

As the imperative role of the boundary constraint handling method (BCHM) was established in [38, 9], an important avenue for future research is to include the BCHM in the process of AOS. For the purposes of this thesis, we chose the BCHM which performed the best overall in [9], but the performance of the AOS methods might be improved by including several BCHMs in the configuration space, as it was showed in [9] that the optimal choice of the BCHM depends on the problem to solve *and* the operators used by DE.

Lastly, a much more in-depth study needs to be performed to identify patterns in the adaptation behavior that lead to either desirable or undesirable performance results. A better understanding of what desired characteristics the course of adaptation should possess on certain problems can aid further development of adaptive algorithms. A next step towards this goal would be to extract features characterizing the course of adaptation in individual runs, and correlating those features with the observed performance of the adaptive algorithm.

References

[1] J. Arabas, A. Szczepankiewicz, and T. Wroniak. Experimental comparison of methods to handle boundary constraints in differential evolution. In *Parallel Problem Solving from Nature, PPSN XI*, pages 411–420, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, May 2002.

[3] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1769–1776 Vol. 2, 2005.

[4] B. Babu, P. G. Chakole, and J. Syed Mubeen. Multiobjective differential evolution (MODE) for optimization of adiabatic styrene reactor. *Chemical Engineering Science*, 60(17):4822–4837, 2005.

[5] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Inc., USA, 1996.

[6] P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In T. Bartz-Beielstein, M. J. Blesa Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, editors, *Hybrid Metaheuristics*, pages 108–122, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[7] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. *F-Race and Iterated F-Race: An Overview*, pages 311–336. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[8] P. E. Black. Manhattan distance. *Dictionary of Algorithms and Data Structures*, 2019. Accessed June 4, 2021.

[9] R. Boks, A. V. Kononova, and H. Wang. Quantifying the impact of boundary constraint handling methods on differential evolution. In *Proceedings of the 2021 Genetic and Evolutionary Computation Conference Companion*, GECCO '21, New York, NY, USA, 2021. Association for Computing Machinery.

[10] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Evolutionary Computation, IEEE Transactions on*, 10:646 – 657, 01 2007.

[11] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, and V. Zumer. Dynamic optimization using self-adaptive differential evolution. In *2009 IEEE Congress on Evolutionary Computation*, pages 415–422, 2009.

[12] E. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64:1695–1724, 07 2013.

[13] C.-W. Chiang, W.-P. Lee, and J.-S. Heh. A 2-opt based differential evolution for global optimization. *Applied Soft Computing*, 10(4):1200 – 1207, 2010. Optimisation Methods & Applications in Decision-Making Processes.

[14] L. Da Costa, A. Fialho, M. Schoenauer, and M. Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, page 913–920, New York, NY, USA, 2008. Association for Computing Machinery.

[15] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, 2011.

[16] A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[17] A. Fialho. *Adaptive Operator Selection for Optimization*. PhD thesis, Université Paris-Sud, 2011.

[18] A. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60(1–2):25–64, Oct. 2010.

[19] A. Fialho, R. Ros, M. Schoenauer, and M. Sebag. Comparison-based adaptive strategy selection with bandits in differential evolution. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, pages 194–203, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[20] A. Fialho, M. Schoenauer, and M. Sebag. Analysis of adaptive operator selection techniques on the royal road and long k-path problems. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, page 779–786, New York, NY, USA, 2009. Association for Computing Machinery.

[21] A. Fialho, M. Schoenauer, and M. Sebag. Fitness-auc bandit adaptive strategy selection vs. the probability matching one within differential evolution: An empirical comparison on the bbob-2010 noiseless testbed. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '10, page 1535–1542, New York, NY, USA, 2010. Association for Computing Machinery.

[22] S. Finck, N. Hansen, R. Ros, and A. Auger. Black-box optimization benchmarking procedure. `https://coco.gforge.inria.fr/COCOdoc/bbo_experiment.html`. Accessed May 26, 2021.

[23] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2010: Presentation of the noiseless functions. `http://coco.gforge.inria.fr/downloads/download16.00/bbobdocfunctions.pdf`, 2010. Accessed May 24, 2021.

[24] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[25] D. Goldberg. Probability matching, the magnitude of reinforcement, and classifier system bidding. *Machine Learning*, 5:407–425, 2005.

[26] W. Gong and Z. Cai. Differential evolution with ranking-based mutation operators. *IEEE Transactions on Cybernetics*, 43(6):2066–2081, 2013.

[27] W. Gong, A. Fialho, and Z. Cai. Adaptive strategy selection in differential evolution. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, page 409–416, New York, NY, USA, 2010. Association for Computing Machinery.

[28] W. Gong, A. Fialho, Z. Cai, and H. Li. Adaptive strategy selection in differential evolution for numerical optimization: An empirical study. *Inf. Sci.*, 181(24):5364–5386, Dec. 2011.

[29] R. Gämperle, S. D. Müller, and P. Koumoutsakos. A parameter study for differential evolution. In *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pages 293–298, 2002.

[30] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[31] N. Hansen. *The CMA Evolution Strategy: A Comparing Review*, pages 75–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[32] N. Hansen. Benchmarking a bi-population cma-es on the bbob-2009 function testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, page 2389–2396, New York, NY, USA, 2009. Association for Computing Machinery.

[33] N. Hansen, Y. Akimoto, and P. Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, Feb. 2019.

[34] N. Hansen, A. Auger, D. Brockhoff, D. Tušar, and T. Tušar. Coco: Performance assessment, 2016.

[35] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *ArXiv e-prints*, arXiv:1603.08785, 2016.

[36] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '10, page 1689–1696, New York, NY, USA, 2010. Association for Computing Machinery.

[37] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, page 1152–1157. AAAI Press, 2007.

[38] A. V. Kononova, F. Caraffini, and T. Bäck. Differential evolution outside the box. *ArXiv e-prints*, arXiv:2004.10489, 2020.

[39] J. Lampinen and I. Zelinka. Mixed integer-discrete-continuous optimization by differential evolution - part 1: the optimization method. In *Czech Republic. Brno University of Technology*, pages 77–81, 1999.

[40] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[41] F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[42] M. López-Ibáñez. Documentation of the irace package for R. `https://cran.r-project.org/web/packages/irace/irace.pdf`, 2020.

[43] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[44] R. Mallipeddi and P. N. Suganthan. Differential evolution algorithm with ensemble of parameters and mutation and crossover strategies. In B. K. Panigrahi, S. Das, P. N. Suganthan, and S. S. Dash, editors, *Swarm, Evolutionary, and Memetic Computing*, pages 71–78, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[45] J. Maturana, A. Fialho, F. Saubion, M. Schoenauer, and M. Sebag. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *2009 IEEE Congress on Evolutionary Computation*, pages 365–372, 2009.

[46] J. Maturana, F. Lardeux, and F. Saubion. Controlling behavioral and structural parameters in evolutionary algorithms. *Evolutionary Algorithms*, 5926, 01 2009.

[47] J. Maturana and F. Saubion. A compass to guide genetic algorithms. In G. Rudolph, T. Jansen, N. Beume, S. Lucas, and C. Poloni, editors, *Parallel Problem Solving from Nature – PPSN X*, pages 256–265, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[48] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello. A comparative study of differential evolution variants for global optimization. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, page 485–492, New York, NY, USA, 2006. Association for Computing Machinery.

[49] J. Mwaura, A. P. Engelbrecht, and F. V. Nepomuceno. Diversity measures for niching algorithms. *Algorithms*, 14(2), 2021.

[50] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[51] Q.-K. Pan, P. Suganthan, L. Wang, L. Gao, and R. Mallipeddi. A differential evolution algorithm with self-adapting strategy and control parameters. *Computers & Operations Research*, 38(1):394–408, 2011. Project Management and Scheduling.

[52] A. P. Piotrowski. Review of differential evolution population size. *Swarm and Evolutionary Computation*, 32:1 – 24, 2017.

[53] K. Price. Differential evolution vs. the functions of the 2/sup nd/ iceo. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pages 153–157, 1997.

[54] K. Price, R. Storn, and J. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. Springer, 2005.

[55] J. Qijang. A unified differential evolution algorithm for global optimization. *IEEE Transactions on Evolutionary Computation*, 2014.

[56] A. K. Qin, V. L. Huang, and P. N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):398–417, 2009.

[57] A. K. Qin and P. N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1785–1791 volume 2, 2005.

[58] J. Ronkkonen, S. Kukkonen, and K. V. Price. Real-parameter optimization with differential evolution. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 506–513 Vol.1, 2005.

[59] M. Sharma, M. López-Ibáñez, and D. Kazakov. Performance assessment of recursive probability matching for adaptive operator selection in differential evolution. In A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, editors, *Parallel Problem Solving from Nature – PPSN XV*, pages 321–333, Cham, 2018. Springer International Publishing.

[60] M. Sharma, M. Lopez-Ibanez, and D. Kazakov. Unified framework for the adaptive operator selection of discrete parameters, 2020.

[61] A. Soler-Dominguez, A. A. Juan, and R. Kizys. A survey on financial applications of metaheuristics. *ACM Comput. Surv.*, 50(1), Apr. 2017.

[62] R. Storn. On the usage of differential evolution for function optimization. In *Proceedings of North American Fuzzy Information Processing*, pages 519–523, 1996.

[63] R. Storn. *Differential Evolution Research – Trends and Open Questions*, pages 1–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[64] R. Storn and K. Price. Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, 23, 01 1995.

[65] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[66] R. Tanabe and A. Fukunaga. Success-history based parameter adaptation for differential evolution. In *2013 IEEE Congress on Evolutionary Computation*, pages 71–78, 2013.

[67] R. Tanabe and A. S. Fukunaga. Improving the search performance of SHADE using linear population size reduction. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1658–1665, 2014.

[68] O. Tange. GNU Parallel 20201122 ('Biden'), Nov. 2020. GNU Parallel is a general parallelizer to run multiple serial command line programs in parallel without changing them.

[69] D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, page 1539–1546, New York, NY, USA, 2005. Association for Computing Machinery.

[70] J. Valadi and P. Siarry. *Applications of Metaheuristics in Process Engineering*. Springer Publishing Company, Incorporated, 2014.

[71] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[72] Y. Wang, Z. Cai, and Q. Zhang. Differential evolution with composite trial vector generation strategies and control parameters. *IEEE Transactions on Evolutionary Computation*, 15(1):55–66, 2011.

[73] F. Wilcoxon. *Individual Comparisons by Ranking Methods*, pages 196–202. Springer New York, New York, NY, 1992.

[74] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[75] D. Zaharie. Critical values for the control parameters of differential evolution algorithms. *Critical Values for the Control Parameters of Differential Evolution Algorithms*, 2:62–67, 01 2002.

[76] J. Zhang and A. C. Sanderson. JADE: Self-adaptive differential evolution with fast and reliable convergence performance. In *2007 IEEE Congress on Evolutionary Computation*, pages 2251–2258, 2007.

[77] K. Zielinski and R. Laur. *Stopping Criteria for Differential Evolution in Constrained Single-Objective Optimization*, pages 111–138. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.