Universiteit Leiden

Master Computer Science

Portfolio Selection in Adaptive Operator Selection for Differential Evolution using Multi-Armed Bandits

Name: Per Hermanus
Student ID: s1856863

Date: 29/11/2024

Specialisation: Bio-Informatics

1st supervisor: Anna V. Kononova
2nd supervisor: Niki van Stein
Daily supervisor: Diederick Vermetten
Daily supervisor: Jacob de Nobel

Master's Thesis in Computer Science

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

Abstract

Differential Evolution (DE) algorithms are flexible algorithms, able to provide good results on a wide variety of optimization problems. One of the challenges with DE is using that flexibility to choose the right configuration for a problem, especially when there is no prior knowledge of the underlying function. Adaptive Operator Selection (AOS) mechanisms allow an algorithm to automatically adapt its own parameters on the fly, vital for good performance on these black-box problems. One such AOS mechanism is the Multi-Armed Bandit (MAB) which uses a portfolio, a pre-defined selection of configurations which it can switch between.

This thesis investigates how the size of a portfolio impacts the performance of MAB-based AOS, using a suite of 24 different problems and portfolios consisting of up to 16 different mutation operators. After determining the best performing (hyper)parameters of all mutation operators and the MAB we found that using different portfolio sizes has limited effect on DE performance on 5-dimensional problems, but has a noticeable impact on 20-dimensional problems. We show that in the latter case, performance peaks at a portfolio consisting of 4 different configurations and continuously degrades as larger and larger portfolios are considered. This suggests little need to consider large portfolios for high-dimensional problems.

Contents

1 Introduction **3**

2 Black-box optimisation **3**
2.1 IOHprofiler 4

3 Differential Evolution **4**
3.1 Initialisation 7
3.2 Mutation operators 7
3.3 Boundary constraints 9
3.4 Crossover operators 10
3.5 Selection operators 11

4 Adaptive Operator Selection **11**
4.1 Deterministic 11
4.2 Adaptive 11
4.3 Self-adaptive 12
4.4 Credit Assignment 13

5 Experiments **13**
5.1 Mutation Operator Baselines 14
5.2 AOS parameters 16
5.3 Portfolio Generation 19

6 Conclusion **24**

7 Code **25**

1 Introduction

Optimisation is a field of study that aims to find and improve methods that efficiently solve problems. The nature of these problems can range from engineering [21] to biological [46] to financial [23], as long as the problem can be expressed as a function. In practice, optimisation often means finding a set of inputs that generates an output that is either as high or as low as possible.
A difficult type of problem encountered in optimisation is the *black-box problem* [18]. These are problems described by a function whose analytical formula is unknown, but the problem can be sampled to obtain the evaluation of a given valid input. The number of times this function can be sampled is limited either by arbitrary constraints, such as the introduction of a maximum budget, or real-world constraints like physical resources or time [36].
Differential Evolution (DE) is a powerful and easily configurable stochastic optimisation algorithm in the field of evolutionary algorithms. It is able to optimise real-valued problems by iteratively generating candidate solutions. The mutation operator is the driving force behind DE. It creates new values for candidate solutions, enabling movement throughout the search space. Changing this operator changes DE's optimisation characteristics. This was already noted by Storn and Price, who suggested multiple different mutation operators both when they introduced DE and soon after [44] [42]. Choosing the right configuration of operator and parameters for a specific problem can dramatically impact the convergence speed of the DE algorithm and/or the final obtained solution [30].
Adaptive Operator Selection (AOS) methods provide the ability to automate the selection of optimal configurations during runtime. This allows an optimisation algorithm such as DE to not only select optimal configurations for different problems but also change the configuration during a single problem. For example, changing from an exploration phase to an exploitation phase. These capabilities are very useful for problems that would otherwise benefit from a lot of human oversight during runtime for optimal results, such as the aforementioned black-box problems. For AOS methods that consider a finite number of configurations, the problem of which configurations and how many configurations to include can be seen as a portfolio optimisation problem [5], where the goal is to build a portfolio of complementary configurations.
This thesis explores how the number of configurations in a portfolio, the portfolio size, affects the performance of DE on 5- and 20-dimensional problems by implementing an AOS strategy based on the Multi-Armed Bandit [28] [38]. With a series of experiments and IOHanalyzer's [47] BBOB test suite [19] we first find optimal parameters for mutation operators when used on their own. Next, we find optimal hyperparameters for the MAB adaptive operator selector. Finally, we use the results of the previous experiments to test portfolio's ranging from 1 up to 16 different mutation operators.

2 Black-box optimisation

In optimisation the goal is to find an optimal solution to a problem. We define this problem as a N-dimensional continuous objective function $f : X_N \to \mathbb{R}$ which outputs a real number. Input X_N is a solution vector of length N with its values constrained by the upper-bounds u_n and lower-bounds l_n defined by box-constraints $\{\forall x_n \in X_N \mid l_n \leq x_n \leq u_n,\ l_n < u_n,\ 0 \leq n < N\}$. Without loss of generality, this solution vector should minimise

the output of f and needs to be found while being limited by a maximum number of function evaluations or a maximum allowed runtime. The objective function can take many shapes. Knowing what kind of characteristics a problem has allows one to tailor an optimisation algorithm to it, as [37] shows for different cases in machine learning.
The challenge of black-box optimisation is to solve a given objective function as efficiently as possible without using any prior information. This includes the exact definition of f or its derivative. The optimisation algorithm can attempt to make its own inferences through information obtained by evaluating an input (querying) at the cost of budget, as long as it reliably converges to the global minimum.

2.1 IOHprofiler

We use IOHprofiler to access and analyse a suite of black-box problems. IOHprofiler [14] is a multipart tool that assists with running benchmarks for optimisation algorithms through IOHexperimenter [13] and analysing the generated data through IOHanalyzer [47]. Its components can be seen in figure 1. IOHexperimenter is able to connect to and log the performance of optimisation algorithms on both continuous real-valued single-objective problems from COCO's [18] suite of Black-Box Optimization Benchmark (BBOB) [19] problems and discrete problem suites. IOHanalyzer allows for an interactive analysis and visualisation of logs obtained with IOHexperimenter through a web-based interface.

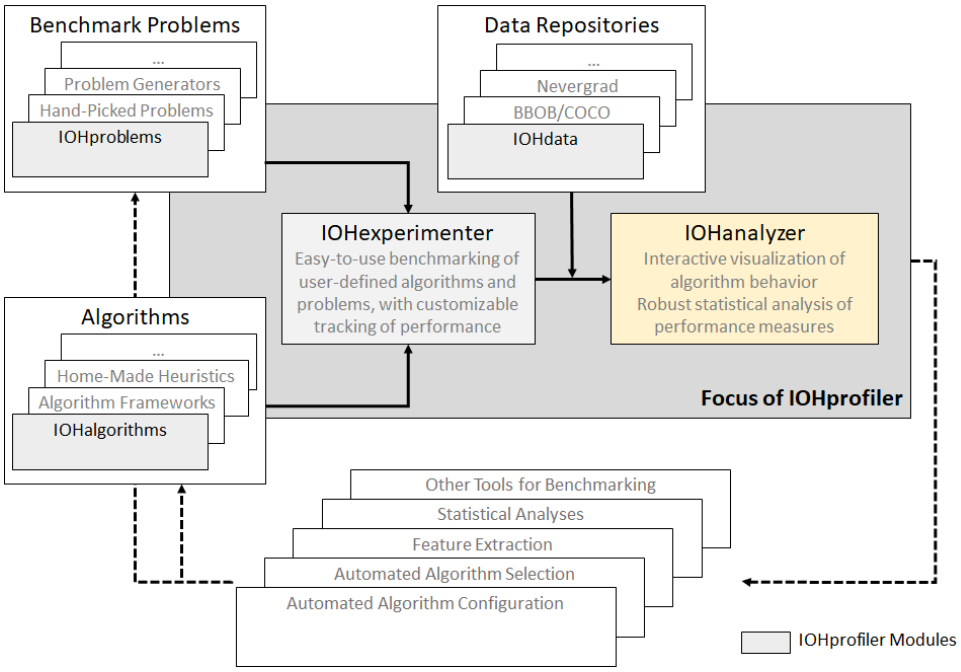

Figure 1: An overview of the steps in Algorithm benchmarking that are targeted by IOHprofiler [47].

3 Differential Evolution

Differential Evolution (DE) is a type of Evolutionary Algorithm (EA). Through repeated mutation, crossover, and selection of a population of candidate solutions, it attempts to

find an optimal solution to a problem defined by an objective function f. Different DE strategies are classified using the notation $DE/a/b/c$ [44]. In this notation a denotes the mutation strategy used, b denotes the number of difference vectors included in the mutation strategy, and c denotes the crossover scheme. The strategy that uses a random vector as base vector during mutation, adds one difference vector to it, and then applies binomial crossover would be described as $DE/rand/1/bin$, of which Pseudocode 1 shows an example.

Algorithm 1: Pseudocode for DE/rand/1/bin

NP $\leftarrow$ population size, N $\leftarrow$ vector size;
P $\leftarrow$ $\{ x_0, \ldots, x_{NP} \}$ with $x_{i,0\ldots N} \xleftarrow{u.a.r} range[x^{min}_{\_,0\ldots N}, \ldots, x^{max}_{\_,0\ldots N}]$; `// Create population`
while *Stop condition not reached* **do**
  foreach $x_i \in P$ **do**
    $x_{r1}, x_{r2}, x_{r3} \xleftarrow{u.a.r} P \backslash \{x_i\}$ without repetition ; `// Mutation`
    $v_i \leftarrow x_{r1} + F \cdot (x_{r2} - x_{r3})$;
    $k \leftarrow \mathcal{U}\lfloor(1, N+1]\rfloor$; `// Crossover`
    $P' \leftarrow \{\emptyset\}$
    foreach $j \in [1, 2, \ldots, N]$ **do**
      if $\mathcal{U}[0,1] \leq Cr$ ***or*** $j == k$ **then**
        $x'_{i,j} \leftarrow v_{i,j}$
      else
        $x'_{i,j} \leftarrow x_{i,j}$
    $P' \leftarrow P' \cup x'_{i,j}$
  foreach $x_i \in P$ **do**
    if $f(x_i) < f(x'_i)$ **then**
      $x_i \leftarrow x'_i$

DE was first introduced by Storn and Price as a more performant alternative to Adaptive Simulated Annealing [20] and Annealed Nelder & Mead [32] [44]. It attempts to adhere to three main requirements. The algorithm should be able to find the global minimum regardless of how the population is initialised, the algorithm should be able to converge with reasonable speed and the algorithm should contain a minimal number of easy-to-use control parameters. So far, numerous applications of DE algorithms have been tested and used successfully in different fields of study. Including but not limited to chemometric experiment optimisation [41], wind speed distribution parameter estimation in wind engineering [21], Multi-Threshold image thresholding [12] and neural network-based Multi-Agent Reinforcement Learning [8].

As can be seen in Figure 2 and Pseudocode 1. The DE algorithm starts with the initialisation of a fixed-size population of candidate solutions throughout the viable solution-space. Each candidate solution is represented by a vector of size N.

The first step of the main loop is mutation. Here, a donor vector v_i (mutant) is generated

for each individual x_i in the population. In DE these donor vectors are created by modifying an existing vector from the current population, this vector is called the base vector. Depending on the chosen mutation strategy, one or more difference vectors are created by subtracting two vectors from each other, also pulled from the current population.
These difference vectors are scaled by mutation rate F and added to the base vector. Often, F is a value between 0.0 and 1.0. Bigger values of F allow for faster movement through the search-space due to the larger difference vectors added to the base vector. This comes at the cost of difficult convergence to a single point close to the base vector, which is easier with small difference vectors.
Since the generated donor vector might contain values that fall outside the defined search space it should be checked and, if needed, repaired according to the specified boundary constraint handling method before continuing to the next step.
The second step is crossover. Trial vector x'_i is created by copying the existing candidate solution x_i and substituting elements in the vector with elements from donor vector v_i. The amount of elements substituted is dependent on Cr. A Cr of 1.0 means all elements are copied from v_i. As Cr gets closer to 0.0, less and less elements get copied, with a minimum of one copied element.
The third and last step is selection. The DE algorithm evaluates the existing candidate solutions and the generated trial vectors and uses either the obtained fitness values or some other scoring metric to create the next generation. These steps are repeated until a stop condition is reached.

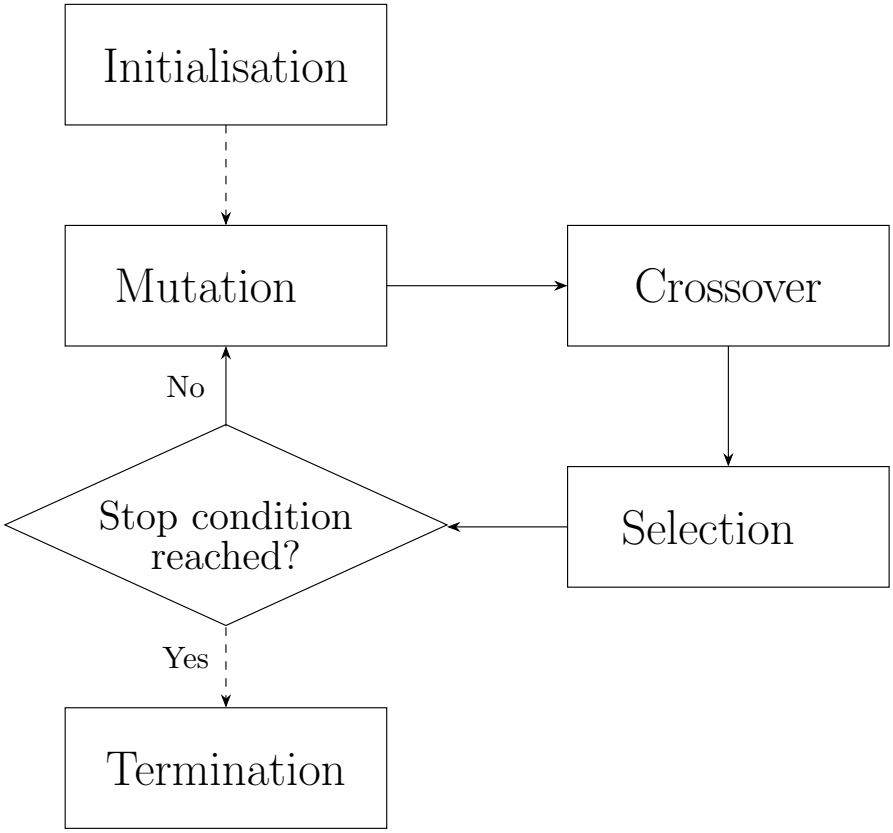

Figure 2: Flowchart of a typical DE algorithm.

Many different initialisation, mutation, boundary constraint, crossover, and selection op-

erators [6]. Sections 3.1, 3.2, 3.3, 3.4 and 3.5 respectively provide information on the operators implemented in this project.

3.1 Initialisation

Since we are dealing with a black-box problem, we are unable to use characteristics of the objective function for value initialisation. What is known are the limits of the search space of f. Each objective function in the BBOB suite has a box-constraint of $[-5, 5]^N$ which can shifted and rotated in space through IOHprofiler *instances*. Initialisation of the population is done using values complying to the box-constrains.
There are multiple suggested methods of initialising the population such as random, Latin Hypercube [26], center-based [35], opposition based [34], and grid initialisation.
Important fixed control parameters that should be decided upon during the initialisation phase are the population size M and the number of allowed function queries, also known as budget, B. Both M and B are generally scaled with dimensionality [31] [22].

3.2 Mutation operators

Mutation operators dictate how the donor vector is generated. This is considered to be the most important step due to it being the driving force behind the exploration of the search space [6]. In the mutation step, a donor vector v_i (mutant) is generated for each individual x_i. This is done by scaling difference vectors and adding them to a base vector. When vectors are chosen randomly in the mutation step, it is done uniformly at random from the current population without replacement. The magnitude of scaling is governed by the mutation rate F with $F > 0$. Many of the following mutation operators have a variant using one difference vector and a variant using two difference vectors. In the equations, these variants are differentiated through the blue coloured difference vector.

Rand/1 & Rand/2

Three different vectors, base vector x_{r1} and vectors x_{r2}, x_{r3} are uniformly at random selected from the population. A difference vector is taken of x_{r2}, x_{r3} which gets scaled by F and is added to x_{r1} [42]. An additional difference vector, calculated using x_{r4} and x_{r5} is added for Rand/2.

$$v_i \leftarrow x_{r1} + F \cdot (x_{r2} - x_{r3})[+F \cdot (x_{r4} - x_{r5})] \tag{1}$$

Best/1 & best/2

Equal to Rand/1 apart from the fact that the base vector is not chosen randomly. Instead, it is the fittest solution x_{best} in the current population [42].

$$v_i \leftarrow x_{best} + F \cdot (x_{r1} - x_{r2})[+F \cdot (x_{r3} - x_{r4})] \tag{2}$$

Target-to-$p$best/1

The base vector is taken from the target individual x_i for which this operator is generating a donor vector. Besides the usual scaled difference vector of randomly selected vectors x_{r1}, x_{r2}, a second scaled difference vector is calculated between x_i and x^p_{best}. Vector x^p_{best} is

selected uniformly at random from the best $p \cdot 100\%$ individuals in the current population with viable values $p \in (0, 1]$. The authors recommend using a new value for p each time this operator is called with $p \xleftarrow{u.a.r} [\frac{2}{M}, 0.2]$ [45].

$$v_i \leftarrow x_i + F \cdot (x_{best}^p - x_i) + F \cdot (x_{r1} - x_{r2}) \tag{3}$$

Rand-to-best/1 & Rand-to-best/2

A random vector x_{r1} that is not the current fittest solution x_{best} is selected as base vector in the rand-to-best operator. The first difference vector is generated by subtracting x_{r1} from x_{best} [42].

$$v_i \leftarrow x_{r1} + F \cdot (x_{best} - x_{r1}) + F \cdot (x_{r2} - x_{r3})[+F \cdot (x_{r4} - x_{r5})] \tag{4}$$

Target-to-best/1 & Target-to-best/2

Target-to-best [17] is a more basic version of Target-to-$p$best. Instead of choosing a vector from the $p \cdot 100\%$ best individuals it always chooses the best one.

$$v_i \leftarrow x_i + F \cdot (x_{best} - x_i) + F \cdot (x_{r1} - x_{r2}) + [F \cdot (x_{r3} - x_{r4})] \tag{5}$$

Target-to-rand/1 & Target-to-rand/2

The target-to-rand operator is similar to Target-to-best but without the scaled difference vector between x_{best} and x_i.

$$v_i \leftarrow x_i + F \cdot (x_{r1} - x_i) + F \cdot (x_{r2} - x_{r3})[+F \cdot (x_{r4} - x_{r5})] \tag{6}$$

2-Opt/1 & 2-Opt/2

This mutation operator is largely similar to Rand/1. However, before calculating v_i 2-opt makes a comparison between the fitness of x_{r1} and x_{r2} [10]. The fittest of these two is chosen as base vector while the other is used with x_{r3} in the creation of the scaled difference vector.

$$v_i \leftarrow \begin{cases} x_{r1} + F \cdot (x_{r2} - x_{r3})[+F \cdot (x_{r4} - x_{r5})] & f(x_{r1}) \leq f(x_{r2}) \\ x_{r2} + F \cdot (x_{r1} - x_{r3})[+F \cdot (x_{r4} - x_{r5})] & f(x_{r2}) < f(x_{r1}) \end{cases} \tag{7}$$

DESMU

The "Differential evolution algorithm using stochastic mutation" mutation operator (DESMU) is a target-to-rand/1 variant that uses values generated based levý-flight random walks lined out in equation 8 to generate a new value for F each time it is called [11].

$$step(s) = \frac{u}{|v|^{1/\alpha}}$$

$$u = N(0, \sigma_u^2), v = N(0, \sigma_v^2)$$

$$\sigma_u = \left(\frac{\Gamma(1+\alpha) sin(\pi\alpha/2)}{\alpha\Gamma[(1+\alpha)/2] 2^{(\alpha-1)/2}} \right)^{1/\alpha}, \sigma_(v) = 1$$

$$F_u = U(0, 1) \cdot step$$

$$v_i \leftarrow x_i + (F_u) \cdot (x_{r1} - x_{r2}) \tag{8}$$

Bacterial Evolutionary Algorithm

Each mutation step, $N_{clone} \geq 2$ clones are made where at most 1 clone is edited by the BEA scheme while others are modified by the DE scheme DE/target-to-rand/1 mentioned above.
For each individual, it and its clones are each divided into a number of segments N_s with length D/N_s. The mutation operator will go through each segment $k \in N_s$ clone-by-clone. For the first clone only it will for each segment randomly, with probability P_{BEA}, apply formula 9 on the indexes within that segment in order to do a local search. In other cases DE/target-to-rand/1 is used. Once all clones have their kth segment mutated the operator evaluates the clones. The best performing clone has their segment k copied into the other clones. This continues until all N_s segments have been mutated [3].

$$v_{i,j} \leftarrow x_{i,j} + [R_{min} + rand(0,1) \cdot (R_{max} - R_{min})] \tag{9}$$

Directional Mutation

The directional mutation operator extends an existing mutation scheme by adding a vector pool [48]. In our case this operator is DE/rand/1. The base state of this operator is to use this existing mutation scheme. The vector pool is updated only when a new global best fitness is reached. This update saves the difference vectors between the all previous positions and new best position to the vector pool. In the iteration after a fitness improvement mutation will not happen through the regular mutation scheme. Instead, the donor vector will be created by adding a random scaled difference vector from the vector pool to a random $x_{r1} \xleftarrow{u.a.r} P \backslash \{x_i\}$ from the population. After this the vector pool gets wiped clean, and the operator goes back to using the existing mutation scheme while again checking for a global fitness improvement, updating and using the vector pool again once this occurs.

Random vector

This mutation operator generates a random vector as donor vector within the box-constraints. With other parts of the DE algorithm still in place, using the random vector operator with a crossover rate of $F = 1.0$ would have the same effectiveness as random search.

3.3 Boundary constraints

After application of the mutation operator it is possible for values in v_i to be out of range for viable input values for f as defined by the box constraints. The boundary constraint operator decides how out-of-bounds values are altered to valid ranges.

Projection

If a value within v_i is infeasible, it gets projected back on the boundary. In other words, the infeasible value is set to the nearest feasible value [9].

Reflection

If a value within v_i is outside the feasible range by a certain distance $dist_{out}$ from the closest boundary cb, the value gets set to $dist_{out}$ on the other side of the boundary cb. Should the value then fall on the other side of the box constraint, the operator repeats until the value is in within the box constraint [39].

Reinitialisation

When reinitialisation is applied as boundary constraint operator, any infeasible values in v_i are discarded. Instead, these values are set to a random value within the box constraints [33].

Resampling

Resampling is a method where, if v_i is an infeasible solution, the mutation operator is reapplied until either a valid solution is generated or a maximum limit of resamples is reached [4]. In the latter case, it will fall back onto another method of boundary constraint handling. In our case, this is the Projection method.

3.4 Crossover operators

The crossover operator dictates the way trial vector x_i' is generated from individual x_i and its donor vector v_i. The following crossover operators have been implemented:

Binomial Crossover

This type of crossover uses a crossover rate $Cr \in [0, 1]$ and a random index j_{rand}. For each index, a uniform random value $\mathcal{U}(0, 1)$ is generated. If $\mathcal{U}(0, 1) \leq Cr$ the element will be taken from v_i, if $\mathcal{U}(0, 1) > Cr$ it will be taken from x_i. The exception is when the current index equals j_{rand}. In this latter case it will always take from v_i.

$$x'_{i,j} \leftarrow \begin{cases} v_{i,j} & \mathcal{U} \leq Cr \text{ or } j = j_{rand} \\ x_{i,j} & \text{all other cases} \end{cases} \tag{10}$$

From this one can see that with higher the values of Cr, on average more the more elements will be swapped [42].

Exponential Crossover

Just as binomial crossover, this operator also uses a crossover rate $Cr \in [0, 1]$ and a random index j_{rand}. It starts at index j_{rand} and takes from v_i for x'_i. From here on it then advances to the next index and generates a random number r. If $r \leq Cr$ it takes from v_i and continues to the next index and repeats with a newly generated r, looping to the start if the end was reached. If it generates a $r > Cr$, the operator stops walking through the indexes. All remaining values are taken from x_i. On average longer sections will be taken from v_i with larger values of Cr [44]. Pseudocode can be found below in Alg 2.

Algorithm 2: Pseudocode for Exponential Crossover

$j_{rand} \leftarrow \mathcal{U}_{int}[0, N]$ $x'_i = x_i$
$x'_{i,j_{rand}} = v_{i,j_{rand}}$
$j = j_{rand} + 1$
while $\mathcal{U}(0,1) \leq Cr$ ***and not*** $j = j_{rand}$ **do**
  $x'_{i,j} = v_{i,j}$ $j = (j+1)$ **mod** N

3.5 Selection operators

The selection operator decides which individuals from the existing candidate solutions and the newly generated trial vectors continue to the next generation. Few different selection operators exist [6]. This work implements only elitist selection. In this scheme each candidate solution $x_i \in P$ is only compared against its corresponding trial vector $x'i$. x_i in the next generation will be the fittest of these two vectors.

$$x_i \leftarrow \begin{cases} x_i & f(x_i) \leq f(x'_i) \\ x'_i & f(x_i) > f(x'_i) \end{cases} \tag{11}$$

4 Adaptive Operator Selection

The basic mechanisms of DE algorithm are easy to understand with only three control parameters: F, Cr, and NP. Despite this, their optimal values are highly problem dependent. Changing these control parameters can have a large impact on the performance of the DE [16] [27]. Manually tuning the control parameter values and/or changing between active operators (in future parts of this thesis referred to as *approach*) for each new problem can be a time-consuming process, and the optimal approach can also change over time. An Adaptive Operator Selection (AOS) strategy automates this process by allowing the algorithm to change (parts of) its approach during runtime. The ways an AOS can determine to switch approach can be divided into three main categories, according to [15]. The following subsections first describe these three categories with examples.

4.1 Deterministic

A deterministic strategy adapts its approach based on a predefined rule. It is possible to cycle through a number of approaches on a fixed budget or time interval. Another deterministic option is switching from an explorative approach like $DE/rand/1/bin$ to a more exploitative approach, such as $DE/best/1/bin$, when a certain amount of the budget is used. Or lowering F over time. Such annealed versions of DE were already theorised during its introduction [43].

4.2 Adaptive

Deterministic strategies give a good idea of what we want an AOS to be capable of, but they are very limited in how and when they can apply a change in approach. Adaptive parameter control strategies are able to change approaches based on direct feedback from the algorithm, such as credit values assigned to different approaches. Many well-known

strategies in DE fall in this category. This subsection describes Multi-Armed Bandits, EPSDE, and jDE.

The Multi-Armed Bandit [38] (MAB) starts with a portfolio, a selection of predefined approaches that can be switched between. For each approach, we keep track of a value we call Q. The starting value of Q should be equal for all approaches and can be set to 0, or an optimistic value. The exact value of the latter case should be dependent on what values the credit assignment operator can return. At the end of each learning period of lp iterations, each approach used is given a score R by a credit assignment operator based on its performance over this learning period. These scores are then used to update the Q values according to equation 12. In this formula, α ($0 \leq \alpha \leq 1$) determines how much emphasis is placed on recent scores compared to older scores. Recent scores are weighed more heavily at lower values of α. Through this, the MAB is able to track which approach is considered optimal during different stages of the search process.

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n] = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i \tag{12}$$

After all Q's are calculated, we start a new learning period by selecting new approaches for all individuals. The approaches are chosen either optimally based on the current values Q or, with a small chance, chosen at random from the portfolio.

Another adaptive AOS is EPSDE [25], which splits approaches into three different pools. The first pool contains operators that can be used. These being DE/rand/1/bin, DE/best/2/bin, and DE/current-to-rand/1/bin. The second pool contains values for F in the range of $[0.4, 0.9]$ in steps of 0.1. The final pool contains values for Cr in the range of $[0.1, 0.9]$, again in steps of 0.1. Each individual x_i uses a random value from each pool. Should the resulting trial vector x'_i improve upon x_i, then the values are kept. If not, a new random combination is drawn from the pools.

The final adaptive strategy we cover, jDE [9], uses the DE/rand/1/bin scheme throughout, but each individual is assigned a F and Cr which it keeps track of. Before the mutation operator is applied to an individual, its F and Cr values independently have a small chance (suggested 10%) to be reinitialised. This is done in the range of $[0.1, 1.0]$ and $[0.0, 1.0]$ respectively. Through the selection operator better values are propagated. Although it is similar to the next category, "self-adaptive", it does not apply the mutation or crossover operators to parameters F and Cr. As such, it cannot truly be classified as a true "self-adaptive" strategy according to the definitions from [15].

4.3 Self-adaptive

In self-adaptive strategies, details of the approach are encoded within the individuals. They are appended to and part of the same mutation, recombination, and selection process as the solution vector used to solve f. The Self-adaptive Pareto Differential Evolution (SPDE) [1] is an evolutionary multi-objective optimisation algorithm that builds upon Pareto-frontier Differential Evolution (PDE) [2]. They use the concept of pareto dominance to decide if an individual is allowed to generate offspring (see [29] for further

reading). SPDE, as its name implies, adds to PDE by randomly initialising F and Cr and having both rates be inheritable from the parents. After crossover, each variable is "perturbed by adding to it a ratio, $F \in Gaussian(0, 1)$, of the difference between the two values of this variable in the two supporting parents".

4.4 Credit Assignment

Some AOS mechanisms use a credit assignment operator to score approaches at the end of each learning period. As this thesis later implements one of these, the Multi-Armed Bandit, thought must be given to what credit assignment operator we use. Four variants were considered. The first operator, *Fitness*, assigns a credit score R equal to the fitness improvement made.

$$R \leftarrow \begin{cases} 0 & f(x_i) \leq f(x'_i) \\ x_i - x'_i & f(x_i) > f(x'_i) \end{cases} \tag{13}$$

The second operator, *Binary*, returns a score of 1 when x'_i improves upon x_i and zero otherwise.

$$R \leftarrow \begin{cases} 0 & f(x_i) \leq f(x'_i) \\ 1 & f(x_i) > f(x'_i) \end{cases} \tag{14}$$

The third operator, *TanH*, can be considered to sit in-between *Binary* and *Fitness*. It attempts to reduce the influence of large fitness improvements that often occur at the start of the search by taking the hyperbolic tangent of the fitness improvement. This results in credit roughly linear to fitness improvement in cases where the fitness improvement $f(x_i) - f(x'_i) < 1$, but a maximum R of 1 is asymptotically approached at larger improvements.

$$R \leftarrow \begin{cases} 0 & f(x_i) \leq f(x'_i) \\ tanh(x_i - x'_i) & f(x_i) > f(x'_i) \end{cases} \tag{15}$$

The last operator we name *Binary+*. It is the same as *Binary*, but as an extra case, returns a score of 10 if x'_i improves upon the global best achieved fitness [40].

$$R \leftarrow \begin{cases} 0 & f(x_i) \leq f(x'_i) \\ 1 & f(x_i) > f(x'_i) \\ 10 & f(x_{best}) > f(x'_i) \end{cases} \tag{16}$$

5 Experiments

This section is divided into three parts. First, we test the 16 mutation operators described in Section 3.2. This allows us to set the best performing settings for F and Cr as default parameters, and gives us baselines to compare against. Next, we implement AOS based on the MAB as described in Section 4.2 and find optimal hyperparameter values for the MAB. Lastly, we test portfolios of different sizes to see if and how portfolio size impacts the performance of our Multi-Armed Bandit.

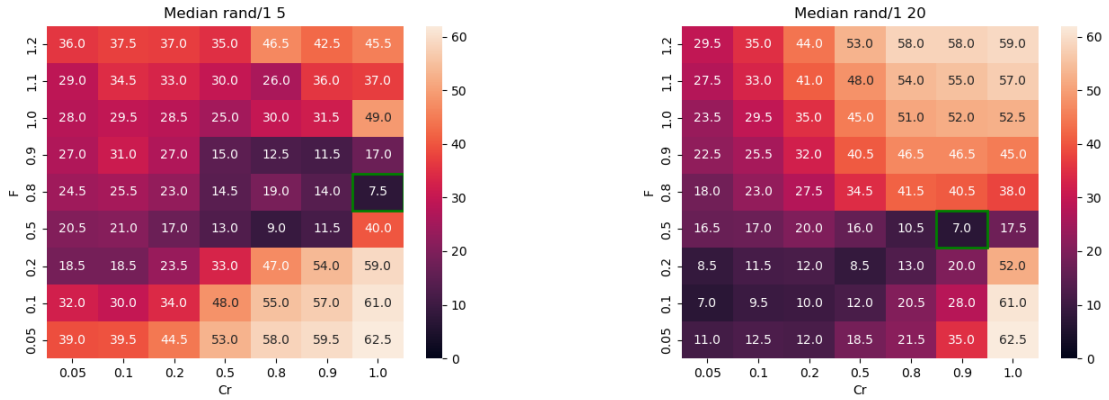

Figure 3: Heatmaps generated through applying median ranking to the individual 5- and 20-dimensional results of *DE/rand/1*. These show how different parameter combinations of F and Cr compare. A lower rank is better, a green square highlights the best performing combination.

5.1 Mutation Operator Baselines

The first experiment consists of a number of relatively low-budget runs done with the fixed adaptation scheme for every mutation operator. We want to verify two different things in this experiment. Firstly, we want to verify if different mutation operators have different performance on different problems. Secondly, we want to verify how different combinations of control parameters F and Cr impact performance.

Setup

For each single mutation operator, a grid search of F and Cr was conducted over $F = \{0.05, 0.1, 0.2, 0.5, 0.8, 0.9, 1.0, 1.1, 1.2\}$ and $Cr = \{0.05, 0.1, 0.2, 0.5, 0.8, 0.9, 1.0\}$. The population is initialised uniformly at random over the search space. Other important settings and settings changed from their default values have been listed below.

- Budget $\{2000N\}$
- Dimensions $\{5, 20\}$
- Functions $\{1, \ldots, 24\}$
- Instances per function $\{15\}$
- Crossover Operator {Binomial}
- Boundary Operator {Resample with Projection as fallback}
- Resample limit $\{10 + ln(N)^2\}$

Results

Using Median ranking, we created heatmaps like the ones in Figure 3 for all mutation operators. These heatmaps show how different combinations of F and Cr compare. The remaining median ranking heatmaps can be found in the appendix 9.
The parameter combination with the best median result is highlighted with a green square. Please note that it is possible that another combination exists with the same median

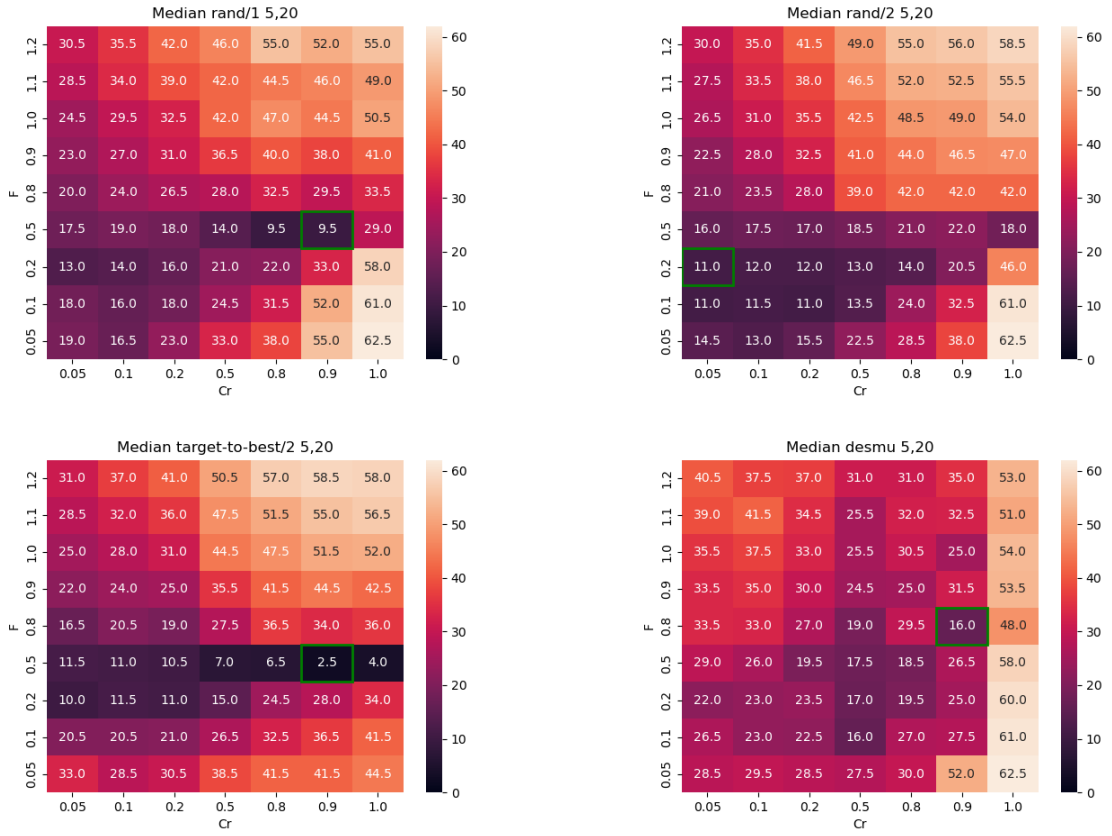

Figure 4: Heatmaps generated through applying median ranking to the averaged 5- and 20-dimensional results of the first experiment. These show how different parameter combinations of F and Cr compare. A lower rank is better, a green square highlights the best performing combination.

rank that is not highlighted. The default F and Cr values were selected using the best performing combination of the combined $d = 5$ and $d = 20$ median ranking, resulting in the following values:

	rand/1	rand/2	best/1	Best/2	ttpbest/1	rtbest/1	rtbest/2	ttbest/1	ttbest/2	ttrand/1	ttrand/2	2opt/1	2opt/2	Desmu	BEA	DirMut
F	0.5	0.2	0.5	0.2	0.5	0.2	0.5	0.5	0.5	0.5	0.5	0.5	0.2	0.1	0.5	0.05
Cr (Bin)	0.8	0.05	0.5	0.05	0.5	0.05	0.9	0.5	0.9	0.1	0.2	0.8	0.05	0.5	1.0	0.5

Some patterns are visible in this experiment. Mutation operators using more than 1 difference vector often prefer very low or high Cr values (0.05 or 0.9). It can also be seen that no single operator prefers a $F \geq 0.8$. The different behaviours of different mutation operators are also visible in Figure 3. Here, DE/target-to-best/2/bin seems to be strongly dependent on F. Favouring an F of 0.5. DE/rand/2/bin, on the other hand, seems to perform similarly on most combinations of low F and Cr.
Simultaneously, a test was run for the mutation operator that generates a random vector. Since there are no mutation rate parameters, only the effect of crossover on random search can be measured. There is a direct correlation between lower Cr and better rank for the random search mutation operator. This is not unexpected, since a low crossover rate allows for more regulated improvement. This is very desirable when trial vectors are generated that can be anywhere in the solution space with equal chance.

5.2 AOS parameters

we mentioned multiple AOS strategies in Section 4. For this work, the Multi-Armed Bandit strategy was implemented. There are five hyperparameters that can be tuned. The learning period length lp, the credit assignment operator, the measure of recency bias α, the strategy to selected operators, and the chance to instead select an operator uniformly at random ϵ. Of these five, we believe that there are three that are able to impact performance and behaviour more than the other two. The first of these three hyperparameters is the learning period lp. This determines how often the MAB should update the Q values and select new approaches. The second hyperparameter is the credit assignment operator, which is called at the end of every learning period to determine the new Q values. Last, we have the MAB selection strategy. The selection strategy, using the new Q values, decides what mutation operator is set on the individuals.

Setup

For the learning period, we are unsure how it affects performance and thus want to test a wide range of values. This leads us to the following range $lp = \{1, 5, 10, 20, 50, 100, 200\}$. We have detailed four simple, yet distinct, ways to assign credit to approaches in Section 4.4. All four of these will be tested. For the MAB selection strategy, we compare *ϵ-greedy* which with a chance of $1 - \epsilon$ selects the approach with the highest Q value, against a *proportional* strategy. With this proportional strategy $P_a = Q_a/Q_{total}$, where P_a is the chance for approach a being chosen, Q_a is the current Q value of a, and Q_{total} is the sum of the Q values of all approaches. In order to reduce computation and because we believe testing different values for these hyperparameters is less needed, we decided upon

the following fixed values for ϵ and α. An ϵ value that is too large would interfere with the selection strategy. A value that is too small would give the MAB very few chances to test not-chosen approaches, leading to slow convergence to the optimal current approach. A value of $\epsilon = 0.1$ should guarantee that a few individuals have randomly set approaches each learning period, but that the MAB selection strategy still selects approaches for the majority of the population. Then we have α, the measure of recency bias. This value exponentially decays Q values. A value close to 0 puts too much emphasis on the most recent learning period. A value close to 1 puts too much emphasis on old results, making it difficult to switch approaches. We therefore decided on $\alpha = 0.5$.

Settings not pertaining to the MAB or mutation operator are the same as the mutation operator baselines experiment, the budget was however increased to $10000N$. The results have been sorted based on the Area-Under-Curve of the Empirical Attainment Function (EAF) [24] implemented in IOHanalyzer. The EAF shows how the results of a stochastic algorithm are distributed. In other words, how likely it is for a given stochastic algorithm to obtain a given result within a given budget. Once ordered, each result was given a corresponding rank. Table 1 groups and shows the average rank of all results using the setting specified in the *Setting* column.

Results

There are $2 \cdot 4 \cdot 7 = 56$ possible MAB parameter combinations that were compared against the best performing versions of the 16 tested operators, the *Random Vector* operator, a *Random Search* (i.e. random operator with $Cr = 1.0$), and *Random Operator* which randomly selects from the viable approaches (i.e. MAB with $\epsilon = 1.0$) for a total of 75 results. We see that *Random Search* performs the worst (ranking starts at 0) with the *Random Vector* operator marginally better, as expected from the previous experiment. We can see that *Random Operator* ranks right in the middle, but looking at Tables 4 and 5 (in appendix) it is outperformed by a few single mutation operators from the baseline experiments in both 5- and 20-dimensional problems. These being *DE/best/2*, *DE/rand-to-best/2*, and *DE/2opt/2* for the 5-dimensional case and *DE/target-to-pbest/1* and *DE/DESMU* for the 20-dimensional case.

Comparing different MAB settings against each other, we can see that *ϵ-greedy* selection is very ineffective at low-dimensional problems compared to proportional selection. In the high-dimensional problems, its average rank was much closer to that of *Proportional* selection, but it was still outperformed. We suspect this to be caused by a combination of the low time to solve 5-dimensional problems, the ϵ value of 0.1, and the population size. *ϵ-greedy* likely could not obtain enough information about all operators quick enough to rapidly converge on a well performing operator for many of these problems. An issue that is less impactful on problems that take more time to solve, such as high dimensional problems where the population size is also larger.

The credit section mostly shows that raw fitness improvement is a poor way to assign credit. Other credit assignment methods have mixed results. The *Binary* approach performed best on low-dimensions, while *Binary+* performed best on high-dimensions. *Tanh* is somewhere in the middle, but not good enough to be selected. Since *Binary+* has a

	Setting	Count	avg Dim=5	avg Dim=20	avg Dim=5,20
comparisons	All	75	37.00	37.00	37.00
	MAB	56	31.55	28.93	30.24
	Fixed Operator	16	51.50	60.62	56.06
	Random Search	1	74.00	74.00	74.00
	Random Vector	1	72.00	73.00	72.50
	Random Operators	1	38.00	38.00	38.00
selection	ϵ-greedy	28	40.93	29.71	35.32
	Proportional	28	**22.18**	**28.14**	**25.16**
credit	Fitness	14	42.43	40.64	41.54
	TanH	14	28.64	25.64	27.14
	Binary	14	**25.07**	27.86	26.46
	Binary+	14	30.07	**21.57**	**25.82**
learning period	lp=1	8	27.88	29.88	28.88
	lp=5	8	**26.94**	**23.44**	**25.19**
	lp=10	8	31.94	25.62	28.78
	lp=20	8	41.69	33.00	37.34
	lp=50	8	36.25	32.12	34.19
	lp=100	8	41.75	36.00	38.88
	lp=200	8	52.50	46.12	49.31

Table 1: Raw results from Appendix Tables 4 and 5 were ranked $[0, 1, \ldots, 74]$ on dimensions 5 and 20 (lower is better). The top part of this table groups and averages all MAB runs, all 16 fixed operator runs and also shows the ranks of Random Vector, Random Search, and Random operators. The lower part of this table similarly groups the MAB results in three different ways in order to discern the best performing settings for each category, listed in bold. An additional visualisation can be found in Figure 10.

better average total on both dimensionalities, it was chosen as the credit assignment function for the last experiment. The learning period section shows that the MAB favours low learning periods. Despite this, a learning period of 1 is too low. A learning period of 5 performed best in both tested dimensionalities and was therefore chosen to continue with.

Looking more into the selection method and learning period, we see an interesting relation between selection method and learning period. It can be seen in Figure 5. The difference in performance on $N = 5$ comes mostly from runs with learning periods 5, 10, 20, and 50. The other runs are closer to the centre line, meaning that selection method mattered little for these learning periods. Looking at the data in tables 4 and 5, this seems to be caused by the performance of *proportional* selection moving towards that of *ϵ-greedy* rather than *ϵ-greedy* performing better for these learning periods. This pattern is much less pronounced on $N = 20$, as was expected from the results in Table 1. This suggest that selection strategy only matters if other hyperparameters are well-configured.

From this experiment we obtained the following MAB settings for the final experiment:

- $\epsilon = 0.1$
- $\alpha = 0.5$

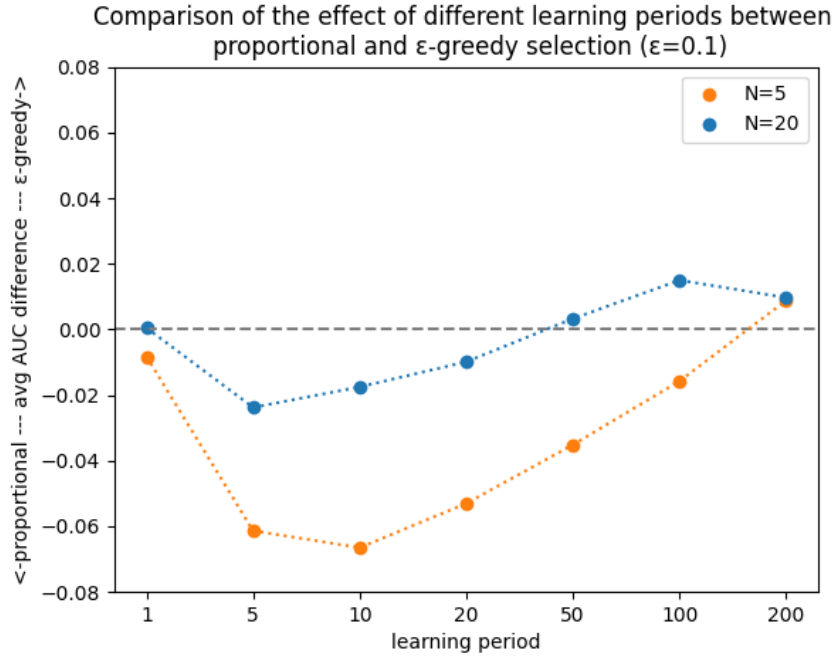

Figure 5: The effect of different learning periods on the two selection methods is analysed. The raw results have been divided based on selection method and were then grouped based on learning periods. These values show the average of the 4 runs in each group.

- Selection = Proportional
- Credit = Binary+
- Learning Period = 5

5.3 Portfolio Generation

Thanks to the previous two experiments, we were able to decide (hyper)parameters for the mutation operators and the MAB adaptive operator selector. From the previous experiments, we saw that with 15 instances, the stochastic nature of DE still made it possible for variations to occur that could cause the average of an individual configuration to rank much higher or lower than expected. Because of this, the 15 instances per function were increased to 50 instances per configuration for increased precision in the final part of this thesis. The generation of a portfolio comes with a large problem. Sixteen different mutation operators gives us 65535 different possible portfolios, too many to test within a reasonable timeframe. This is why two different strategies of portfolio generation have been used.

Setup

The first strategy is done *top-down*. Which means we start with a set of all operators. The next step is to have a set of c configurations, where each configuration has one different mutation operator o switched off (we call this 1-out). This switched-off operator cannot be chosen by the MAB. The best performing configuration in this series can be seen as the configuration where the switched-off operator o_{w0} contributed the least. The next set

consist of configurations that turn off o_{w0} and one of the other viable operators, $c-1$ configurations in total. From this we obtain a new worst performing operator o_{w1} which we now also always turn off for subsequent configurations sets. This was repeated until only one operator was left.
The second strategy is done the other way around, *bottom-up*. We start with a set of just one operator turned on at a time. The best performing operator o_{b0} is chosen, and the next set is each combination of o_{b0} plus an additional operator. This is again done until we are at the configuration with all operators enabled.
Using these two methods of building a portfolio reduces our number of operator combinations tested to a maximum of $(16+15+\cdots+2+1) = 136$ per strategy.

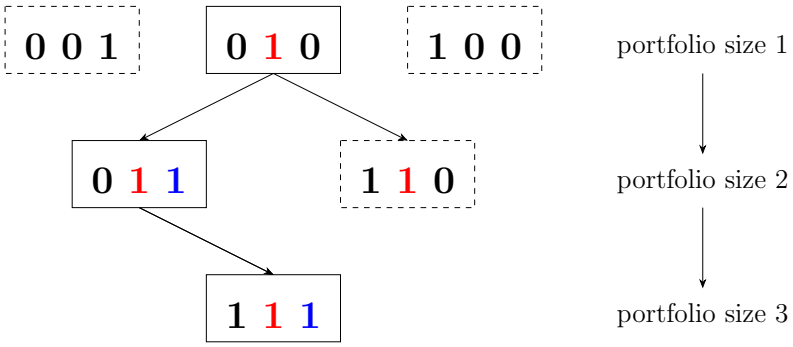

Figure 6: An illustration of what the bottom-up approach could look with 3 possible mutation operators. The combination of available operators is shown through a binary indicator (1 = can be used, 0 = cannot be used). The solid lined box indicates the best performing configuration for a given portfolio size. The best new operator is taken note of and set to be always turned on in subsequent rounds of the experiment.

Measuring performance

Just like the previous experiment, we use the EAF tool implemented in IOHanalyzer to score the performance of configurations. Unlike the previous experiment, ranking was not a clear indicator of the best performing portfolio for each size. The frequent occurrence of multi-way ties necessitated a different approach. A simple rating by averaging the AUC over both dimensionalities is also not possible. The best and worst AUC values on 5 dimensions are generally further apart than those on 20 dimensions. Measuring performance in this way would cause an unfair advantage for configurations that perform better on lower numbers of dimensions. In the end, it was decided to scale the obtained AUC-values for each number of dimensions and portfolio size step from 0.0 to 1.0. Taking the average of the scaled AUC-values gave both dimensionalities equal importance, while also taking into account how much configuration outperformed each other within a certain number of dimensions.

Results

From the tables 2 and 3 it can be seen that smaller portfolio sizes perform best for 20-dimensional problems. For the 5-dimensional problems, the values do not seem to show a clear trend. This is illustrated more clearly in Figure 7. We can also see that both approaches resulted in the same operator combinations for the portfolio's of size 1-4.

portfolio size	rand/1	rand/2	best/1	best/2	target-to-$p$best	rand-to-best/1	rand-to-best/2	target-to-best/1	target-to-best/2	target-to-rand/1	target-to-rand/2	two-opt/1	two-opt/2	DESMU	BEA	DIRMUT	AUC N=5/N=20
16																	0.802 / 0.555
15																	0.801 / 0.571
14																	0.807 / 0.575
13																	0.814 / 0.574
12																	**0.817** / 0.578
11																	0.813 / 0.582
10																	0.810 / 0.586
9																	0.813 / 0.590
8																	0.813 / 0.593
7																	0.813 / 0.593
6																	0.812 / 0.602
5																	0.813 / 0.602
4																	0.812 / 0.598
3																	0.806 / **0.613**
2																	0.809 / 0.589
1																	0.798 / 0.578

Table 2: Top-down approach.
Coloured cells show the best performing selection of operators for that portfolio size.

portfolio size	rand/1	rand/2	best/1	best/2	target-to-$p$best	rand-to-best/1	rand-to-best/2	target-to-best/1	target-to-best/2	target-to-rand/1	target-to-rand/2	two-opt/1	two-opt/2	DESMU	BEA	DIRMUT	AUC N=5/N=20
1																	0.798 / 0.578
2																	0.809 / 0.589
3																	0.806 / 0.601
4																	0.805 / **0.613**
5																	0.813 / 0.598
6																	**0.821** / 0.600
7																	0.816 / 0.591
8																	0.818 / 0.588
9																	0.819 / 0.584
10																	0.812 / 0.583
11																	0.816 / 0.583
12																	0.811 / 0.583
13																	0.809 / 0.581
14																	0.803 / 0.574
15																	0.801 / 0.571
16																	0.802 / 0.555

Table 3: Bottom-up approach.
Coloured cells show the best performing selection of operators for that portfolio size.

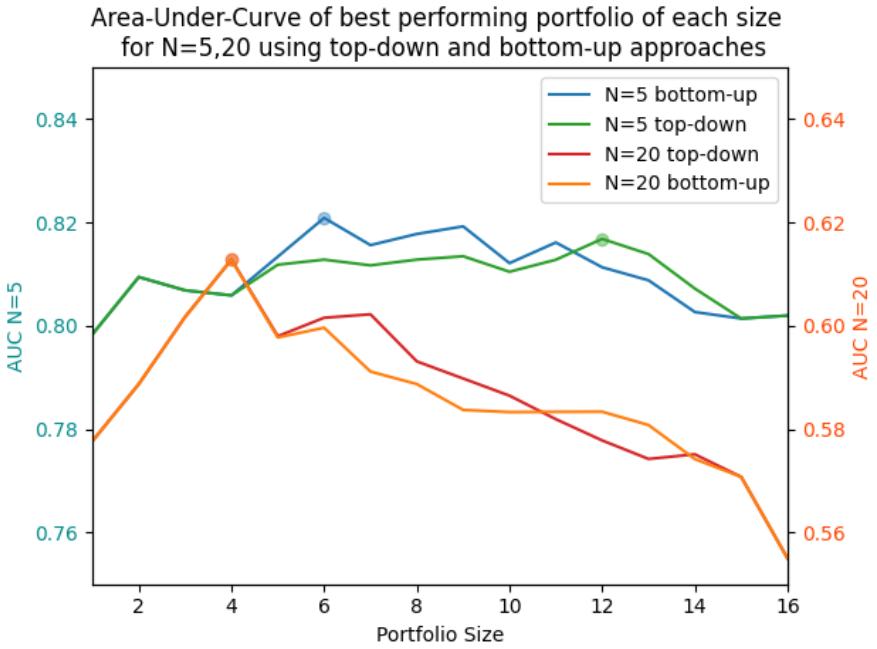

Figure 7: The obtained EAF AUC values found in tables 2 and 3 have been plotted to illustrate how performance changes with portfolio size. A dot has been placed to show the best performing portfolio found for each dimensionality with each strategy.

In the line plot shown in Figure 7, the lines for the 20-dimensional experiments show that the use of a small portfolio with multiple mutation operators is clearly better than using a single mutation operator. Once more operators are added, the positive effect diminishes and the performance can even go below the performance of a single operator. The 5-dimensional experiments show a small improvement when more than one operator can be selected. In contrast to N=20 however, there does not seem to be much of a drop in effectiveness of larger portfolio sizes until nearly all possible operators have been added to the portfolio.

In order to verify the effect of the MAB a few bar charts were made. In Figure 8 it can be seen that, using the found optimal 2-operator portfolio, the two operators were used in different amounts on different functions. This is especially notable for the functions belonging to the category "Functions with high conditioning and unimodal", f10-f14. From the 16-operator portfolio, we can see some difference in MAB behaviour between the two dimensions. The operators *rand/2* and *rand-to-best/1* had mediocre Q-values in $N = 5$ but had above average Q-values in $N = 20$, meaning in the latter case they regularly led to improvement. Next, in $N = 5$ operators *target-to-best/2* and *rand-to-best/2* are selected very frequently for problems f10-f14. In $N = 20$ they still have an above average chance of being chosen, but the gap between other operators is a lot less. A final thing that stood out was the operator *2opt/1*. It performed average in $N = 20$, but in $N = 5$ it frequently has above average chances of being chosen. Despite this, it was discarded halfway during the top-down approach and added as 11'th operator during the bottom-up approach. This likely means it has a very similar performance profile to respectively one of the remaining or one of the earlier added operators. Not what we are looking for when

Figure 8: These bar charts show relative Q-values. These indicate for each function how likely it was for each operator to be picked, averaged over all instances and points in time. From top to bottom, the bar plots show the likelihoods for the best portfolio's of sizes 2, 4, and 16. The binary string in the title of each figure can be mapped to the operators as ordered in earlier tables, such as Tables 2 and 3

constructing a portfolio with complementary operators.

6 Conclusion

This thesis investigated the impact of Adaptive Operator Selection and portfolio size in the context of Differential Evolution. It did so using a modular C++ implementation that utilizes a Multi-Armed Bandit which tracks the performance of mutation operators and accordingly adjusts which of the up to 16 mutation operators are used at which points in the search.

Sixteen mutation operators have been run with a wide range of parameter settings. Many combinations of mutation rates (F) and crossover rates (Cr) were tested, showing how different mutation operators prefer certain values to other values when no adaptation is applied. From these results, we saw that values of F≥ 0.8 were not competitive. We also saw that some mutation operators, such as *rand/2*, have a wide range of parameter combinations that perform about equally well. Others, like *target-to-best/2*, have only a few combinations or sometimes even just one with which they achieve their best performance compared to other combinations.

A Multi-Armed Bandit was implemented to serve as adaptive operator selection strategy. There are five hyperparameters that can be changed, three of which have been tested. *Proportional* operator selection was found to clearly outperform *ϵ-greedy* selection in 5-dimensional problems, while the performance of both was a lot closer on 20-dimensional problems. A learning period of 5 was the optimal found value for both 5- and 20-dimensional problems. Further investigation can be done into learning periods within the range $[1, 10]$. The true optimal value may not be necessarily 5 itself, but it is very likely to be in the aforementioned range. We also expect some sort of correlation between the optimal learning period and the in this thesis untested hyperparameter α and the population size. The latter of which was within suggested ranges for high-dimensional problems but in retrospect on the smaller side for 5-dimensional problems since no minimum population size was implemented. This smaller population allowing for less data points when calculating the MAB's Q-values. In contrast to the learning period and operator selection method, the credit assignment had no clear best option. The *Binary* credit assigner performed best on 5-dimensional problems and worst on 20-dimensional problems with *Binary+* having opposite results. Due to a better average, *Binary+* was eventually chosen. More intricate credit metrics exist, some of which have been implemented in [7]. These take into account more aspects than just the fitness improvement and might outperform the tested credit metrics in this thesis. Testing these different configurations showed that decently configured Multi-Armed Bandits performed better than any single mutation operator or randomly switching between mutation operators.

With the Multi-Armed Bandit, it was possible to easily to test different portfolios by changing a single passed hyperparameter. With this, two methods of portfolio generation were used to test the effect of portfolio size on performance. One approach that starts at the largest possible portfolio size and continuously removes the mutation operator that contributes the least. The other starts from the single best performing mutation operator

and adds the operator that contributes the most to the portfolio of one size larger. Results show that for the low-dimensional case, performance drops slightly for very high and very small portfolio sizes. The best performance occured in the rather large portfolio size range of 6 to 13, with slight variations depending on the portfolio construction method used. The high-dimensional case shows a much stronger effect. Both approaches resulted in the same optimal portfolio with a size of 4. The addition or removal of mutation operator here resulted in steadily declining performance. These four operators are *target-to-best/2*, *target-to-pbest/1*, *rand-to-best/2*, and *DESMU*. It is not unexpected that the way a portfolio is constructed has a larger effect when the problems are overall harder to solve. Having more operators means more options to weigh against each other, which seems to impact performance much more in these higher-dimensional cases. This means that if one wishes to generate a portfolio for similar a case or even higher dimensional cases, considerable time can be saved by excluding small and large portfolio sizes. We suggest only testing portfolio's with 4 to 6 mutation operators.

Inspecting the order in which the mutation operators were chosen or eliminated in the two approaches shows that the well known and first-suggested operators *best/1* and *rand/1* had lower than average contribution to portfolio performance. The contribution of more mechanically complex operators ranged from very good to very bad. It may therefore be worthwhile to test how other novel mutation operators impact portfolio performance. It is also not always predictable if an operator performs better with one or two difference vectors. It likely depends a lot on the rest of the portfolio. Since adding or removing a difference vector is often trivial, implementing versions with different amount of difference vectors can be worth investigating when creating or implementing any mutation operator. Finally, there is an avenue this thesis could not delve into but is still interesting to explore. At which point in the search process are certain operators chosen? The wildly differing lengths of runs made this tricky to explore, but it could give more insight into why certain mutation operators were eliminated, if and how certain roles can be assigned to different operators and give opportunities to decide F and Cr values more effectively.

7 Code

The code used to run the experiments can be found in the following GitHub repository: `https://github.com/PerH3R/basicDE`.

References

[1] H.A. Abbass. The self-adaptive pareto differential evolution algorithm. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 1, pages 831–836 vol.1, 2002.

[2] H.A. Abbass, R. Sarker, and C. Newton. PDE: a pareto-frontier differential evolution approach for multi-objective optimization problems. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, volume 2, pages 971–978 vol. 2, 2001.

[3] Rawaa Dawoud Al-Dabbagh, Janos Botzheim, and Mohanad Dawood Al-Dabbagh. Comparative analysis of a modified differential evolution algorithm based on bacterial mutation scheme. In *2014 IEEE Symposium on Differential Evolution (SDE)*, pages 1–8. IEEE, 2014.

[4] Jarosław Arabas, Adam Szczepankiewicz, and Tomasz Wroniak. Experimental comparison of methods to handle boundary constraints in differential evolution. In Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, pages 411–420, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[5] Michael J Best. *Portfolio optimization*. CRC Press, 2010.

[6] Bilal, Millie Pant, Hira Zaheer, Laura Garcia-Hernandez, and Ajith Abraham. Differential evolution: A review of more than two decades of research. *Engineering Applications of Artificial Intelligence*, 90:103479, 2020.

[7] Rick Boks. Dynamic configuration of operators and parameters in differential evolution through combined fitness and diversity-driven adaptation methods. Master's thesis, Leiden University, Niels Bohrweg 1 2333 CA Leiden, 7 2021.

[8] Koen Bouwman. Solving multi-agent reinforcements learning problems using differential evolution. Master's thesis, Leiden University, Niels Bohrweg 1 2333 CA Leiden, 2 2024.

[9] Janez Brest, Sao Greiner, Borko Boskovic, Marjan Mernik, and Viljem Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[10] Cheng-Wen Chiang, Wei-Ping Lee, and Jia-Sheng Heh. A 2-opt based differential evolution for global optimization. *Applied Soft Computing*, 10(4):1200–1207, 2010.

[11] Nikky Choudhary, Harish Sharma, and Nirmala Sharma. Differential evolution algorithm using stochastic mutation. In *2016 International Conference on Computing, Communication and Automation (ICCCA)*, pages 315–320. IEEE, 2016.

[12] Erik Cuevas, Daniel Zaldívar, Marco Perez-Cisneros, Erik Cuevas, Daniel Zaldívar, and Marco Perez-Cisneros. Image segmentation based on differential evolution optimization. *Applications of evolutionary computation in image processing and pattern recognition*, pages 9–22, 2016.

[13] Jacob de Nobel, Furong Ye, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. IOHexperimenter: Benchmarking platform for iterative optimization heuristics, 2022.

[14] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics, 2018.

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

[16] R. Gaemperle, S. D. Mueller, and P. Koumoutsakos. A parameter study for differential evolution. In *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, page 293–298. WSEAS, WSEAS, 2002. Proceedings of the 3rd WSEAS International Conference on Neural Networks and Applications (NNA '02), Fuzzy Sets and Fuzzy Systems (FSFS '02), Evolutionary Computation (EC '02).

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

[18] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. COCO: a platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, August 2020.

[19] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA, 2009.

[20] Lester Ingber et al. Adaptive simulated annealing (ASA). *Global optimization C-code, Caltech Alumni Association, Pasadena, CA*, 1993.

[21] Rajesh Kumar and Arun Kumar. Application of differential evolution for wind speed distribution parameters estimation. *Wind Engineering*, 45(6):1544–1556, 2021.

[22] Jing Liang, B. Qu, and Ponnuthurai Suganthan. Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. Technical report, 12 2013.

[23] Tao Liu and Zhongyang Yu. The analysis of financial market risk based on machine learning and particle swarm optimization algorithm. *EURASIP Journal on Wireless Communications and Networking*, 2022(1):31, 2022.

[24] Manuel López-Ibáñez, Luís Paquete, and Thomas Stützle. Exploratory analysis of stochastic local search algorithms in biobjective optimization. In Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 209–222. Springer, Berlin/Heidelberg, 2010.

[25] R. Mallipeddi, P.N. Suganthan, Q.K. Pan, and M.F. Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11(2):1679–1696, 2011. The Impact of Soft Computing for the Progress of Artificial Intelligence.

[26] Heitor R. Medeiros, Diogo M. F. Izidio, Antonyus P. do A. Ferreira, and Edna N. da S. Barros. Latin hypercube initialization strategy for design space exploration of deep neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, page 295–296, New York, NY, USA, 2019. Association for Computing Machinery.

[27] Efrñn Mezura-Montes, Jesús Velázquez-Reyes, and Carlos 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.

[28] Jr. Michael N. Katehakis, Arthur F. Veinott. The multi-armed bandit problem: Decomposition and computation. *Mathematics of Operations Research*, 12:262–268, 1987.

[29] Patrick Ngatchou, Anahita Zarei, and A El-Sharkawi. Pareto multi objective optimization. In *Proceedings of the 13th international conference on, intelligent systems application to power systems*, pages 84–91. IEEE, 2005.

[30] Magnus Erik Hvass Pedersen. Good parameters for differential evolution. *Magnus Erik Hvass Pedersen*, 49, 2010.

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

[32] William H Press. Numerical recipes in C, 1992.

[33] K. Price, Rainer Storn, and J. Lampinen. *Differential Evolution-A Practical Approach to Global Optimization*, volume 141. Springer Berlin, Heidelberg, 01 2005.

[34] Shahryar Rahnamayan, Hamid R Tizhoosh, and Magdy MA Salama. Opposition-based differential evolution. *IEEE Transactions on Evolutionary computation*, 12(1):64–79, 2008.

[35] Shahryar Rahnamayan and G Gary Wang. Center-based initialization for large-scale black-box problems. In *Proceedings of the 8th WSEAS international conference on Artificial intelligence, knowledge engineering and data bases*, pages 531–541, 2009.

[36] Elena Raponi, Nathanaël Carraz Rakotonirina, Jérémy Rapin, Carola Doerr, and Olivier Teytaud. Optimizing with low budgets: A comparison on the black-box optimization benchmarking suite and openai gym. *IEEE Transactions on Evolutionary Computation*, 2023.

[37] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*, 2018.

[38] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2018.

[39] 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.

[40] Mudita Sharma, Alexandros Komninos, Manuel Lopez Ibanez, and Dimitar Kazakov. Deep reinforcement learning based parameter control in differential evolution, 2019.

[41] Zack Stokes, Abhyuday Mandal, and Weng Kee Wong. Using differential evolution to design optimal experiments. *Chemometrics and Intelligent Laboratory Systems*, 199:103955, 2020.

[42] Rainer Storn. On the usage of differential evolution for function optimization. In *Proceedings of North American fuzzy information processing*, pages 519–523. Ieee, 1996.

[43] Rainer Storn and Kenneth Price. Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces. *International computer science institute*, 1995.

[44] Rainer Storn and Kenneth Price. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341, 1997.

[45] Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. pages 71–78, 06 2013.

[46] Adi L Tarca, Vincent J Carey, Xue-wen Chen, Roberto Romero, and Sorin Drăghici. Machine learning and its applications to biology. *PLOS Computational Biology*, 3(6):1–11, 06 2007.

[47] Hao Wang, Diederick Vermetten, Furong Ye, Carola Doerr, and Thomas Bäck. IOH-analyzer: Detailed performance analyses for iterative optimization heuristics. *ACM Trans. Evol. Learn. Optim.*, 2(1), apr 2022.

[48] Xin Zhang and Shiu Yin Yuen. A directional mutation operator for differential evolution algorithms. *Applied soft computing*, 30:529–548, 2015.

ID	Runtime	AUC	AOC
rand/1_F0.5_Bin_Cr0.8	50000	0.696543237268291	0.303456762731709
rand/2_F0.2_Bin_Cr0.05	50000	0.707929893803196	0.292070106196804
best/1_F0.5_Bin_Cr0.5	50000	0.678354600053396	0.321645399946604
best/2_F0.2_Bin_Cr0.05	50000	0.735399519068578	0.264600480931422
ttpb/1_F0.5_Bin_Cr0.9	50000	0.689540943519795	0.310459056480205
rand-to-best/1_F0.2_Bin_Cr0.05	50000	0.448077239334064	0.551922760665936
rand-to-best/2_F0.5_Bin_Cr0.9	50000	0.758026719901712	0.241973280098288
target-to-best/1_F0.5_Bin_Cr0.5	50000	0.691947014399355	0.308052985600645
target-to-best/2_F0.5_Bin_Cr0.9	50000	0.737981318040853	0.262018681959147
target-to-rand/1_F0.5_Bin_Cr0.1	50000	0.627398195922547	0.372601804077453
target-to-rand/2_F0.5_Bin_Cr0.2	50000	0.626122622427706	0.373877377572294
2opt/1_F0.5_Bin_Cr0.8	50000	0.711010376327307	0.288989623672693
2opt/2_F0.2_Bin_Cr0.05	50000	0.718622605841749	0.281377394158251
desmu_F0.1_Bin_Cr0.5	50000	0.703177844257761	0.296822155742239
BEA_F0.5_Bin_Cr1.0	50000	0.48440363690643	0.51559636309357
DIRMUT_F0.05_Bin_Cr0.5	50000	0.535784297100179	0.464215702899821
MAB_lp1eps0.1sel0crd0	50000	0.736962238290841	0.263037761709159
MAB_lp5eps0.1sel0crd0	50000	0.711141095076916	0.288858904923084
MAB_lp10eps0.1sel0crd0	50000	0.700802658156848	0.299197341843152
MAB_lp20eps0.1sel0crd0	50000	0.68513305527002	0.31486694472998
MAB_lp50eps0.1sel0crd0	50000	0.685794421018079	0.314205578981921
MAB_lp100eps0.1sel0crd0	50000	0.685777681612254	0.314222318387746
MAB_lp200eps0.1sel0crd0	50000	0.687306929157644	0.312693070842356
MAB_lp1eps0.1sel0crd1	50000	0.736962238290841	0.263037761709159
MAB_lp5eps0.1sel0crd1	50000	0.724945061959387	0.275054938040613
MAB_lp10eps0.1sel0crd1	50000	0.730981465825348	0.269018534174652
MAB_lp20eps0.1sel0crd1	50000	0.727480481586755	0.272519518413245
MAB_lp50eps0.1sel0crd1	50000	0.730739491311788	0.269260508688212
MAB_lp100eps0.1sel0crd1	50000	0.71461570725024	0.28538429274976
MAB_lp200eps0.1sel0crd1	50000	0.701632261738575	0.298367738261425
MAB_lp1eps0.1sel0crd2	50000	0.736962238290841	0.263037761709159
MAB_lp5eps0.1sel0crd2	50000	0.758937934621327	0.241062065378673
MAB_lp10eps0.1sel0crd2	50000	0.749733852439518	0.250266147560482
MAB_lp20eps0.1sel0crd2	50000	0.734122672090227	0.265877327909773
MAB_lp50eps0.1sel0crd2	50000	0.702151388185121	0.297848611814879
MAB_lp100eps0.1sel0crd2	50000	0.704845303232058	0.295154696767942
MAB_lp200eps0.1sel0crd2	50000	0.709123170013991	0.290876829986009
MAB_lp1eps0.1sel0crd3	50000	0.745147438266652	0.254852561733348
MAB_lp5eps0.1sel0crd3	50000	0.712569786635573	0.287430213364427
MAB_lp10eps0.1sel0crd3	50000	0.685575864696387	0.314424135303613
MAB_lp20eps0.1sel0crd3	50000	0.680821604773435	0.319178395226565
MAB_lp50eps0.1sel0crd3	50000	0.696720467283956	0.303279532716044
MAB_lp100eps0.1sel0crd3	50000	0.710270123704324	0.289729876295676
MAB_lp200eps0.1sel0crd3	50000	0.711489227213254	0.288510772786746
random_operators	50000	0.713317169035616	0.286682830964384
Random_search	50000	0.436310993689696	0.563689006310304
Random_vectors	50000	0.463464235121277	0.536535764878723
MAB_lp1eps0.1sel1crd0	50000	0.741562268297277	0.258437731702723
MAB_lp5eps0.1sel1crd0	50000	0.78763158446953	0.21236841553047
MAB_lp10eps0.1sel1crd0	50000	0.7627523321878	0.2372476678122
MAB_lp20eps0.1sel1crd0	50000	0.729412776344628	0.270587223655372
MAB_lp50eps0.1sel1crd0	50000	0.71114566851275	0.28885433148725
MAB_lp100eps0.1sel1crd0	50000	0.693979657679584	0.306020342320416
MAB_lp200eps0.1sel1crd0	50000	0.677163254405933	0.322836745594067
MAB_lp1eps0.1sel1crd1	50000	0.741562268297277	0.258437731702723
MAB_lp5eps0.1sel1crd1	50000	0.784280128479451	0.215719871520549
MAB_lp10eps0.1sel1crd1	50000	0.787338635068493	0.212661364931507
MAB_lp20eps0.1sel1crd1	50000	0.75889447857906	0.24110552142094
MAB_lp50eps0.1sel1crd1	50000	0.73237623113157	0.26762376886843
MAB_lp100eps0.1sel1crd1	50000	0.716384257388832	0.283615742611168
MAB_lp200eps0.1sel1crd1	50000	0.693311464111736	0.306688535888264
MAB_lp1eps0.1sel1crd2	50000	0.741562268297277	0.258437731702723
MAB_lp5eps0.1sel1crd2	50000	0.793597780972807	0.206402219027193
MAB_lp10eps0.1sel1crd2	50000	0.794496838992993	0.205503161007007
MAB_lp20eps0.1sel1crd2	50000	0.775951109573724	0.224048890426276
MAB_lp50eps0.1sel1crd2	50000	0.752338782172117	0.247661217827883
MAB_lp100eps0.1sel1crd2	50000	0.725494093697743	0.274505906302257
MAB_lp200eps0.1sel1crd2	50000	0.696837580966172	0.303162419033828
MAB_lp1eps0.1sel1crd3	50000	0.76450214800112	0.23549785199888
MAB_lp5eps0.1sel1crd3	50000	0.787513090195196	0.212486909804804
MAB_lp10eps0.1sel1crd3	50000	0.788511431849631	0.211488568150369
MAB_lp20eps0.1sel1crd3	50000	0.775310723710197	0.224689276289803
MAB_lp50eps0.1sel1crd3	50000	0.760135179061711	0.239864820938289
MAB_lp100eps0.1sel1crd3	50000	0.742540003356616	0.257459996643384
MAB_lp200eps0.1sel1crd3	50000	0.706977868597565	0.293022131402435

Table 4: Raw EAF AUC/AOC results for Dim=5 for experiment set 2

ID	Runtime	AUC	AOC
rand/1_F0.5_Bin_Cr0.8	2e+05	0.501585699420824	0.498414300579176
rand/2_F0.1_Bin_Cr0.05	2e+05	0.4693546288934	0.5306453711066
best/1_F0.5_Bin_Cr0.5	2e+05	0.528220478769292	0.471779521230708
best/2_F0.1_Bin_Cr0.05	2e+05	0.503984542137303	0.496015457862697
ttpb/1_F0.5_Bin_Cr0.9	2e+05	0.53570239077805	0.46429760922195
rand-to-best/1_F0.1_Bin_Cr0.05	2e+05	0.400210375792009	0.599789624207991
rand-to-best/2_F0.5_Bin_Cr0.9	2e+05	0.508528603915042	0.491471396084958
target-to-best/1_F0.5_Bin_Cr0.5	2e+05	0.509377898122106	0.490622101877894
target-to-best/2_F0.5_Bin_Cr0.9	2e+05	0.503297388620418	0.496702611379582
target-to-rand/1_F0.5_Bin_Cr0.1	2e+05	0.506747562546482	0.493252437453518
target-to-rand/2_F0.5_Bin_Cr0.2	2e+05	0.504156837541553	0.495843162458447
2opt/1_F0.5_Bin_Cr0.8	2e+05	0.510143468174572	0.489856531825428
2opt/2_F0.2_Bin_Cr0.05	2e+05	0.475821785099531	0.524178214900469
desmu_F0.1_Bin_Cr0.5	2e+05	0.535816779506836	0.464183220493164
BEA_F0.5_Bin_Cr1.0	2e+05	0.397856634714165	0.602143365285835
DIRMUT_F0.05_Bin_Cr0.5	2e+05	0.453934531857812	0.546065468142188
MAB_lp1eps0.1sel0crd0	2e+05	0.528648112723088	0.471351887276912
MAB_lp5eps0.1sel0crd0	2e+05	0.528259572306492	0.471740427693508
MAB_lp10eps0.1sel0crd0	2e+05	0.529844805056135	0.470155194943865
MAB_lp20eps0.1sel0crd0	2e+05	0.524635406665549	0.475364593334451
MAB_lp50eps0.1sel0crd0	2e+05	0.520122537887269	0.479877462112731
MAB_lp100eps0.1sel0crd0	2e+05	0.51923090961422	0.48076909038578
MAB_lp200eps0.1sel0crd0	2e+05	0.522801752503362	0.477198247496638
MAB_lp1eps0.1sel0crd1	2e+05	0.528648112723088	0.471351887276912
MAB_lp5eps0.1sel0crd1	2e+05	0.545114846849625	0.454885153150375
MAB_lp10eps0.1sel0crd1	2e+05	0.54512215576578	0.45487784423422
MAB_lp20eps0.1sel0crd1	2e+05	0.544285460791489	0.455714539208511
MAB_lp50eps0.1sel0crd1	2e+05	0.541256214639757	0.458743785360243
MAB_lp100eps0.1sel0crd1	2e+05	0.554542551054046	0.445457448945954
MAB_lp200eps0.1sel0crd1	2e+05	0.537198216419733	0.462801783580267
MAB_lp1eps0.1sel0crd2	2e+05	0.528648112723088	0.471351887276912
MAB_lp5eps0.1sel0crd2	2e+05	0.54379846500382	0.45620153499618
MAB_lp10eps0.1sel0crd2	2e+05	0.542185113759215	0.457814886240785
MAB_lp20eps0.1sel0crd2	2e+05	0.54587780360194	0.45412219639806
MAB_lp50eps0.1sel0crd2	2e+05	0.547003131618204	0.452996868381796
MAB_lp100eps0.1sel0crd2	2e+05	0.548031721449288	0.451968278550712
MAB_lp200eps0.1sel0crd2	2e+05	0.536156077866241	0.463843922133759
MAB_lp1eps0.1sel0crd3	2e+05	0.545018025098785	0.454981974901215
MAB_lp5eps0.1sel0crd3	2e+05	0.545693756567485	0.454306243432515
MAB_lp10eps0.1sel0crd3	2e+05	0.54760802593508	0.45239197406492
MAB_lp20eps0.1sel0crd3	2e+05	0.545686015221541	0.454313984778459
MAB_lp50eps0.1sel0crd3	2e+05	0.547409836921885	0.452590163078115
MAB_lp100eps0.1sel0crd3	2e+05	0.544725060386802	0.455274939613198
MAB_lp200eps0.1sel0crd3	2e+05	0.533764950830772	0.466235049169228
random_operators	2e+05	0.531210356979749	0.468789643020251
Random_search	2e+05	0.365667830796954	0.634332169203046
Random_vectors	2e+05	0.379220527834085	0.620779472165915
MAB_lp1eps0.1sel1crd0	2e+05	0.527943262929114	0.472056737070886
MAB_lp5eps0.1sel1crd0	2e+05	0.574089816150028	0.425910183849972
MAB_lp10eps0.1sel1crd0	2e+05	0.565270259930928	0.434729740069072
MAB_lp20eps0.1sel1crd0	2e+05	0.547455644387656	0.452544355612344
MAB_lp50eps0.1sel1crd0	2e+05	0.52709276656883	0.47290723343117
MAB_lp100eps0.1sel1crd0	2e+05	0.521019913013061	0.478980086986939
MAB_lp200eps0.1sel1crd0	2e+05	0.519388049659748	0.480611950340252
MAB_lp1eps0.1sel1crd1	2e+05	0.527943262929114	0.472056737070886
MAB_lp5eps0.1sel1crd1	2e+05	0.571254240338092	0.428745759661908
MAB_lp10eps0.1sel1crd1	2e+05	0.563751297958526	0.436248702041474
MAB_lp20eps0.1sel1crd1	2e+05	0.554120054510833	0.445879945489167
MAB_lp50eps0.1sel1crd1	2e+05	0.541861535412652	0.458138464587348
MAB_lp100eps0.1sel1crd1	2e+05	0.52825254435036	0.47174745564964
MAB_lp200eps0.1sel1crd1	2e+05	0.522035996834346	0.477964003165654
MAB_lp1eps0.1sel1crd2	2e+05	0.527943262929114	0.472056737070886
MAB_lp5eps0.1sel1crd2	2e+05	0.55533355036052	0.44466644963948
MAB_lp10eps0.1sel1crd2	2e+05	0.55247559430211	0.44752440569789
MAB_lp20eps0.1sel1crd2	2e+05	0.545593295711293	0.454406704288707
MAB_lp50eps0.1sel1crd2	2e+05	0.535481147879869	0.464518852120131
MAB_lp100eps0.1sel1crd2	2e+05	0.525938445476062	0.474061554523938
MAB_lp200eps0.1sel1crd2	2e+05	0.523009275673036	0.476990724326964
MAB_lp1eps0.1sel1crd3	2e+05	0.545269534133	0.454730465867
MAB_lp5eps0.1sel1crd3	2e+05	0.557282771203944	0.442717228796056
MAB_lp10eps0.1sel1crd3	2e+05	0.553344854807727	0.446655145192273
MAB_lp20eps0.1sel1crd3	2e+05	0.552794218790989	0.447205781209011
MAB_lp50eps0.1sel1crd3	2e+05	0.538222094293331	0.461777905706669
MAB_lp100eps0.1sel1crd3	2e+05	0.531362323673265	0.468637676326735
MAB_lp200eps0.1sel1crd3	2e+05	0.526767757823642	0.473232242176358

Table 5: Raw EAF AUC/AOC results for Dim=20 for experiment set 2

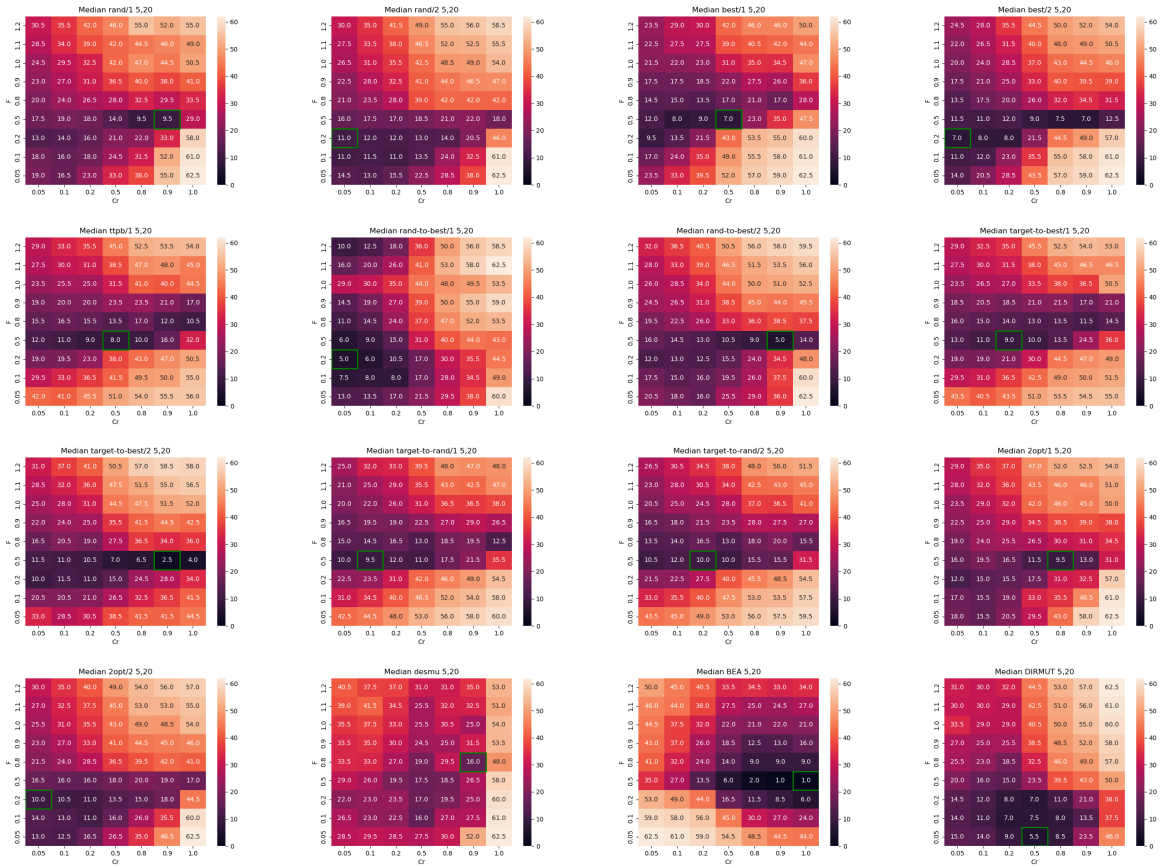

Figure 9: Heatmaps for all mutation operators for $N = 5, 20$. Values of each configuration based on average of median ranked results on 5- and 20-dimensional problems.

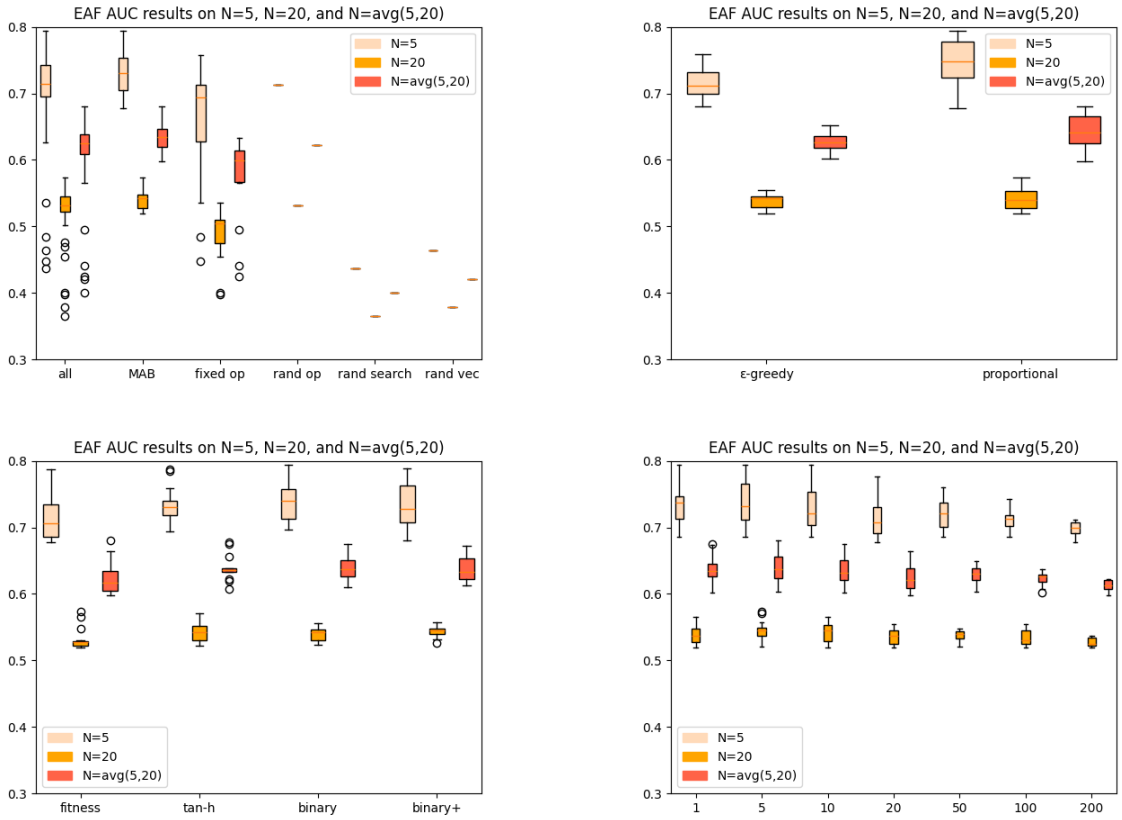

Figure 10: Boxplots of the results in Table 1. Top-left: general comparisons, top-right: selection method, bottom-left: credit assignment method, bottom-right: Learning Period